Chapter 4

EVOLUTIONARY PARADIGMS

Franciszek Seredynski
Polish-Japanese Institute of Information Technology and Polish Academy of Sciences

Abstract
In recent years, evolutionary computation (EC) techniques have became one of the most popular heuristic search methods successively applied to solve complex research and real-life problems. This chapter presents an overview of the field of EC. Main concepts of biological evolution and some biological paradigms are shown, their influence on EC is discussed, and a general computational scheme currently used in EC is presented. The best recognized classes of EC algorithms are described, such as *Evolution Strategies, Genetic Algorithms, Genetic Programming, Evolutionary Programming*, and *Learning Classifier Systems*. However, the main emphasise is on the class of Genetic Algorithms (GAs). Mechanisms of controlling evolutionary process in GAs are discussed, the most known variants of GAs are presented, and current issues of development of GAs are considered.

1 EVOLUTION, LEARNING, AND EVOLUTIONARY COMPUTATION

1.1 Lamarckian Evolution

In the nineteenth century, several theories of biological evolution were proposed and three of them are used today to different degrees in evolutionary computing (EC). These are (e.g., [31,85,102]) *Lamarckian evolution, Darwinian evolution* (Darwin, 1859), and the theory proposed by Baldwin known as the *Baldwin effect* (Baldwin, 1896). Some other concepts introduced later, such as *species*, *niches*, or *coevolution* of species, are also used in EC.

One of the first concepts of evolution was the one proposed by J. B. Lamarck. He suggested that the experience of organisms during their lifetime—their ability of adaptation—may directly influence evolution over many generations. This meant that traits such as the development of some organs (or the degeneration of others) across individuals' lifetimes to make activity more efficient, or learned behaviors such as how to avoid preditors, could be passed on to offspring by inheritance alone, and the offspring would not need to learn these traits. He believed that after some number of generations, this process could lead to the emergence of new species.

Using the notion of the *phenotype* as the observed characteristics of an organism and the notion of the *genotype* as the actual genetic structure of the organism, one can see Lamarckian evolution as a mapping from the phenotype to the genotype, where environment and individual experience directly change the individual genetical makeup. While today Lamarckian evolution is not an accepted model of biological evolution, some research in the field of EC used this model and shown that the search process may converge to a local optimum [131] or that it can sometimes improve (e.g. [1,54]) the effectiveness of an evolutionary algorithm.

1.2 Darwinian Evolution

Today the best known evolutionary algorithms (EAs) are loosely based on simulated Darwinian evolution. Darwin's theory pointed out two main factors of an evolutionary process: *natural selection* and *genetic variation*. Natural selection, which is today briefly described as the principle of survival of the fittest, states that the individuals whose variations are better adapted to the environment have a greater probability of surviving and reproducing, and selection is the mechanism that reduces the number of less-adapted individuals. Shortly after the publication of Darwin's theory, Gregor Mendel discovered the genetic basis of inheritance, and later some scientists like Hugo de Vries developed the concept of genetic mutations. Research from *genetics* shows that genes determine individual characteristics and only genes are transmitted thorough generations. The genetic material of the organisms is the result only of a continual variation of individuals, with possible influences from environmental conditions.

In contrast to Lamarckian evolution, the driving force of Darwinian evolution is a mapping from the genotype to the phenotype. This means that the environment and genetic information determine characteristics of individuals. Today the EC community interprets Darwinian evolution as a life-cycle of some organisms, as presented in Figure 4.1.

It is assumed that a population $P(t) = \{x_1^t, \ldots, x_n^t\}$ consisting of n individuals of the same species begins its life-cycle in generation t, called also an iteration t. The individuals live in some environment where they get food, struggle with illnesses, and avoid predators. At the end of their lifetime, the fitness of each individual in the population is evaluated, and this fitness is the basis to apply in some way the principle of natural selection, which gives a higher chance of survival to more fit individuals. Next, for those members of the population that survived, genetic mechanisms are applied to reproduce them by creating new individuals (offspring, children) that inherit features of their parents. As a last step of the life-cycle of the current population, it is assumed that new members of the population replace partially or fully the old members, and they are considered as members of a new population that begins a new life-cycle, termed generation $t + 1$.

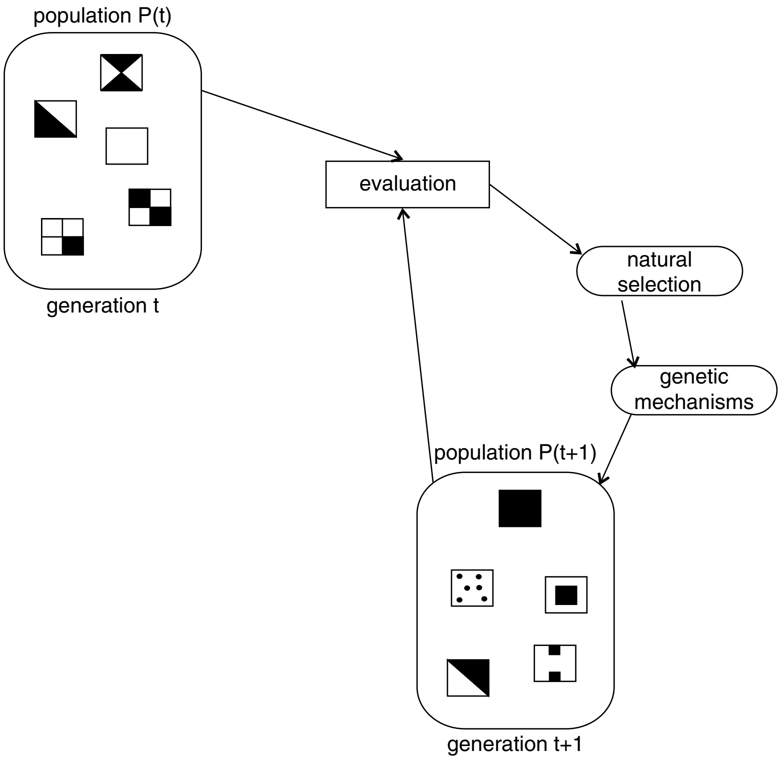

Figure 4.1. Darwinian evolution as a life-cycle of a population of individuals

1.3 Baldwin Effect

Baldwin suggested the idea that individual learning can change the course of evolution. The individual learning does not affect directly the genetic code of the individual, but individuals with increased learning capabilities may have higher probabilities to survive, which may result in an increasing number of their offspring. If, e.g., a new predator appears [85] in the environment of some species, individuals capable of learning to avoid the predator will be favored. As the proportion of such individuals in the population grows, the population will be able to support a more diverse gene pool, allowing the evolutionary process to adapt more rapidly. This may in turn enable the standard evolutionary process to more quickly evolve a genetic trait to avoid the predator. This mechanism is called the *Baldwin effect*. Some results of applying the Baldwin effect in EC show [131] that the search process converges to the global optimum, while the search process without learning converges to local optimum. Also, positive effects of the application of the Baldwin effect were observed in experiments with evolving neural networks [57] and in modeling immune systems [96].

1.4 Species and Niches

The concepts of *species* and *niches* existing in biology and ecology, respectively, has been recently recognized by EC community (see section 3.3) to create new EAs. While the process of emerging species is not fully understood, there is some agreement concerning the main principles of the development of species [31]. Biological species are recognized more by their phenotypic differences than by genetic criteria. It is assumed that interbreeding may happen only between individuals of the same species. New traits in the population may be a result only of such processes as reproduction, mutation, gene flow, and genetic drift, which are realized on genes pool of the same species. It is believed that traits favored by the mechanism of natural selection may lead to the appearance of new species.

Members of the same species occupy an ecological region called a *niche*. The ecological niche is associated with a survival strategy of the species, the environment in which the species' relations to food and enemies are established.

1.5 Coevolution

The prevailing number of EAs is based on simulations of the life-cycle of a population of a single species. The open new area of EC are EAs based on a paradigm of *coevolution* of different species (see section 3.5). Coevolution can be defined [31] as a change in the genetic composition of one or more species in response to a genetic change in another, which happens during the evolutionary process as the result of interactions between different species. Different species can occupy different niches or share the same niches. They may compete for one or more resources. In the outcomes of coevolution, different forms of coexistence between species can be observed.

When coexistence has the form of *competition* between different species, the presence of each species is associated with reducing the growth of another species. Some kind of equilibrium between species can appear. When the relation between two species is based on *exploitation*, then the presence of one species stimulates the growth of the second species, while this second species inhibits the grow of the first. This form of coexistence is based on interaction between species having a character of either a *predator–prey* interaction, resulting in the extinction of one species, or a *host–parasite* interaction, leading to two coevolving species where extinction does not take place. Most of the predator–prey forms of interaction are based on the original model of Lotka and Voltera [75,127].

If the last form of coexistence, *cooperation*, takes place between species, then the presence of each species stimulates the growth of the other species.

1.6 Evolutionary Algorithms

Evolutionary Algorithms (EAs) are search, optimization, and learning techniques based on the Darwinian concept of natural evolution and biology. Today there are several well-established streams of EAs: Evolutionary Strategies (ESs), Genetic Algorithms (GAs), Genetic Programming (GP), Evolutionary Programming (EP) and Learning Classifier Systems (LCSs) [11,35,55,71,82,90,139,137,90]. A common accepted term referring to this type of computation is *evolutionary computation* (EC). Despite the differences between these streams, which will be shown later, they all use the basic notions and mechanisms of evolution and biology, such as (1) a population of individuals, (2) the fitness of an individual, (3) the birth/death cycle of individuals, (4) inheritance, and (5) reproduction, variation, and selection/competition.

Figure 4.2 shows a general computational scheme for EAs. The scheme can also be presented with the use of the following pseudocode:

```
0. Construct a representation (an individual of a population) of a solution for a
   given problem;
   $t \leftarrow 0$
1. Randomly create an initial population $P(t)$ of individuals
2. Evaluate fitness of all individuals in $P(t)$
3. If a **termination condition** is satisfied then go to Step 5 else go to Step 4
4. Apply selection and genetic operators in $P(t)$;
   4′. *Optionally*: apply competition mechanisms;
   4″. *Optionally*: apply local search algorithms;
   $t \leftarrow t + 1$; go to Step 2;
5. *Optionally*: If **restart** then go to Step 6, else go to Step 7
6. $t \leftarrow 0$;
   Generate modified initial population $P(t)$; go to Step 2
7. Consider the best individual from $P(t)$ as a solution of the problem.
```

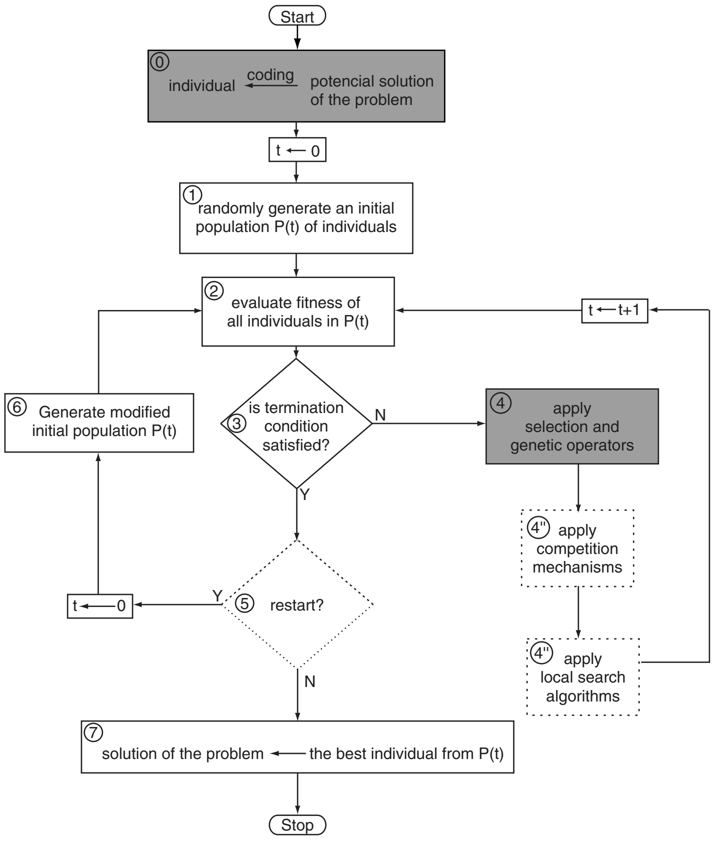
Start
0
individual
coding
potencial solution of the problem
t ← 0
1 randomly generate an initial population P(t) of individuals
2 evaluate fitness of all individuals in P(t)
t ← t+1
6 Generate modified initial population P(t)
3 is termination condition satisfied?
N
4 apply selection and genetic operators
Y
4" apply competition mechanisms
t ← 0
Y
5 restart?
4" apply local search algorithms
N
7 solution of the problem ← the best individual from P(t)
Stop

Figure 4.2. Computational scheme of an evolutionary algorithm

Steps 4′, 4″, 5, and 6 are not used in classical versions of EAs. As a *termination condition*, a predefined number of generations of simulated evolutionary process is usually used, or some more complex stopping criteria can be applied, like, e.g., "stop if the fitness of the best individual in the population has not increased during the predefined number of generations."

