Evolutionary Algorithms

Enrique Alba and Carlos Cotta

Dept. Lenguajes y Ciencias de la Computación, ETSI Informática
Universidad de Málaga, Campus de Teatinos, 29016 - Málaga - Spain

{eat, ccottap}@lcc.uma.es

February 19, 2004

Abstract

Evolutionary algorithms (EAs) are stochastic optimization techniques based on the principles of natural evolution. An overview of these techniques is provided here. We describe the general functioning of EAs, and give an outline of the main families into which they be divided. Subsequently, we analyze the different components of an EA, and provide some examples on how these can be instantiated. We finish with a glimpse of the numerous applications of these tecniques.

1 Introduction

One of the most striking features of Nature is the existence of living organisms adapted for surviving in almost any ecosystem, even the most inhospitable: from abyssal depths to mountain heights, from volcanic vents to polar regions. The magnificence of this fact becomes more evident when we consider that the life environment is continuously changing. This motivates that certain life forms become extinct whereas other beings evolve and preponderate due to their adaptation to the new scenario. It is very remarkable that living beings do not exert a conscious effort for evolving (actually, it would be rather awkward to talk about consciousness in amoebas or earthworms); much on the contrary, the driving force for change is controlled by supra-organic mechanisms such as natural evolution.

Can we learn –and use for our own profit– the lessons that Nature is teaching us? The answer is a big YES, as the optimization community has repeatedly shown in the last decades. *'Evolutionary algorithm'* are the key words here. The term evolutionary algorithm (EA henceforth) is used to designate a collection of optimization techniques whose functioning is loosely based on metaphors of biological processes.

This rough definition is rather broad and tries to encompass the numerous approaches currently existing in the field of evolutionary computation [10].

Quite appropriately, this field itself is continuously evolving; a quick inspection at the proceedings of the relevant conferences and symposia suffices to demonstrate the impetus of the field, and the great diversity of the techniques that can be considered 'evolutionary'.

This variety notwithstanding, it is possible to find a number of common features of all (or at least most of) EAs. The following quote from [70] illustrates such common points:

> "The algorithm maintains a collection of potential solutions to a problem. Some of these possible solutions are used to create new potential solutions through the use of *operators*. Operators act on and produce collections of potential solutions. The potential solutions that an operator acts on are selected on the basis of their quality as solutions to the problem at hand. The algorithm uses this process repeatedly to generate new collections of potential solutions until some stopping criterion is met."

This definition can be usually found in the literature expressed in a technical language that uses terms such as *genes*, *chromosomes*, *population*, etc. This jargon is a reminiscence of the biological inspiration mentioned before, and has deeply permeated the field. We will return to the connection with biology later on.

The objective of this work is to present a gentle overview of these techniques, comprising both the classical 'canonical' models of EAs, as well as some modern directions for the development of the field, namely the use of parallel computing, and the introduction of problem-dependent knowledge.

2 Learning from Biology

Evolution is a complex fascinating process. Along History, scientist have attempted to explain its functioning using different theories. After the development of disciplines such as comparative anatomy in the middle of the 19th century, the basic principles that condition our current vision of Evolution were postulated. Such principles rest upon Darwin's Natural Selection Theory [41], and Mendel's work on genetic inheritance [81]. They can be summarized in the following points (see [85]):

- Evolution is a process that does not operate on organisms directly, but on *chromosomes*. These are the organic tools by means of which the structure of a certain living being is encoded, i.e. the features of a living being are defined by the decoding of a collection of chromosomes. These chromosomes (more precisely, the information they contain) pass from one generation to another through reproduction.

- The evolutionary process takes place precisely during reproduction. Nature exhibits a plethora of reproductive strategies. The most essential ones are *mutation* (that introduces variability in the gene pool) and *recombination* (that introduces the exchange of genetic information among individuals).

- Natural selection is the mechanism that relates chromosomes with the adequacy of the entities they represent, favoring the proliferation of effective, environment-adapted organisms, and conversely causing the extinction of lesser effective, non-adapted organisms.

These principles are comprised within the most orthodox theory of evolution, the Synthetic Theory [68]. Although alternate scenarios that introduce some variety in this description have been proposed –e.g. the Neutral Theory [73], and very remarkably the Theory of Punctuated Equilibria [60]– it is worth to consider initially the former basic model. It is amazing to see that despite the apparent simplicity of the principles upon which it rests, Nature exhibits unparallel power in developing and expanding new life forms.

Not surprisingly, this power has attracted the interest of many researchers, who have tried to translate the principles of evolution to the realm of algorithmics, pursuing the construction of computer systems with analogous features. An important point must be stressed here: evolution is an undirected process, i.e. there exists no scientific evidence that evolution is headed to a certain final goal. On the contrary, it can be regarded as a reactive process that makes organisms change in response to environmental variations. However, it is a fact that human-designed systems do pursue a definite final goal. Furthermore, whichever this goal might be, it is in principle desirable to reach it quickly and efficiently. This leads to distinguish two approaches to the construction of nature-inspired systems:

a) Trying to reproduce Nature principles with the highest possible accuracy, i.e., simulate Nature.

b) Using these principles as inspiration, adapting them in whatever required way so as to obtain efficient systems for performing the desired task.

Both approaches concentrate nowadays the efforts of researchers. The first one has given rise to the field of Artificial Life (e.g. see [77]), and it is interesting because it allows re-creating and studying numerous natural phenomena such as parasitism, predator/prey relationships, etc). The second approach can be considered more practical, and constitutes the source of EAs. Notice anyway that these two approaches are not hermetic containers, and have frequently interacted with certainly successful results.