The main differences between the streams of EAs lie in Steps 0 and 4 of the computational scheme of EAs. Let us consider the issue of a *representation* [107] of a solution of a problem to be solved (Step 0 in the computational scheme of ES). For this purpose, it is useful to consider two separate spaces (e.g., [9]): a solution space and a search space (see Figure 4.3). If we want, e.g., to design a car with specific features, then we can imagine a space of solutions of the problem (see Figure 4.3, left) as a set of potential solutions. A single actual solution is a collection of parameters of a desired construction and is called a *phenotype*. Some EAs search for a solution directly in a solution space, i.e., in the space of phenotypes. However, some other EAs search a solution indirectly in a search space, which is constructed by mapping to it objects from the solution space. A *genotype* is an object of the search space and represents a coded version of parameters of a corresponding phenotype from the solution space. Figure 4.3 (right) shows an example of a search space of genotypes used in GAs. A binary string is a representation of a genotype. Values of a single gene called *alleles* code parameters of searched solutions. A collection of genes is called a *chromosome*. Evolving individuals from the search space requires mapping genotypes into space of phenotypes to read correctly an actual quality of a solution.

ESs differs also in Step 4, where selection and genetic operators are applied. The differences are in the type of operator, the means of their construction, and the order of their application.

2 EVOLUTION STRATEGIES

Evolution Strategies (ESs) were independently developed by Rechenberg and Schwefel in the early 1960s in Germany as a method to solve practical optimiza-

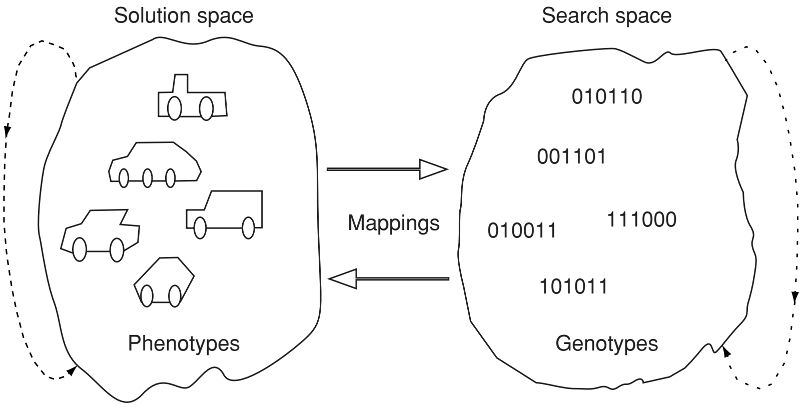

Figure 4.3. Solution space (phenotypes) and search space (genotypes)

tion problems in engineering. For continuous optimization problems, ES directly processes a real-valued n-dimensional vector $\mathbf{x}$ that is associated with the extremum of a function $F(\mathbf{x}) : R^n \to R$. It means that there is no process of coding (see Figure 4.2) of potential solutions (phenotypes in a solution space) to a problem into individuals of a population (genotypes in a search space) and that ES directly operates on phenotypes (see Figure 4.3).

A number of ESs have been developed (see, e.g., [61,104,113,90,10]). All of them are described using a specific notation, and in particular the following notation of parameters is used: μ—the size of parent population, λ—the size of offspring population, ρ—the size of family (parents) ($1 \le \rho \le \mu$). An individual $\mathbf{v}$ is a pair of float-valued vectors $\mathbf{v} = (\mathbf{x}, \boldsymbol{\sigma})$, where $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ is a point in a solution space, and $\boldsymbol{\sigma} = (\sigma_1, \sigma_2, \ldots, \sigma_n)$ is a vector of standard deviations. The earliest ES was based on a population consisting of one individual only, and was referred to as (1 + 1) -*ES*.

2.1 The (1 + 1) -*ES*

The (1 + 1) -*ES* algorithm, also called *two-member* ES, is based on a simple mutation-selection scheme. An initial population at generation $t = 0$ consists of one parent $\mathbf{v}^t$. One offspring is created by an operator of *mutation*, which adds to components of the vector $\mathbf{x}^t$ normally distributed random numbers, i.e.,

$$x^{t+1} = x^t + N(0, \sigma), \tag{1}$$

where $N(0, \boldsymbol{\sigma})$ is a vector of normally distributed (isotropic Gaussian), independent random numbers with a mean of 0 and standard deviations $\boldsymbol{\sigma}$. An explanation for this operator is the biological observation that offspring are similar to their parents and that small changes are more likely than larger ones. It is possible to use other distributions [11] such as nonisotropic Gaussian or other continuous distributions, two-point distribution in binary search spaces, or "move operators" for combinatorial optimization problems.

When the population temporarily contains two individuals, the operator of *deterministic selection* is applied. The selection operator selects the better of the two individuals, which then moves to the next generation. The algorithm is continued until a termination condition is satisfied. One can notice that the evolution process is based mainly on the mutation operator.

While it can be proved that the algorithm converges to a global optimum when $\sigma_i = const$, the algorithm may get stuck after a certain number of generations. Rechenberg observed [104] that progress in evolution exists only for a small bandwidth (evolution window) of mutation strength. He proposed a statistical inference method called *1/5-rule* to control σ during the evolutionary process. The rule says that ϕ—a quotient of a number of successful mutations (which improved the fitness of individuals) to the total number of mutations—should be equal to 0.2: if a current value of ϕ is greater than 0.2, then the standard deviation σ associated with the operator of mutation should be increased, and when ϕ is less than 0.2, the σ should be decreased. Recently some other techniques using adaptation or self-adaptation and based on statistical inference or an evolutionary approach were proposed (see, e.g., [10,11]) to control σ.

2.2 The ($\mu + 1$) -*ES*

The algorithm assumes an existing population consisting of μ individuals, where $\mu > 1$. Two individuals are selected randomly from the population to create one offspring by using an operator of *discrete recombination*. The offspring ($\mathbf{x}$, $\boldsymbol{\sigma}$) has components x_i and σ_i, which are randomly copied from parents. Next, as in (1 + 1)-ES, the operator of mutation is performed on the offspring. Finally, the operator of deterministic selection is applied, which removes from the population of the size $\mu + 1$ the least fit individual.

2.3 The ($\mu + \lambda$) -*ES*

The ($\mu + \lambda$)-ES algorithm is a natural extension of the previous one. In the algorithm, μ parents (usually $\mu \leq \lambda$) produce λ offspring. Offspring are mutated, but the mutation operator is modified by introducing an additional level, where σ is controlled by the mutation operator, instead of the internal strategy handling σ (e.g., *1/5-rule*). If ($\mathbf{x}$, $\boldsymbol{\sigma}$) is an offspring obtained in the result of the recombination operator, then the two-level mutation operator converts it into an individual ($\mathbf{x}'$, $\boldsymbol{\sigma}'$) in the following way: first, the $\boldsymbol{\sigma}$ component of the individual is modified into $\boldsymbol{\sigma}'$: $\boldsymbol{\sigma}' = \boldsymbol{\sigma} e^{N(0,\,\Delta\sigma)}$, and next the component $\mathbf{x}$ of the individual is modified: $\mathbf{x}' = \mathbf{x} + N(0,\boldsymbol{\sigma}')$, where $\Delta\sigma$ is a step-size meta-control parameter.

The temporary population of size $\mu + \lambda$ is next reduced by deterministic selection to μ best individuals. This kind of selection is called (+) selection. The algorithm can be recommended for the solution of combinatorial and discrete problems. In such cases, e.g., for the Traveling Salesman Problem (TSP), an individual is a permutation list containing a sequence of cities to be visited in predefined order, and permutation operators of mutation similar to those used in GAs are applied. Note that in all cases considered so far, ES algorithm parents survive until they are replaced by fittest offspring, and well-adapted individuals may survive forever. This feature may give rise to some disadvantages of the ($\mu + \lambda$)-ES, such as, e.g., getting stuck on a problem with an optimum that moves over time.

2.4 The (μ, λ) -*ES*

To avoid disadvantages associated with ($\mu + \lambda$)-ES, a modification of this algorithm known as (μ, λ) -*ES* was proposed. As in the previous algorithm, a μ-member population of parents produces λ offspring by means of recombination and mutation. However, the selection operator is applied only to the population of offspring, reducing it to μ parents of the next generation, and this kind of selection is called (,) selection. While the general computational scheme of ES is as shown in Figure 4.2, Step 4 of ES is presented in Figure 4.4 and contains a sequence of operators executed in the following order: recombination, mutation, and selection.

2.5 ES with Self-adaptation: (1, λ) – σSA – ES

Observation and comparison of behavior of ($\mu + \lambda$) and (μ, λ) – *ES* models show (e.g., [10,11]) that (1) if mutation strength is for some reason constant, then

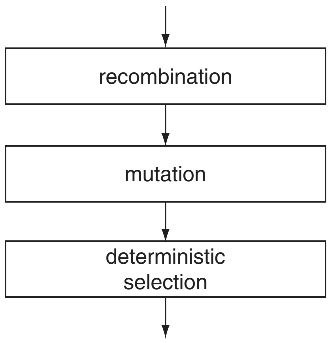

Figure 4.4. Order of selection and genetic operators applied in ESs

(,) strategies lead to a saturation behavior, and (2) if the mutation strength cannot be scaled down, the (+) strategy is always beneficent. Also, for both models, the existence of an evolution window was observed, with a value of mutation that provided evolutionary progress. This leads to the necessity of developing more general algorithms such as $(\mu \overset{+}{,} \lambda) - ES$, where both strategies can be used, or $(\mu/\mu, \lambda) - ES$ with recombination.

In [114], *contemporary* ESs were proposed. These are referred as $(\mu, \kappa, \lambda, \rho) - ES$ and allow a gradual transition from either $(\mu, \lambda) - ES$ or $(\mu, \lambda) - ES$ by introducing a lifespan parameter κ—the upper limit for life span ($\kappa \geq 1$), $\lambda \geq \mu$ if $\kappa = 1$; and ρ—the number of ancestors for each descendant ($1 \leq \rho \leq \mu$). The (,) and (+) strategies can be used depending on the value of κ.

The more general solution for controlling mutation rate and its scalability are ESs with self-adaptation, such as, e.g., $(1, \lambda) - \sigma SA - ES$ [10,11]. Each individual in the algorithm includes object parameters and evolvable (*endogenous*) strategy parameters. Endogenous strategy parameters control the variation of the individual's object parameters by mutation. These are inherited together with the object parameters.

2.6 Advanced ES Techniques

A number of advanced ES techniques are currently under study [10,11]. The general $(\mu/\rho \overset{+}{,} \lambda) - ES$ algorithm uses the μ/ρ recombination and both (,) and (+) strategies. The recombination is applied to ρ ($\rho \leq \mu$) parents. The ρ parents (also called a *ρ–family*) produce one offspring. If $\rho < \mu$, then the members of the ρ–family are chosen randomly from the set of μ parents, and if $\rho = \mu$, then all μ parents are involved in the process of creating a child. The recombination is applied to the object parameters and can be applied also to endogenous strategy parameters.

The Meta-ES (or hierarchically organized ES) $[\mu'/\rho' \overset{+}{,} \lambda' (\mu_i/\rho_i \overset{+}{,} \lambda_i)^\gamma] - ES$, $i = 1, 2, \ldots, \lambda'$, gives a possibility of mixed structural and parameter optimization. There are λ' populations in the algorithm, and γ (the exogenous strategy parameter) sets an isolation period time between them. The outer $[] - ES$ (structure evolution) improves parameters of the inner $() - ES$ (parameter evolution) populations.

Another approach to construct adaptive ES search techniques to control the variation operators (mutation, recombination) is based on use of statistical information. While the 1/5-rule (see, section 2.1) is the simplest example of such a technique, currently more advanced algorithms are used, such as the *Cumulative Step-size Adaptation* (CSA) algorithm or the *Covariance Matrix Adaptation* (CMA) algorithm.

3 GENETIC ALGORITHMS

Genetic Algorithms (GAs)[34, 46, 81] were originally developed in the late 1960s at the University of Michigan by John Holland and his team, who conducted their research on robust, adaptive systems. Later, GAs were refined by De Yong, Goldberg, Michalewicz, and many others. While the computational scheme of GAs is as shown in Figure 4.2, GAs distinctively differ in Steps 0 and 4 from the other EAs, in particular from ESs, in the following ways: (1) a search space of genotypes is used, and a binary string is a representation of a genotype, as shown in Figure 4.3 (Step 0 in Figure 4.2), and (2) the sequence of selection and genetic operators is usually, as shown in Figure 4.5, performed in the following order: *stochastic selection*, *crossover* (corresponding to operator *recombination* in ESs), and *mutation* (Step 4 in Figure 4.2).

Overviews concerning current issues on GAs can be found in [133, 55, 35].

3.1 Simple Genetic Algorithm

Selection and genetic operators of a Simple Genetic Algorithm (SGA) [46] have the following properties: (1) *proportional selection* is used, alternatively called a selection with a *roulette wheel*, (2) a *single-point crossover* is performed on each chromosome, with a probability p_c, and (3) a *single-bit mutation* is performed with a probability p_m. These selection and genetic operators are shown in Figure 4.6.

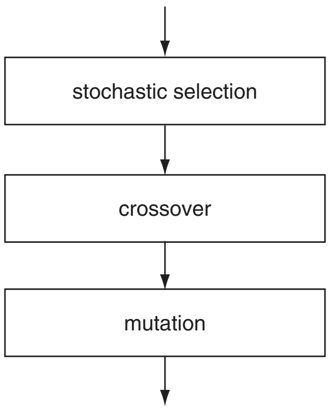

Figure 4.5. Order of selection and genetic operators applied in GAs

The proportional selection operator provides a method of stochastic selection that selects an individual *i* (to survive and have a chance for mating) with a probability

$$p_i = \frac{f_i}{\sum_{j=1}^{n} f_j} \tag{2}$$

proportional to its fitness f_i, where *n* is the size of the population. Assuming that there are five individuals in a population with fitness 27, 45, 11, 5, and 32, respectively, one can construct the roulette wheel with five areas (see Figure 4.6a), each corresponding to a single individual, that are proportional to their probability of selection. A single spin of the wheel results in a selection of one individual corresponding to the area of the roulette pointed to by a pointer when the wheel stops spinning. The selection is completed after spinning the wheel five times.

Each individual that passes selection is chosen next, with a probability p_c for mating. A single-point crossover is performed on two parent individuals (see Figure 4.6b). A random position is chosen in both chromosomes, and two children individuals are produced after the exchange of genetic material. The choice of which parent contributes the bit for a given position of children can be also determined by an additional string called the *crossover mask*. For the crossover shown in Figure 4.6b, the crossover mask is 111100. The 1s and 0s of the crossover mask define the contribution of bits of parent *P*1 and parent *P*2, respectively, to the child *Ch*1. The second child uses the same mask but switches the roles of the two parents.

After crossover, a genetic mutation operator is performed. Each locus of each chromosome of the population is selected for mutation, with a probability p_m.

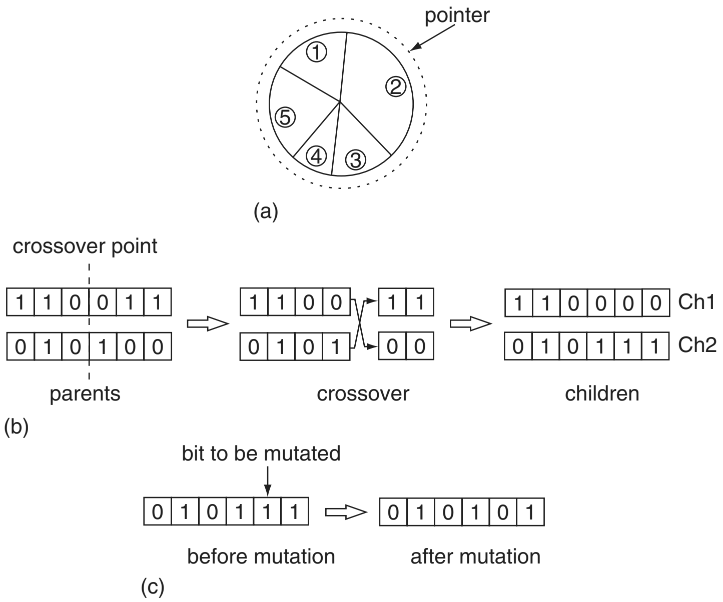

Figure 4.6. Operators of selection, crossover, and mutation used in SGA: (a) roulette wheel (proportional selection), (b) single-point crossover, (c) single-bit mutation

If mutation takes place, a corresponding bit switches its value either from $0 \rightarrow 1$ (see Figure 4.6c) or from $1 \rightarrow 0$.

SGA has good theoretical foundations [128], but most applications do not use it. The main problems with SGA are the following: (1) premature convergence to a local optimum, because of sensitivity of such parameters as population size or crossover/mutation rate, (2) strong convergence to the wrong solution for the problems known as *deceptive problems* [46], and (c) possible poor representation and poor operators for some problems. For these reasons, a number of extensions of SGA have been proposed, and for practical applications, customized/hybrid GAs with domain-dependent representations and operators are frequently designed.

3.2 Mechanisms to Control Evolutionary Process in GAs

Representations and Encoding

Representation is one of the key issues influencing the performance of EAs used as optimizers. As mentioned earlier, when the problem of optimization with EA use is considered, it is useful to consider two spaces: the phenotype space representing the original definition of the problem and the genotype space representing encoded solutions. The purpose of representation is to assign genotypes to corresponding phenotypes [102, 107], which is often called *genotype–phenotype mapping*. This mapping can be done in different ways. It influences the suitability of applied genetic operators and the performance of the evolutionary process.

The most common representation applied in GAs is a *binary* string used, e.g., in SGA. In some problems binary encoding is natural, e.g., for the 0-1 *knapsack* problem that is very well known in operational research. In such cases there is no distinction between genotype and phenotype spaces. The same result takes place when *continuous* encoding (similar to that used in ESs) with real-valued vectors is used for function optimization problems. For some discrete nonbinary problems, e.g., the *rotor stacking problem* [102], a discrete alphabet of higher cardinality might be appropriate.

One of the problems that may arise when the standard binary code is used is that adjacent genotypes may not have adjacent phenotypes. In such cases, the use of *Gray code* may be preferable and more effective. Some recent research [132, 133] shows that Gray encoding reduces the number of local optima, which is important for local search algorithms, and beats binary encoding in many test problems.

For some, problems (e.g., TSP, flowshop problem, multiprocessor scheduling), natural encoding is a *permutation* representation. For these problems, special crossover operators must be designed because the standard crossover operators usually fail to preserve the permutation. For other problems, e.g., *tree optimization problems*, a graph can be represented by its characteristic vector.

Incorporating problem-specific knowledge in the representation can increase [107] the GA performance. This can be done in particular by (1) considering specific properties of the optimal solutions, e.g., trees and stars, and (2) delivering to the population solutions that are similar to the optimal solution.

Currently, analytical models do not exist that describe the influence of representation on the performance of EAs, but there are some general recommendations [107]. Goldberg's recommendations [46] are based on (1) the principle of meaningful building blocks and (2) the principle of minimal alphabets, and are oriented on effective processing schemata in GAs. Radcliff [101] suggests that representation and operators cannot be considered separately from each other and makes some recommendations [116] on how to design representation-independent EAs. According to Palmer [92], (1) encoding should represent all possible phenotypes, and all possible individuals should be equally represented in the set of all possible genotypic individuals, (2) encoding should encode no infeasible solutions, should possesses locality and be adjusted to a set of genetic operators, and should minimize nonlinearities in fitness functions, and (3) decoding of the phenotype from the genotype should be easy.

One of the open issues in current research on representation in GAs concerns the use of redundant representation (see, e.g., [106]), where the number of genotypes is larger than the number of phenotypes.

Population Manipulation

Population size and the individual replacement strategy are very sensitive parameters [102, 55] of GAs. An analysis performed by Goldberg and his colleagues [48] suggested a linear dependence of population size on string length. However, some empirical results (see, e.g., [112]) show that population sizes as small as 30 are adequate in many cases, e.g., for binary-encoded problems, but for higher-cardinality alphabets, much larger populations are needed. For a minimum population size, a principle was suggested in [100] that every point in the search space should be reachable in the initial population by crossover operation only. Some reports also show (e.g., [2]) that including in the initial population some good-quality solution obtained from another metaheuristic can improve the performance of GAs but may also lead to premature convergence to a poor solution.

The most traditional individual replacement strategy used in GAs is a *generational* reproduction: a current population is completely replaced by offspring generated by selection and genetic operators. Such a replacement strategy is used in SGA and most GAs. De Yong proposed [34] a simple strategy called *elitism*, which ensures the survival of the best individual in the population, and the concept of *population overlaps*, which assumes replacing only a fraction *G* (*generation gap*) of the population in each generation. He also introduced a *crowding* operator, which specifies the number of individuals initially selected as candidates to be replaced by a newly generated offspring. A new offspring replaces the most similar individual, where the similarity measure can be, e.g., the *Hamming distance* between individuals.

The opposite strategy is assumed by *steady-state* reproduction. In each generation, only one or two individuals are created, which replace the worst individuals. This strategy can be modified [102] in such a way that candidates for replacement are chosen from those worse than the median. Another modification of this strategy, useful for optimization in dynamic environments [15], suggests replacing [125] the *oldest* instead of the worst individual but not replacing it [122] when it is currently the best in the population.

Selection

The selection operators used in GAs operate on the fitness of individuals. The mechanism of the *roulette wheel* used in SGA to implement *proportional selection* can be changed by introducing an n-armed spinner (n is the size of the population) [102] providing *stochastic universal selection,* an effective method of implementating proportional selection. The *scaling* problem associated with roulette-wheel selection—i.e., when values of individual fitness in subsequent generations become less distinguished and *selection pressure* becomes weaker—can be solved by a number of algorithms proposed in [46].

Ranking selection does not need scaling and is more efficient. It sorts individuals in each generation according to their decreasing/increasing fitness and assigns new values of selection probability. Selection probabilities assigned to ranked individuals can increase linearly or nonlinearly, creating in this way a *linear ranking* selection or a *nonlinear ranking* selection, respectively. A number of algorithms exist [4, 81] for assigning linear or nonlinear probabilities to ranked individuals. The algorithms provide a possibility for control of the selection pressure.

Another effective and simple selection operator is *tournament selection*. It requires choosing k individuals from a population (often $k = 2$), comparing their fitness, and selecting the most fit as the winner of the tournament. A variant of tournament selection is called *strict* tournament, and it has similar properties to ranking selection. To provide higher selection pressure, yet another version of tournament selection is used, which is called *soft* tournament. In this case, the winner of the selection is accepted with some predefined probability.

In *truncation selection*, some percentage of the best individuals of a population is selected, and parents from this selected subset only are chosen randomly for mating. A comparison and theoretical analysis of selection schemes can be found in [13].

Search Operators

Crossover

Two sources of bias exist [102] that can be exploited in GAs by crossover operators: *positional bias* and *distributional bias*. From this point of view, much empirical evidence supports the opinion that one-point crossover is not the best crossover construction. *Two-point crossover* and generally *multipoint crossover* are logical extensions of the one-point crossover. In the two-point crossover, offspring are created by substituting intermediate segments of one parent into the middle of the second parent string. The intermediate segment is represented in the crossover mask by a contiguous block of 1s with the borders of the segments created randomly. For the crossover mask 001110, two-point crossover will create offspring 110101 and 010010.

The crossover operator that removes any bias is *uniform crossover*. It combines bits sampled uniformly from two parents. For the crossover mask with random string of bits 101100, two identical offspring are created: 110000. When the mask is created using a Bernoulli distribution, this uniform crossover is referred as UX.

As presented above, crossover operators are suitable for problems with binary representation. However, for problems with permutation representation, they can produce infeasible solutions and therefore cannot be applied. For permutation problems such as TSP, sequencing, or scheduling, a number of nonlinear crossover operators have been constructed. Among these, the best known are (1) PMX (Partially Mapped Crossover), which exchanges a partial segment between parents, (2) CX (Cycle Crossover), which finds all mapping cycles between parents and next copies elements of the two parents to the offspring in corresponding positions, (3) OX (Order Crossover), which randomly selects several of the same elements in both parents and next makes exchanges between parents in those selected positions, (4) HUX crossover, a variant of uniform crossover in which exactly half the bits are exchanged, (5) the *edge recombination* crossover, and others [81]. Recently, the *edge assembly* crossover [89] was proposed and applied successfully for solving TSP problem for more than 3000 cities.

Mutation.

Mutation in GAs is usually considered as a secondary genetic operator. The purpose of mutation is to introduce some randomness into the search and to prevent the optimization process from getting stacked into local optima. Most variants of GAs apply mutation with a constant low rate, e.g., 0.005. Some research used higher mutation rates ranging from 0.001 up to 0.01, but it was found that higher mutation rates may transform the optimization process performed by the GA into a random search process. The appropriate value of the mutation rate of the GA for a given optimization problem is an open research issue.

The formula for the near optimal value of the probability of mutation p_m for a set of test functions was found experimentally [112] to be

$$p_m = \frac{1.7}{n\sqrt{l}}, \tag{3}$$

where n is the population size and l is the length of a chromosome. However, theoretical analysis conducted in [87] using the ONEMAX function shows that the optimal mutation rate for any unimodal binary function is approximated by the formula

$$p_m = \frac{1}{n}. \tag{4}$$

Results of research presented in [7] suggest that the mutation rate should change dynamically during the evolutionary searching process in the following way:

$$p_m(t) = (2 + \frac{l-2}{T-1}t)^{-1}, \tag{5}$$

where t is the current generation and T is the total number of generations. These results show that the mutation rate probability should change from an initial value of 1/2 to $1/l$. In [91] it was shown that the value of optimal mutation rates in GAs differs according to whether recombination is used or not. A new mutation operator proposed in [22] and named *minimum-allele-reserve-keeper* ensures a minimum amount of each allele in each locus with the least possible amount of gene inversion. In [33], a nature-based mutation operator called the *frame-shift* is proposed.

A number of novel GAs with new mutation operators have been proposed recently. In [58], a *parental mutation GA* is proposed in which mutation occurs not only in offspring but also at the parental level. In [124], a novel genetic algorithm named the *Split Search GA* was proposed to fully utilize the mutation operator. Experimental results using this algorithm show that increasing the role of mutation in the evolutionary search may be beneficial.

Competition Mechanisms

A typical competition mechanism that can be applied (see Figure 4.2) is *elitism* (see section 3.2). It assumes always keeping the best individual in the current population to replace the worst individual in the next generation, if the individual in the next generation is worse that the best one in the previous population. Another competition mechanism is applying *truncation* selection (see section 3.2) to the population of parents and children.

Local Search Algorithms

Local search algorithms are based on the idea of iterative improvement of a current solutions and are often used in GAs. A number of local search algorithms have been developed. The best known of these are the following:

- *next ascent bit-climbing*: a *flip-list* describing the order of flipping of a selected chromosome is created, and (1) the bit of the chromosome corresponding to the actual position on the flip-list is flipped, (2) the flip is accepted if the new string has a higher fitness than before flipping, and (3) after flipping all bits according to the flip-list, the string with the highest fitness is considered to be a solution
- *steepest ascent bit-climbing*: as in the next-ascent algorithm, bits of a chromosome are flipped in a predefined sequence; however, after flipping a current bit according to the flip-list, (1) the remaining sequence of bits is sequentially flipped; for each new string, fitness is evaluated, and after that the flip is removed, (2) the bit (from the sequence of flipped bits) that obtained the highest fitness of the chromosome is accepted, and (c) the procedure is continued for the next bit from the flip-list
- *random bit-climbing (RBC)* [28]: similar to the next-ascent algorithm, but the flip-list is defined as a permutation of bits' positions in the chromosome; permutations are generated until no improving flips are found
- Lamarckian evolution [131] (see section 1.1)
- Baldwin effect [131] (see section 1.3)
- *(1+1)-ES* [37] (see section 2.1)
- *random mutation hill-climbing* or random local search [83, 129], which works like a simple standard evolutionary algorithm (EA), which is mutation based and works with population size 1 (referred to as *(1+1)-EA*) but with a different mutation operator
- *Random Walk with Uniform (or Normal) Distribution* [37]

Recently some other local search algorithms have been proposed. These are (1) the *quad search* algorithm [130], a specialized form of steepest ascent that operates on a reduced neighborhood and uses a Gray encoding, (2) *consecutive* *exchange* [23], a modification of the *2-Opt* heuristic combined with *tabu search*, and (3) a local search strategy based on the idea of iterative improvement of a solution via a series of neighbor moves until no improvement can be made [62].

Restart

One method to prevent GAs from prematurely converging to local optima before discovering a global solution is periodically restarting GA (see Figure 4.2) according to some restart strategy that can be either *static* or *dynamic* [42]. The restart can be performed with the use of a new seed. When GAs are applied in dynamic environments [15], the important issue is the content of the initial population of GA after restarting. The new initial randomly created population usually contains some percent of the population from the previous run.

3.3 Variants of GAs

SGA with *elitism.*

This is one of the simplest extensions of SGA (see Figure 4.2).

Hybrid GAs

This variant includes a wide range of GAs currently used or proposed that apply local search algorithms (see section, 3.2 and Figure 4.2).

Genitor [134]: A Steady-State GA

This algorithm uses rank-based selection and a *steady-state* strategy for reproduction. In each generation, only one (or two) individuals are created. Two-point crossover with reduced surrogates is used to produce a pair of children, and then one of them is selected randomly for mutation. The offspring displace the worst individual in the population.

CHC [36]

A fixed population of size 50 is used. Members of the population are paired randomly, and only parents sufficiently different are mated. As crossover, a reduced surrogate HUX is used, and no mutation is applied to offspring. Truncation selection is used and restart is applied, in which a new population is created by using the best solutions from the previous population with 30% mutation.

GENOCOP [81]

This variant is a GA-based hybridized evolutionary system used in several versions for solving constrained optimization problems. Real-numbers representation and a number of crossover and mutation operators are employed.

Breeder GA [88]

Breeder GA is an SGA-style algorithm developed to solve continuous problems directly, without the need for a discrete genotype. The parameters of the algorithm that control the evolutionary process are (1) population size, (2) mutation rate, and (3) selection intensity. Mutation is performed at a rate of $1/l$ (l is string length), which is claimed to be optimal. A form of truncation selection used in breeding is applied, namely, the best individual takes place in all matings.

GAVaPS [81]

Population size in the algorithm can vary during the evolutionary process, and it depends on the *age* of an individual, which is its parameter. An individual that exceeds its *lifetime* is eliminated from the population. The value of lifetime is determined at each generation, and it depends on some population statistics.

Niching Algorithms

In many applications such as multimodal or multiobjective optimization, dynamic function optimization, or machine learning, the important issues are (1) the maintenance of diversity of a population and converging to different solutions, and (2) preventing premature convergence when only one solution is required. These issues are addressed by *niching algorithms*. One of the first mechanisms introduced to support diversity is *crowding* [34], which assures that new individuals replace similar individuals in a population. The mechanism of *fitness sharing* [46] forces similar individuals to share their fitness. The *mating restriction mechanism* [32] prevent recombination between individuals in different niches. *Deterministic crowding* [76] modifies crowding by a mechanism of minimizing the sum of parent-to-offspring distance. Recently the mechanism of *probabilistic crowding* [80] was proposed and a concept of *niching pressure* was introduced [115] and used in the context of agent-based EC systems.

Multiobjective Evolutionary Algorithms

Multiobjective Evolutionary Algorithms (MOEAs) are one of the current trends in developing EAs. An excellent overview of current issues, algorithms, and existing systems in this area is presented in [24,25].

Evolutionary Optimization in Dynamic Environments

EAs for time-varying environments are the subject of current study within the area of EC. Different aspects of dynamic optimization problem are discussed in [15], along with evolutionary concepts to solve these problems.

Parallel GAs

See section 3.4.

Coevolutionary GAs

See section 3.5.

Competent GAs

See section 3.6.

3.4 Parallel GAs

Single-population Master-Slaves Model

The *single-population master-slaves model* [18,51,61] offers the easiest and simplest way of parallelizing single-population GAs and is presented in Figure 4.7a. A master-processor runs the GA performing selection and genetic operators.

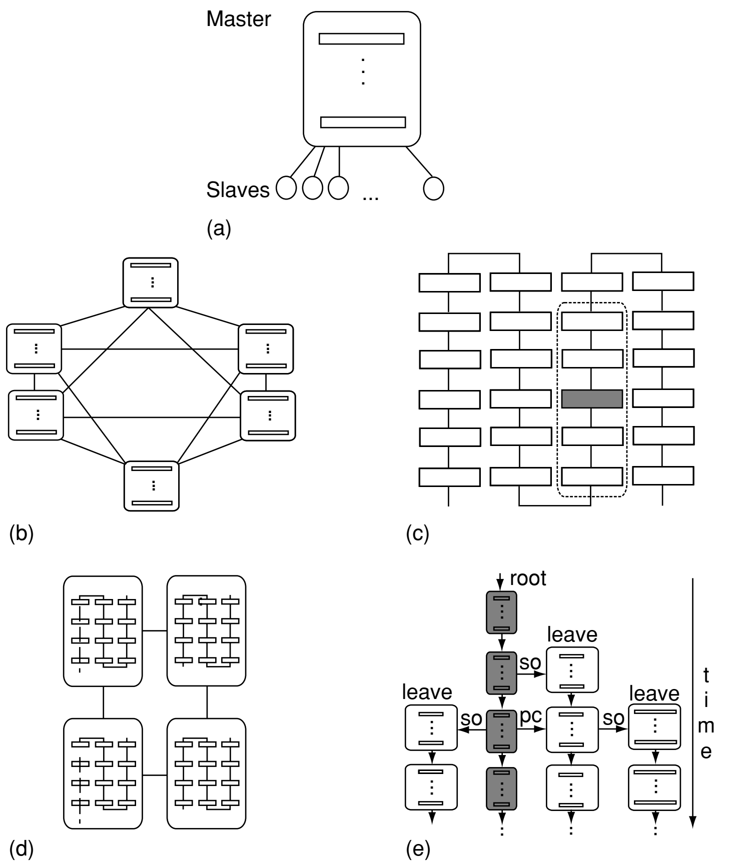

Figure 4.7. Types of parallel GAs: (a) single-population master-slaves model, (b) island model, (c) diffusion model (cellular GAs, fine-grained parallel GAs), (d) hierarchical parallel GAs, (e) hierarchical genetic strategy

The operation of fitness evaluation is parallelized. Due to this parallelization, the fitness of individuals in the population can be calculated *in parallel* by slave-processors, if a corresponding number of processors is available.

Island Model

In the *island model*, also called the *migration* or *multiple population model* [18,19,119,77,61], a population of GA is divided (see Figure 4.7b) into some number of subpopulations, called *islands* or *demes*, which are located on different, usually MIMD-class processors. Each subpopulation is a complete GA and evolves *in parallel*, exchanging periodically with other subpopulations the best individuals. A population structure composed of subpopulations is defined by the topology of a communication graph, which specifies a neighborhood for each subpopulation and serves to exchange individuals between neighbor subpopulations. Both the number of subpopulations and the topology of the communication graph are user-defined parameters.

Each subpopulation runs during some predefined number of generations called an *epoch*, in the same way as it is done in the one-population GA. After each *epoch*, neighbor subpopulations communicate by exchanging some number of their best individuals, which migrate and then are assimilated into subpopulations in such a way as to maintain a constant size of the subpopulation. The *island model* outperforms the one-population GA, providing a nearly linear speed-up when parameters of the algorithm are well tuned. Current modifications of the model introduce a variable subpopulation size of each island, different lengths of chromosomes in subpopulations, and asynchronous interactions between islands.

Diffusion Model

In the *diffusion model*, also called the *neighborhood*, *fine-grained*, or *cellular model* [18,119,97,61], each individual from a GA population is placed into a single processor, typically of the SIMD class. A neighborhood relation is set between all individuals by considering each individual as a node of a user-defined communication graph, which can be linear (see Figure 4.7c) or planar. Additionally, a local neighborhood of each individual is specified (see dotted area for the shadowed individual in Figure 4.7c). All selection and genetic operators are defined locally on such (possibly overlapping) neighborhoods. Different local mating strategies [50] and different neighborhood sizes and shapes [109] can be used.

Hierarchical Parallel GAs

Hierarchical parallel GAs are the result of an attempt to integrate the advantages of the master-slaves, island, and diffusion models [18]. In [73], the island model was combined with the diffusion model to solve the scheduling problem efficiently, and the idea of such a hierarchical model is presented in Figure 4.7d. A hierarchical model combining both the island and master-slaves models can be found in [46].

Hierarchical Genetic Strategy

Recently, in [110], a parallel GA referred as *hierarchical genetic strategy* has been proposed. It is a variable-length chromosome multipopulations GA model with a number of subpopulations changing dynamically in time (see Figure 4.7e). The algorithm starts from a single population (root) of chromosomes of the same length. During the evolutionary process, new subpopulations (leaves) can be created by using two operators: a *prefix comparison* operator and a *sprouting* operator. The root population can create a new subpopulation by using the sprouting operator when a promising individual appears, and this will be detected by the comparison operator. The new subpopulation always contains chromosomes with increased length and runs in parallel with the root subpopulations. The process of creating new subpopulations can be continued by both root and leaves subpopulations. The stop criterion of a running subpopulation is a stagnation of evolutionary process in a subpopulation.

3.5 Coevolutionary GAs

The idea of *coevolutionary algorithms* comes from the biological observation that coevolving some number of *species*, defined as collections of phenotypically similar individuals, is more realistic than simply evolving a population containing representatives of one species. So, instead of evolving a population (global or distributed) of similar individuals representing a global solution, it is more appropriate to coevolve subpopulations of individuals representing specific parts of the global solution. Four coevolutionary algorithms, presented below, depict specific lines of the research currently conducted in this area.

Coevolutionary Genetic Algorithms

The *Coevolutionary GA* [93,94], described in the context of the constraint satisfaction problem and the neural network optimization problem, is based on a *predator–prey* paradigm [56]. The algorithm operates on two subpopulations: the main subpopulation $P^1()$, containing individuals $\overline{x}$, and an additional subpopulation $P^2()$, containing individuals $\overline{y}$ coding some constraints, conditions, or simply test points concerning a solution $\overline{x}$. Both or only one subpopulation evolves to optimize a global function $f(\overline{x}, \overline{y})$.

A single act of coevolution is based on the independent selection of individuals $\overline{x}$ and $\overline{y}$ from subpopulations in order to encounter them and evaluate their $f(\overline{x}, \overline{y})$. The manner of assigning fitness to the individuals stems from the predator–prey relation: success of one individual should mean failure of the second one. During one generation, individuals are confronted a predefined number times. At the end of the evolution process, the best individual from P^1 () is considered to be a solution of a problem.

Cooperative Coevolutionary Genetic Algorithms

The *Cooperative Coevolutionary GA* (CCGA) has been proposed [98] in the context of a function optimization problem and is one of the best-known coevolutionary algorithms. Each of N variables x_i of the optimization problem is considered as a species with its own chromosome structure, and subpopulations for each variable are created. A global function $f(\overline{x})$ is an optimization criterion. To evaluate the fitness of an individual from a given subpopulation, it is necessary to communicate with selected individuals from all subpopulations.

In the initial generation ($t = 0$), individuals from a given subpopulation are matched with randomly chosen individuals from all other subpopulations. The fitness of each individual is evaluated, and the best individual in each subpopulation is found. The process of *cooperative coevolution* starts from the next generation ($t = 1$). For this purpose, in each generation a cycle of operations is repeated in a round-robin fashion. Only one current subpopulation is active in a cycle, while the other subpopulations are frozen. All individuals from the active subpopulation are matched with the best values of the frozen subpopulations. When the evolutionary process is completed, a composition of the best individuals from each subpopulation represents a solution of a problem. The algorithm has been successfully used in different applications (e.g., [65]).

Loosely Coupled Genetic Algorithms

The *Loosely Coupled GA* (LCGA) [117,119] is a coevolutionary algorithm exploring a paradigm of *competitive coevolution* and motivated by noncooperative models of game theory.

For an optimization problem described by some function (a global criterion) of *N* variables, local chromosome structures are defined for each variable, and local subpopulations are created for each of them. With each subpopulation, a locally defined function is associated, if possible, that describes relations between the variable associated with the population and other variables and subpopulations. This relation is described by a communication graph called a *graph of interaction*. While the purpose of each subpopulation is to optimize own local function under constraints defined by the influence of other local variables, an optimization of a global criterion is expected as the result of achieving by subpopulations some equilibrium, equivalent to a Nash equilibrium point in noncooperative models of game theory. If local functions are not known, the subpopulations directly optimize the global criterion.

The LCGA works in such a way that after initialization of subpopulations, each subpopulation performs in parallel the same set of operations in each generation. Each individual in a subpopulation is matched with randomly chosen individuals from subpopulations according to the interaction graph, and its fitness is calculated according to a local (or global) function assigned to a subpopulation. This matching is repeated for each individual a predefined number of times. Next, standard GA operators are applied locally in subpopulations. The evolutionary process is continued for a predefined number of generations until the system achieves the state of equilibrium equivalent to a Nash equilibrium point.

LCGAs have been applied to solve the *multiprocessor mapping and scheduling* problem [118] and the function optimization problem [120].

Coevolutionary Distributed Genetic Algorithm

The *Coevolutionary Distributed Genetic Algorithm* (CDGA), described [63,79] in the context of integrated manufacturing planning and scheduling, combines features of diffusion models with coevolutionary concepts. *N* coevolving species with their own genotypes represent partial solutions to a problem, e.g., plans for a particular component to be manufactured in a machine shop. The quality of each partial solution can be evaluated by a local function. The challenge is designing an optimal schedule to minimize the total cost of executing, in parallel, a set of plans represented in a given subpopulation. A global measure of the performance of a given plan, executed in parallel together with all plans from a population, is a global function taking into account a possible conflict in the use of common resources in the machine shop, and resolved by a local arbitrator.

A population of the CDGA is composed of subpopulations occupying a predefined number of cells arranged in some user-defined topological structure, e.g., a toroidal grid. In each cell, there are single representatives (individuals) of each species and also an individual representing an arbitrator. Only individuals of the same species from neighborhood subpopulations take part in the breeding.

Coevolution, i.e., an influence of another species on a given species, is taken into account by calculating a value of a global function. An offspring that is a result of breeding in a given local neighborhood replaces an individual in this neighborhood.

3.6 Competent GAs

A theoretical explanation of the work of SGA and a number of its extensions is based on the Holland's concept of *building blocks* (BBs) [46]. According to this concept, to find a global optimum of a problem GA requires identifying and grouping together partial solutions-schemata (BBs) with above-average value of fitness. For many hard optimization problems such as permutation problems, GAs and especially SGA have a problem doing that. These problems are frequently modeled by designing hard multimodal optimization problems called *deceptive problems* (see, e.g., [46,69])—combinations of *deceptive subfunctions* that mislead GAs to converge to a global optimum. Related to the deceptive problems is the *linkage problem*, which states that no fixed operators of recombination are able to provide mixing individuals with arbitrary codes to obtain proper BBs.

One possible solution to deceptive-like problems is to apply problem-specific coding and operators. A more general approach is to design more flexible and powerful GAs, which are referred to as *competent GAs* [47,69]. A number of competent GAs have been developed, and all of them fall into one of two classes [69]: (1) algorithms (the *fast messy GA* [49], the *gene expression messy GA* [5], and the *linkage learning GA* [52]) based on evolving the representation of solutions or adapting recombination operators, and (b) algorithms (the *extended compact GA* [53] and the *Bayesian optimization algorithm* (BOA) [95]) based on extracting information from a set of promising solutions.

Fast Messy GAs and OmeGA

In messy GAs [47,69], the genes (messy genes) of a chromosome (messy chromosome) are represented by a pair of number (*gene locus, gene value*). For example, the chromosome ((2 0)(4 1)(1 1)(3 0)(5 1)) represents the binary string 10011. Messy chromosomes may have different lengths, and they may be *underspecified* or *overspecified*. As in SGA, selection and genetic operators are used. However, the traditional crossover is replaced by *cut and split operators*.

In the fast messy GA (*fmGA*), two loops—outer and inner—are performed. In each cycle of the outer loop, three phases of the inner loop are performed. In the first, *initialization phase*, a population of individuals containing all possible genic and allelic combinations is created. In the second phase, called the *building-block filtering phase*, the population is filtered in such a way as to contain a high proportion of gene combinations belonging to BBs. In the *juxtapositional phase*, tournament selection and genetic operators are applied to form a high-quality solution.

To apply *fmGA* for solving permutation problems, the definition of messy gene is modified: a *random key* (real random number) instead of a binary digit (for gene value) is used, and such a extension of the algorithm is called *ordering* *messy GA* (*OmeGA*) [69]. For the TSP with five towns, a possible genotype may look like ((1, 0.26)(2, 0.22)(3, 0.72)(4, 0.19)(5, 0.20)). After sorting keys, the following phenotype is decoded: ((4, 0.19)(5, 0.20)(2, 0.22) (1, 0.26) (3, 0.72)), which corresponds to the permutation (4 5 2 1 3). One can easily check that the traditional single-point crossover operator will always generate feasible offspring when random key vectors are used.

Gene Expression Messy GA

The overall organization of the *gene expression messy GA* (*gemGA*) [5,47] is similar to that in *fmGA*, but the representation and the basic mechanism of the algorithm are different. The *gemGA* has no variable-length chromosomes and no under- or overspecification, and genes are stored in regular arrays. As was the case of *fmGA*, the main purpose of the *gemGA* is to determine the linkage groups, and the most important innovation of the algorithm to do that is the idea of *transcription* or *antimutation*. During the one-bit perturbation of each string, the perturbations that improve the structure are ignored and perturbations that degrade the structure are selected as possible linkage group candidates for subsequent processing.

3.6.3 Linkage Learning GA

In the *linkage learning GA* (*LLGA*) [52,47], the main concepts of the organization and the messy representation of chromosomes are similar to those in *fmGA* except that chromosomes have a circular structure. The main innovation of this messy algorithm is the mechanism called *probabilistic expression*, which reorders chromosomes in such a way as to detect important BBs in the encoding. The *extended compact GA* [53] is a more efficient version of the *LLGA*.

3.6.4 Bayesian Optimization Algorithm

The *Bayesian optimization algorithm* (*BOA*) [95,47] is a messy GA that identifies linkage-like data in a population through the construction of Bayesian networks. Traditional selection operators (truncation and tournament) are applied to choose a subset of solutions in the population that is used to construct a good Bayesian network modeling that subset. The probabilistic model corresponding to the structure of the Bayesian network is used next to generate a new population.

4 GENETIC PROGRAMMING

Genetic Programming (GP) is an evolutionary optimization technique proposed by Koza [70]. The general computational scheme of EA presented in Figure 4.2 is still valid for GP, but the main differences from other evolutionary techniques concern (1) a representation of a solution (Step 0), and (2) the order of selection and genetic operators (Step 4). Solutions are represented by trees (see Figure 4.8), which provide a flexible way of describing computer programs in

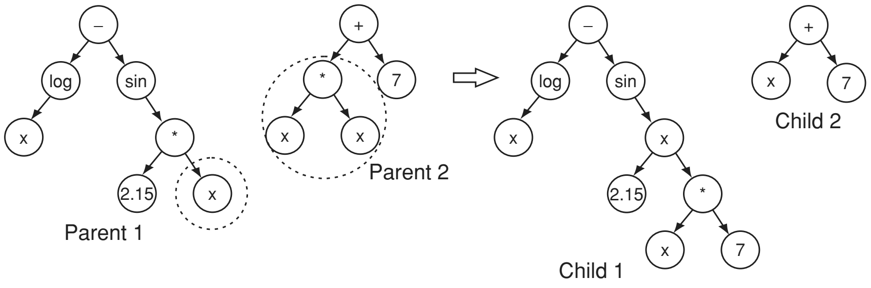

Figure 4.8. Individuals and crossover in GP

LISP language, functions, or variable length structures. To represent a tree in GP, a potential solution of a problem and a set of *functions* and *terminals* corresponding to a given problem domain must be provided by a user. For the individuals represented in Figure 4.8, the set of functions is $F = \{ +, *, log, sin\}$ and the set of terminals is $T = \{2.15, 7, x\}$. Two individuals, *parent1* and *parent2*, represent the expressions $log\ (x) - sin(2.15 * x)$ and $x^2 + 7$, respectively. After calculation of fitness of each individual, selection and genetic operators are applied. Figure 4.9 shows the order of application of the operators. Members of a new generation are created either by a selection operator with a probability p_s or by a crossover operator with a probability p_c or by a mutation operator with a probability p_m ($p_s + p_c + p_m = 1$). A crossover operator creates offspring by exchanging subtrees in parents, as shown in Figure 4.8.

Advanced GP issues concern developing *automatically defined functions* and specialized operators such as *permutation*, *editing*, or *encapsulation* [71,72]. One

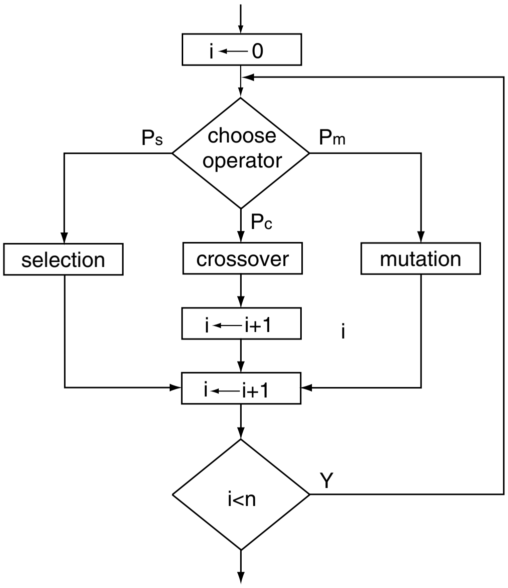

Figure 4.9. Order of selection and genetic operators used in GP

of the research issues concerns developing methodologies to reduce the search space and increase search efficiency. *Context-Free Grammar-based GP*, or *Constrained GP* [66], belongs to some proposed methodologies for automatic processing of additional constraints. Recently [67], a methodology using automatically adapting GP representation has been proposed. GP techniques have been recently used to solve problems of classification and pattern recognition, data mining, forecasting, programming parallel computers and cellular automata, synthesis of analog circuits, and many others.

5 EVOLUTIONARY PROGRAMMING

Evolutionary Programming (EP) is another evolutionary technique developed by Fogel and co-workers [38]. It uses finite state machines (FSMs) as a representation of solutions (see Step 0 in Figure 4.2) in a population of individuals. Surprisingly, it does not use a crossover operator but only mutation and stochastic selection, as shown in Figure 4.10. In its standard version, μ parents create by Gaussian mutation μ offspring, and tournament selection is usually applied. The basic cycle of EP is similar to $(\mu + \mu) - ES$. EP has been used as an approach to artificial intelligence [40] and to combinatorial optimization problems [39]. Recently [140], in the context of multimodal function optimization, *fast EP* has been proposed by introducing a new mutation operator based on Cauchy random numbers. Currently [41], a *meta EP* type of EP is used with multiple mutation operators and is built in to individual parameters to allow self-adaptation.

6 LEARNING CLASSIFIER SYSTEMS

Learning Classifier Systems (LCSs) are a class of rule-based learning machines in which rules are generated and modified by GA [14,46]. Two approaches to LCSs are known: the Pittsburgh approach (see, e.g., [3]) and, much more popular, the Michigan approach. An LCS maintains a population of production rules called *classifiers*. Each rule consists of two parts: a condition part and an action part. The condition part is built using the ternary alphabet {0, 1, #}, where the # symbol matches both 0 and 1. If the condition part of a classifier matches the input sent from the environment (defined by an application), the action part is executed. If more rules match an input from the environment a

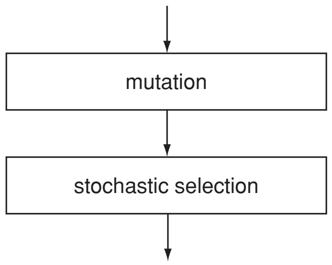

Figure 4.10. Order of selection and genetic operators used in EP

conflict resolution algorithm should be performed. Classifiers interacting with the environment receive *rewards* and their fitness is updated, usually by use of the *bucket brigade* algorithm in classical LCSs. Periodically, GA is applied to produce new rules, but only a small amount of the population is changed during one generation.

Classical LCSs [14,46] appeared from the simplification of Holland's initial work [59] and have been successfully applied in many areas, in particular for data mining (e.g., [60]) and complex control problems (e.g., [122]). In their implementation, a direct reward allocation scheme was used, which was problematic when applied to complex delayed reward tasks [8]. First strength-based ZCSs [135] and sometime later an extended classifier system (XCS) [136] were proposed as a solution to problems encountered in classical strength-based LCSs, and most current research and development is focused on this class of LCSs (e.g., [17,30,137]).

Figure 4.11 shows a simplified version of XCS. It consists of a number of classifiers sets: the *population set* [*P*] of all classifiers (initially empty); the *match set* [*M*]—the set of classifiers whose conditions match the current environmental input; and the *action set* [*A*]—the set of classifiers whose actions will be send to the environment. For a classifier, in addition to the condition and action parts and fitness, some other parameters are specified, such as *prediction p*, *error* ε, and fitness *F*. All these parameters are modified by the system predictions with the use of learning techniques. The action of a classifier is chosen based on the predicted payoffs of the matching rules. GA is applied not to the whole population of rules

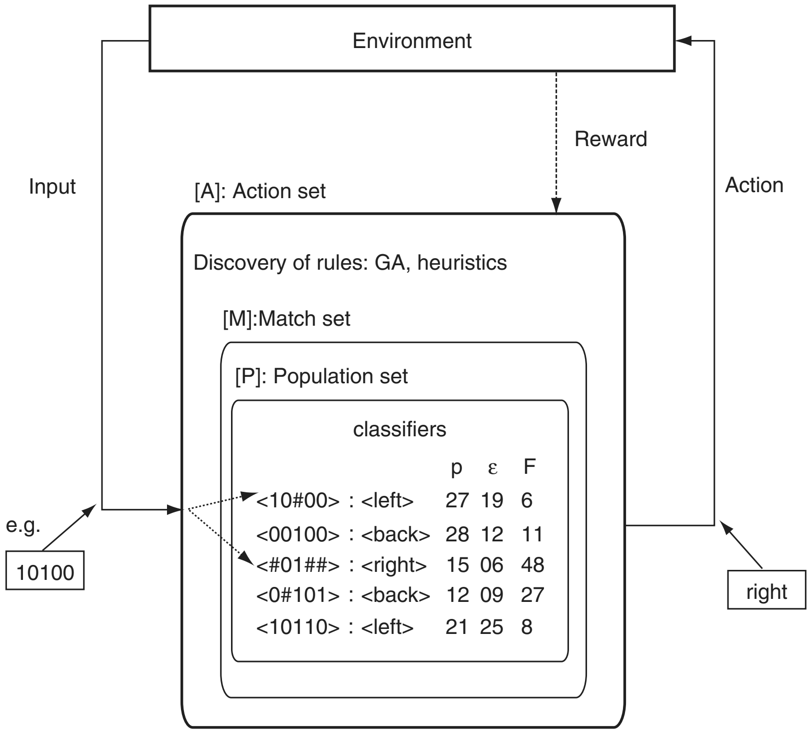

Figure 4.11. Concept of XCS learning classifier system

but only to rules from the *action set*. The wheel roulette or tournament selection and genetic operators of crossover and mutation are applied. A pair of offspring is added to the population [*P*] and replaces two other classifiers from this population.

Recently (e.g., [16,45,123]), Anticipatory Learning Classifier Systems (ALCSs) have emerged in which model-based reinforcement learning is used and, instead of GAs, heuristics are used for improvement of rules.

7 CONCLUSIONS

In this chapter, an overview of the field of EC has been presented. The main emphasis has been on genetic algorithms, the most popular class of EC. However, other important classes of EC were also presented, such as evolution strategies, genetic programming, evolutionary programming, and learning classifier systems. The purpose of the overview has been to present the current state of the field of EC and to discuss the most promising directions of developments in the field.

The EC is a relatively young research area in which the main stream of research is oriented toward experimentation. Despite this emerging state of the field, EC has already proved its potential in solving many theoretical and practical problems. Techniques of EC have been successfully applied to solve, in particular, such commercial problems as [29] cellular telephone tower placement, optical fiber network design, a securities trading system, or process scheduling.

While the theory of EC is still under development, some advances in building such a theory can be noted [102,128]. The *no free lunch theorem* [138] shows that the performance of all search metaheuristics and algorithms averaged over all possible functions is the same if they satisfy certain conditions. The cumulative effects of selection, crossover and mutation operators on evolutionary processes can be studied by designing *Markov chain* models. Properties of GAs can be rigorously proven by the *exact dynamical system model*, covering in particular the original *schema theorem*, and GA dynamics can be approximated by the *statistical mechanics* approach. The concept of *landscape* and some methodologies (e.g., Walsh representation) can be used to predict the performance of GA for solving some problems.

The well-established field of EC serves also as a platform for development of new population-based search algorithms. *Differential evolution* (e.g., [99]), *memetic algorithms* (e.g., [86]), *cultural algorithms* [105], or *probabilistic incremental program evolution* [108] are examples of such search algorithms that are tightly coupled with EC. *Artificial immune systems* (e.g., [31, 64]) and *particle swarm optimization* (e.g., [12,68]) represent search algorithms that are based on new paradigms, but their intersection with evolutionary concepts is visible.

REFERENCES

[1] D. Ackley, M. Litman (1994): A case for Lamarckian evolution. In: *Langton C (ed) Artificial Life III*, Reading, MA, Addison Wesley.

[2] R. K. Ahuja, J. B. Orlin (1997): Developing fitter GAs. *Inform J. Computing*, *9*: 251–253.

[3] J. Bacardit, J. M. Garrel (2003): Evolving multiple discretizations with adaptive intervals for a Pittsburgh rule-based learning classifier system. In: [21]: 1818–1831.

[4] T. B*ä*ck, D. B. Fogel, Z. Michalewicz (eds) (1997): *Handbook of Evolutionary Computation*, IOP Publishing Ltd. and Oxford University Press.

[5] S. Bandyopadhyay, H. Kargupta, G. Wang (1998): Revisiting the GEMGA: scalable evolutionary optimization through linkage learning. *Proc. of the Fourth Int. Conf. on Evolutionary Computation*: pp. 603–608.

[6] W. Banzhaf, et al. (eds) *Proc. of the Genetic and Evolutionary Computation Conference GECCO'99*, Morgan Kaufmann Publishers.

[7] T. B*ä*ck, M. Schütz (1996): Intelligent mutation rate control in canonical genetic algorithms. In: Ras Z W, Michalewicz M (eds) *Foundations of Intelligent Systems*, Springer, *LNAI 1079*: 158-167.

[8] A. Barry (2003): Limits in long path learning with XCS. In: [21]: 1832–1843.

[9] P. J. Bentley, D. W. Corne (eds.)(2002): *Creative Evolutionary Systems*, Morgan Kaufmann.

[10] H. Beyer -G (2001): The theory of evolution strategies, *Natural Computing Series*, Springer, Heidelberg.

[11] H. Beyer -G (2003): Introduction to evolution strategies. In: [44]: 384–426.

[12] T. M. Blackwell (2003): Swarms in dynamic environments. In [20]: 1–12.

[13] T. Blickle, L. Thiele (1996): A comparison of selection schemes used in evolutionary algorithms. *Evolutionary Computation*, *4*: 361–394.

[14] L. B. Booker, D. E. Goldberg, J. H. Holland (1989): Classifier systems and genetic algorithms. *Artificial Intelligence 40*: 235–282.

[15] J. Branke (2002): *Evolutionary Optimization in Dynamic Environments*, Kluwer Academic Publishers.

[16] M. V. Butz (2002): Biasing exploration in an anticipatory learning classifier system. In: Lanzi et al. (eds) *Advances in Learning Classifier Systems*, LNAI 2321, Springer: 3–22.

[17] M. V. Butz, K. Sastry, D. E. Goldberg (2003): Tournament selection: stable fitness pressure in XCS. In: [21]: 1857–1869.

[18] E. Cantu-Paz (2003): Parallel genetic algorithms. In: [44]: 241–257.

[19] E. Cantu-Paz (1999): Topologies, migration rates, and multi-population parallel genetic algorithms. In: [6]: 91–98.

[20] E. Cantu-Paz et al. (eds) (2003): Genetic and Evolutionary Computation-GECCO 2003, Part I, LNCS 2723, Springer.

[21] E. Cantu-Paz et al. (eds) (2003): Genetic and Evolutionary Computation-GECCO 2003, Part II, LNCS 2724, Springer.

[22] Z. S. H. Chan, H. W. Ngan, A. B. Rad (1999): Minimum-allele-reserve-keeper (MARK): a fast and effective mutation scheme for genetic algorithm. In: [6], *1*: 106–113.

[23] H. Choe, S-S. Choi, B-R. Moon (2003): A hybrid genetic algorithm for hexagonal tortoise problem. In: [20]: 850–861.

[24] C. A. Coello Coello (1999): A comprehensive survey of evolutionary-based multiobjective optimization techniques. *Knowledge and Information Systems 1*(3):269–308.

[25] C. A. Coello Coello, D. A. Van Veldhuizen, G. B. Lamont (2002): Evolutionary Algorithms for Solving Multi-objective Problems. Kluwer Academic.

[26] D. Corn, M. Dorigo, F. Glover (eds) (1999): New Ideas in Optimization. McGraw-Hill, London, 1999.

[27] Y. Davidor, H-P. Schwefel, R. Manner (eds) (1994): Parallel Problem Solving from Nature—PPSN III, LNCS 866, Springer.

[28] L. Davis (1991): Bit-climbing, representational bias, and test suite design. In: L. Booker, R. Belew (eds) *Proc. of the 4th Int. Conf. on GAs*, Morgan Kaufmann: 18–23.

[29] L. D. Davis (1999): Commercial applications of evolutionary computation: some case studies. In: [43]: 38–51.

[30] D. Dawson (2003): Improving performance in size-constrained extended classifier systems. In: [21]: 1870–1881.

[31] L. N. De Castro, J. Timmis (2002): Artificial Immune Systems: A New Computational Intelligence Approach, Springer.

[32] K. Deb, D. E. Goldberg (1989): An investigation on niche and species formation in genetic function optimization. In: Schaffer J D et al. (eds) *Proc. of the Third Int. Conf. on Genetic Algorithms*. Morgan Kaufmann Publishers: pp. 42–50.

[33] I. De Falco, A. Iazzetta, E. Tarantino (1999): Towards a simulation of natural mutation. In: [6], *1*: 156–163.

[34] K. De Jong (1975): An Analysis of the Behavior of a Class of Genetic Adaptive Systems. Doctoral dissertation, University of Michigan, Ann Arbor, Michigan.

[35] K. De Jong (2003): Evolutionary computation: a unified approach. In: [44]: 644–652.

[36] L. J. Eshelman (1991): The CHC adaptive search algorithm: how to have safe search when engaging in nontraditional genetic recombination. In: G. J. E. Rawlins (ed) Foundations of Genetic Algorithms, Morgan Kaufmann, San Mateo, CA: 265–283.

[37] F. P. Espinoza, B. S. Minsker, D. E. Goldberg (2003): Performance evaluation and population reduction for a self adaptive hybrid genetic algorithm (SAHGA). In: [20]: 922–933.

[38] L. J. Fogel, A. J. Owens, M. J. Walsh (1966): Artificial Intelligence Through Simulated Evolution. John Wiley, Chichister, UK.

[39] D. B. Fogel (1993): Applying evolutionary programming to selected traveling salesman problems. *Cybern. Syst.*, *24*: 27–36.

[40] D. B. Fogel (1995): Evolutionary Computation. Towards a New Philosophy of Machine Intelligence, IEEE Press.

[41] G. B. Fogel, K. Chellapilla (1999): Simulated sequencing by hybridization using evolutionary programming. In: *Proc. of the 1999 Congress on Evolutionary Computation*, *1*: 463–469.

[42] A. S. Fukunaga (1998): Restart scheduling for genetic algorithms. In: A. E. Eiben et al.(eds) Parallel Problem Solving from Nature—PPSN V, Springer, LNCS *1498*: 357–366.

[43] GECCO-1999: 1999 Genetic and Evolutionary Computation Conference. Tutorial Program. Orlando, Florida, July 14, 1999.

[44] GECCO-2003: 2003 Genetic and Evolutionary Computation Conference. Tutorial Program. Chicago, Illinois, July 13, 2003.

[45] P. Gerard, O. Sigaud (2003): Designing efficient exploration with MACS: modules and function approximation. In: [21]: 1882–1893.

[46] D. E. Goldberg (1989): Genetic Algorithms in Search, Optimization, and Machine Learning, Addison-Wesley, Reading, Massachusets.

[47] D. E. Goldberg (2002): The Design of Innovation. Lessons from and for Competent Genetic Algorithms. Kluwer Academic Publishers, Boston/Dordrecht/London.

[48] D. E. Goldberg, K. Deb, J. H. Clark (1992): Genetic algorithms, noise and the sizing of population. *Complex Systems*, 6: 333–362.

[49] D. E. Goldberg, K. Deb, H. Kargupta, G. Harik (1993): Rapid, accurate optimization of difficult problems using fast messy genetic algorithms. *Proc. of the Fifth Int. Conf. on Genetic Algorithms*: 56–64.

[50] M. Gorges-Schleuter (1992): Comparison of local mating strategies in massively parallel genetic algorithms. In: [78]: 553–562.

[51] J. Grefenstette (1997): Efficient implementation of algorithms. In: [4]: E2.1:1–E2.1:6.

[52] G. R. Harik (1997): Learning gene linkage to efficiently solve problems of bounded difficulty using genetic algorithms. Unpublished doctoral dissertation, University of Michigan, Ann Arbor, also IlliGAL Report No. 97005.

[53] G. R. Harik (1999): Linkage Learning via Probabilistic Modeling in the ECGA. IlliGAL Report No. 99010, Urbana, IL, University of Illinois at Urbana-Champaign.

[54] W. Hart , R. Belew (1995): Optimization with genetic algorithm hybrids that use local search. In: R. Below and M. Mitchell (eds.) Adaptive Individuals in Evolving Populations: Models and Algorithms, Reading, MA, Addison Wesley.

[55] R. Heckendorn (2003): An introduction to genetic algorithms: theory and practice. In: [44]: 225–240.

[56] W. D. Hillis (1992): Co-evolving parasites improve simulated evolution as an optimization procedure. In: C. G. Langton et al. (eds) Artificial Life II. Addison-Wesley.

[57] G. E. Hinton, S. J. Nowlan (1987): How learning can guide evolution. *Complex Systems*, *1*: 495–502.

[58] T. P. Hoehn, C. C. Pettey (1999): Parental and cyclic-rate mutation in genetic algorithms: an initial investigation. In: [6], *1*: 297–304.

[59] J. H. Holland (1985): Properties of the bucket brigade algorithm. In: J. J. Grefenstette (ed) *Proc. of the 1st Int. Conf. on Genetic Algorithms and Their Applications*: 1–7.

[60] J. H. Holmes (1996): A genetics-based machine learning approach to knowledge discovery in clinical data. J. American Medical Informatics Association Supplement.

[61] F. Hoffmeister, T. Bäck (1992): Genetic Algorithms and Evolution Strategies: Similarities and Differences. Technical Report No SYS-1/92, University of Dortmund.

[62] G. Huang , A. Lim (2003): Designing a hybrid genetic algorithm for the linear ordering problem. In: [20]: 1053–1064.

[63] P. Husbands (1994): Distributed coevolutionary genetic algorithms for multi-criteria and multi-constraint optimization. In: T. C. Fogarty (ed) Evolutionary Computing, LNCS 865, Springer: 150–165.

[64] IEEE Trans. on Evolutionary Computation (2002). Special issue on artificial immune systems, *6*, 3(1).

[65] A. Iorio, X. Li (2002): Parameter control within a co-operative co-evolutionary genetic algorithm. In: M. Guervos et al. (eds) *Proc. of the Seventh Conf. on Parallel Problem Solving from Nature (PPSN VII)*, Springer: pp. 247–256.

[66] C. Z. Janikow (1996): A methodology for processing problem constraints in genetic programming. *Computers and Mathematics with Applications*, vol. 32, No 8: 97–113.

[67] C. Z. Janikow, R. A. Deshpande (2003): Adaptation of representation in GP. In: C. H. Dagli et al. (eds) *Smart Engineering System Design*, *13*: 45–50.

[68] J. Kennedy, R. C. Eberhart (1999): The particle swarm: social adaptation in information-processing systems. In: [26]: 379–387.

[69] D. Knjazew (2002): OmeGA. A Competent Genetic Algorithm for Solving Permutation and Scheduling Problems. Kluwer Academic Publishers, Boston/Dordrecht/London.

[70] J. R. Koza (1992): Genetic programming: on the programming of computers by natural selection. MIT Press, Cambridge, MA.

[71] J. R .Koza (2003): Introduction to genetic programming. In: [44]: 1–34.

[72] W. B. Langdon , R. Poli (2003): Foundations of genetic programming. In: [44]: 53–105.

[73] S. -C. Lin, E. D. Goodman, W. F. Punch, III (1997): Investigating parallel genetic algorithms on job shop scheduling problems. In: Evolutionary Programming VI, LNCS 1213, Springer: 383–393.

[74] J. Lis, A. E. Eiben (1996): A multi-sexual genetic algorithm for multiobjective optimization. In: T. Fukuda, T. Furuhashi (eds) *Proc. of the 1996 Int. Conf. on Evolutionary Computation*. IEEE: 59–64.

[75] A. J. Lotka (1925), Elements of Physical Biology, Williams and Wilkins, Baltimore.

[76] S. W. Mahfoud (1992): Crowding and preselection revisited. In: [78]: 27–36.

[77] W. N. Martin, J. Lienig, J. P. Cohoon (1997): Island (migration) models: evolutionary algorithms based on punctuated equlibria. In: [4]: C6.3:1–C6.3:16.

[78] R. M*ä*nner, B. Manderick (eds) (1992): Parallel Problem Solving from Nature, 2. North-Holland.

[79] M. McIlhagga , P. Husbands, R. Ives (1996): A comparison of optimization techniques for integrating manufacturing, planning and scheduling. In: [126]: 604–613.

[80] O. J. Mengshoel, D. E. Goldberg (1999): Probabilistic crowding: deterministic crowding with probabilistic replacement. In: [6]: 409–416.

[81] Z. Michalewicz (1996): Genetic Algorithms + Data Structures = Evolution Programs, Springer-Verlag, Berlin.

[82] Z. Michalewicz (1995): Evolutionary computation: an overview. In: J. Komorowski (eds) *Proc. of the 8th Scandinavian Conf. on Artificial Intelligence*. IOS Press, *28*: 322–337.

[83] M. Mitchell, J. H. Holland, S. Forrest (1994): When will a genetic algorithm outperform hill climbing. In: J. D. Cowan et al. (eds) Advances in Neural Information Processing Systems, vol. 6, Morgan Kaufmann: 51–58.
[84] M. Mitchel (1996): An Introduction to Genetic Algorithms. The MIT Press, Cambridge Massachusetts.
[85] T. M. Mitchell (1997): Machine Learning. McGraw-Hill.
[86] P. Moscato (1999): Memetic algorithms: a short introduction. In: [26]: 219–244.
[87] H. M*ü*hlenbein (1992): How genetic algorithms really work I. Mutation and hillclimbing. In: [78]: 15–25.
[88] H. M*ü*hlenbein, D. Schlierkamp-Voosen (1994): The science of breeding and its application to the breeder genetic algorithm. *Evolutionary Computation*, *1*: 335–360.
[89] Y. Nagata, S. Kobayashi (1997): Edge assembly crossover: a high-power genetic algorithm for the traveling salesman problem. In: T. B*ä*ck (ed) Proc. of 7th Int. Conf. on Genetic Algorithms, Morgan Kaufmann, San Francisco, CA: 450–457.
[90] V. Nissen, J. Biethahn (1995): An introduction to evolutionary algorithms. In: J. Biethahn and V. Nissen (eds) Evolutionary Algorithms in Management Applications, Springer: 3–97.
[91] G. Ochoa, I. Harvey, H. Buxton (1999): On recombination and optimal mutation rates. In: [6], *1*: 488–496.
[92] C. C. Palmer (1994): An Approach to a Problem in Network Design using Genetic Algorithms. Unpublished Ph.D. thesis, Polytechnic University, Troy, NY.
[93] J. Paredis (1994): Co-evolutionary constraint satisfaction. In: [27]: 46–55.
[94] J. Paredis (1996): Coevolutionary life-time learning. In: [126]: 72–80.
[95] M. Pelikan, D. E. Goldberg, E. Cantu-Paz (1999): BOA: The Bayesian optimization algorithm. In: [6]: 525–532.
[96] A. S. Perelson, R. Hightower, S. Forrest (1996): Evolution and somatic learning in V-Region genes. *Research in Immunology*, *147*: 202–208.
[97] C. C. Pettey (1997): Diffusion (cellular) models. In: [4]: C6.4:1–C6.4:6.
[98] M. A. Potter, K. A. De Yong (1994): A cooperative coevolutionary approach to function optimization. In: [27]: 249–257.
[99] K. V. Price (1999) An introduction to differential evolution. In: [26]: 79–108.
[100] C. R. Reeves (ed) (1993): Modern Heuristics Techniques for Combinatorial Problems. Blackwell Scientific, Oxford, UK.
[101] N. Radcliffe (1992), Non-linear genetic representations. In: [78]: 259–268.
[102] C. R. Reeves, J. E. Rowe (2003): Genetic Algorithms: Principle and Perspectives: A Guide to GA Theory. Kluwer Academic Publishers.
[103] S. Ronald (1997): Robust encoding in genetic algorithms: a survey of encoding issues. In: *Proc. of the Forth Int. Conf. on Evolutionary Computation*, Piscataway, NJ, IEEE: 43–48.
[104] I. Rechenberg (1994): Evolutionsstrategie. Frommann-Holzboog Verlag, Stuttgart.
[105] R. G. Reynolds (1999): Cultural algorithms: theory and applications. In: [26]: 367–377.
[106] F. Rothlauf (2003): Population sizing for the redundant trivial voting mapping. In: [21]: 1307–1319.

[107] F. Rothlauf (2003): Representations for genetic and evolutionary algorithms. In: [44]: 203–224.

[108] R. Salustowicz, J. Schmidhuber (1999): From probabilities to programs with probabilistic incremental program evolution. In: [26]: 433–450.

[109] J. Sarma, K. A. De Jong (1996): An analysis of the effects of neighborhood size and shape on local selection algorithms. In: [126]: 236–244.

[110] R. Schaefer, J. Kolodziej (2003): Genetic search reinforced by the population hierarchy. In: K. A. De Jong, R. Poli, J. E. Rove (eds) Foundations of Genetic Algorithms 7, Morgan Kaufmann: 383–399.

[111] J. D. Schaffer (ed)(1989): *Proc. of 3rd Int. Conf. on Genetic Algorithms*, Morgan-Kaufmann, San Mateo, CA.

[112] J. D. Schaffer, R. A. Caruana, L. J. Eshelman, R. Das (1989): A study of control parameters affecting online performance of genetic algorithms for function optimization. In: [111]: 51–60.

[113] H -P. Schwefel (1995): Evolution and Optimum Seeking, Wiley, New York.

[114] H -P. Schwefel, C. Rudolph (1995): Contemporary evolution strategies. In: Third Int. Conf. on Artificial Life, LNCS 929, Springer Verlag: 893–907.

[115] R. E. Smith, C. Bonacina (2003): Mating restriction and niching pressure: results from agents and implications for general EC. In: [21]: 1382–1393.

[116] D. Surry, N. Radcliffe (1996): Formal Algorithms + Formal Representations = Search Strategies. In: [126].

[117] F. Seredynski (1994): Loosely coupled distributed genetic algorithms. In: [27]: 514–523.

[118] F. Seredynski (1997): Competitive coevolutionary multi-agent systems: the application to mapping and scheduling problems. *Journal of Parallel and Distributed Computing*, *47*: 39–57.

[119] F. Seredynski (1998): New trends in parallel and distributed evolutionary computing. Fundamenta Informaticae 35, IOS Press: 211–230.

[120] F. Seredynski, A. Y. Zomaya, P. Bouvry (2003): Function Optimization with Coevolutionary Algorithms. In: M. A. Klopotek et al. (eds) Intelligent Information Processing and Web Mining, Advances in Soft Computing, Springer: 13–22.

[121] R. E. Smith, B. A. Dike, R. K. Mehra, B. Ravichandran, A. El-Fallah (1999): Classifier systems in combat: two-sided learning of maneuvers for advanced fighter aircraft. In: Computer Methods in Applied Mechanics and Engineering, Elsevier.

[122] J. E. Smiths, F. Vavak (1999): Replacement strategies in steady state genetic algorithms: dynamic environments. *Journal of Computing and Information Technology*, *7*(1): 49–59.

[123] W. Stolzmann (2003): Anticipatory classifier systems. In: [44]: 493–517.

[124] R. Tsang, P. Lajbcygier (2002): Optimizing technical trading strategies with split search genetic algorithms. In: S.-H. Chen (ed) Evolutionary Computation in Economic and Finance. Physica-Verlag, Heildeiberg, New York: 333–358.

[125] F. Vavak, T. C. Fogarty, K. Jukes (1996): A genetic algorithm with variable range of local search for tracking changing environments. In: [126].

[126] H -M. Voight et al. (eds) (1996): Parallel Problem Solving from Nature-PPSN IV, Springer, LNCS 1411.

[127] V. Volterra (1926): Variazoni e Fluttuazioni Del Numero D'individui in Specie Animali Conviventi. *Memorie della R. Accaddemia Nazionale dei Lincei*, *2*: 31–113.

[128] M. D. Vose (1999): The Simple Genetic Algorithm. MIT Press.

[129] I. Wegener, W. Carsten (2003): On the optimization of monotone polynomials by the (1 + 1) EA and randomized local search. In: [20]: 622–633.

[130] D. Whitley, D. Garrett, J -P. Watson (2003): Quad search and hybrid genetic algorithms. In: [21]: 1469–1480.

[131] D. Whitley, V. S. Gordon, K. Mathias (1994): Lamarckian evolution, the Baldwin effect and function optimization. In: [27]: 6–15.

[132] D. Whitley (1999): A free lunch proof for Grey versus binary encoding. In: [6]: 726–733.

[133] D. Whitley (2003): Evaluating search algorithms. In: [44]: 132–147.

[134] D. Whitley (1989): The GENITOR algorithm and selection pressure: why rank-based allocation of reproductive trials is best. In: [111]: 116–121.

[135] S. W. Wilson (1994): ZCS: A zeroth level classifier system. *Evolutionary Computation 2*(1): 1–18.

[136] S. W. Wilson (1995): Classifier fitness based on accuracy. *Evolutionary Computation 3*: 149–175.

[137] S. W. Wilson (2003): Structure and Function of the XCS classifier system. In: [44]: 547–555.

[138] D. H. Wolpert, W. G. Macready (1997): No free lunch theorems for optimization. *IEEE Trans. on Evolutionary Computation*, 1: 67–82.

[139] X. Yao (1996): An overview of evolutionary computation. *Chinese Journal of Advanced Software Research*, *3*, 1:(1) 12–29.

[140] X. Yao (1999): Evolutionary programming made faster. *IEEE Trans. on Evolutionary Computation*, *3*, 2(1): 82–102.