3 Nature's way for Optimizing

As mentioned above, the standpoint of EAs is essentially practical: using ideas from natural evolution in order to solve a certain problem. Let us focus on optimization and see how this goal can be achieved.

3.1 Algorithm Meets Evolution

An EA is a stochastic iterative procedure for generating tentative solutions for a certain problem $\mathcal{P}$. The algorithm manipulates a collection P of individuals (the *population*), each of which comprises one or more chromosomes. These chromosomes allow each individual represent a potential solution for the problem under consideration. An encoding/decoding process is responsible for performing this mapping between chromosomes and solutions. Chromosomes are divided into smaller units termed *genes*. The different values a certain gene can take are called the *alleles* for that gene.

Initially, the population is generated at random or by means of some heuristic seeding procedure. Each individual in P receives a *fitness* value: a measure of how good the solution it represents is for the problem being considered. Subsequently, this value is used within the algorithm for guiding the search. The whole process is sketched in Figure 1.

As it can be seen, it is assumed the existence of a set $\mathcal{F}$ (also known as *phenotype space*) comprising the solutions for the problem at hand. Associated with $\mathcal{F}$, there also exists a set $\mathcal{G}$ (known as *genotype space*). These sets $\mathcal{G}$ and $\mathcal{F}$ respectively constitute the domain and codomain of a function g known as the *growth* (or *expression*) function. It could be the case that $\mathcal{F}$ and $\mathcal{G}$ were actually equivalent, being g a trivial identity function. However, this is not the general situation. As a matter of fact, the only requirement posed on g is surjectivity. Furthermore, g could be undefined for some elements in $\mathcal{G}$.

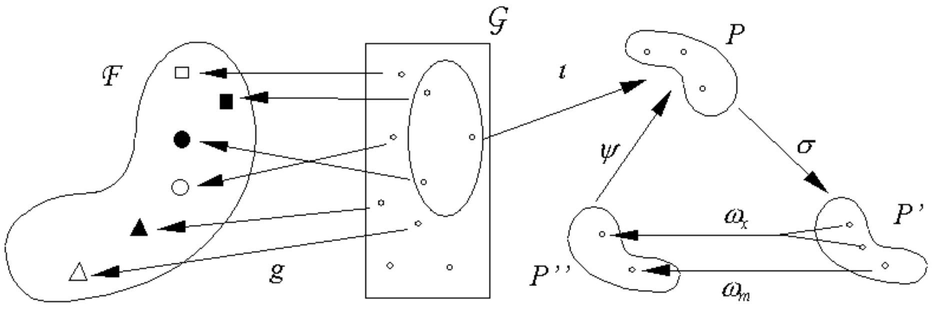

Figure 1: Illustration of the evolutionary approach to optimization.

After having defined these two sets $\mathcal{G}$ and $\mathcal{F}$, notice the existence of a function ι selecting some elements from $\mathcal{G}$. This function is called the *initialization* function, and these selected solutions (also known as *individuals*) constitute the so-called *initial population*. This initial population is in fact a pool of solutions onto which the EA will subsequently work, iteratively applying some evolutionary operators to modify its contents. More precisely, the process comprises three major stages: *selection* (promising solutions are picked form the population by using a selection function σ), *reproduction* (new solutions are created by modifying selected solutions using some reproductive operators ω_i), and *replacement*

Evolutionary-Algorithm:

```
1. $P \leftarrow$ **apply** $\iota$ on $\mathcal{G}$ to obtain $\mu$ individuals (the initial population);
2. **while** Termination Criterion is not met **do**
    (a) $P' \leftarrow$ **apply** $\sigma$ on $P$; /* selection */
    (b) $P'' \leftarrow$ **apply** $\omega_1, \cdots, \omega_k$ on $P'$; /* reproduction */
    (c) $P \leftarrow$ **apply** $\psi$ on $P$ and $P''$; /* replacement */
   **endwhile**
```

Figure 2: Pseudocode of an evolutionary algorithm.

(the population is updated by replacing some existing solutions by the newly created ones, using a replacement function ψ). This process is repeated until a certain termination criterion (usually reaching a maximum number of iterations) is satisfied. Each iteration of this process is commonly termed a *generation*.

According to this description, it is possible to express the pseudocode of an EA as shown in Figure 2. Every possible instantiation of this algorithmic template[1] will give rise to a different EA. More precisely, it is possible to distinguish different EA families, by considering some guidelines on how to perform this instantiation.

3.2 The Flavors of Evolutionary Algorithms

EAs, as we know them now, began their existence during the late 1960s and early 1970s (some earlier references to the topic exist tough; see [53]). In these years –and almost simultaneouly– scientists from different places in the world began the task of putting Nature at work in algorithmics, and more precisely in search or problem solving duties. The existence of these different primordial sources originated the rise of three different EA models. These classical families are:

- *Evolutionary Programming* (EP): this EA family originated in the work of Fogel *et al.* [55]. EP focuses in the adaption of individuals rather than in the evolution of their genetic information. This implies a much more abstract view of the evolutionary process, in which the behavior of individuals is directly modified (as opposed to manipulating its genes). This behavior is typically modeled by using complex data structures such as finite automata or as a graphs (see Figure 3-left). Traditionally, EP uses asexual reproduction –also known as mutation, i.e. introducing slight changes in an existing solution– and selection techniques based on direct competition among individuals.

[1]The mere fact that this high-level heuristic template can host a low-level heuristic, justifies using the term *metaheuristic*, as it will be seen later.

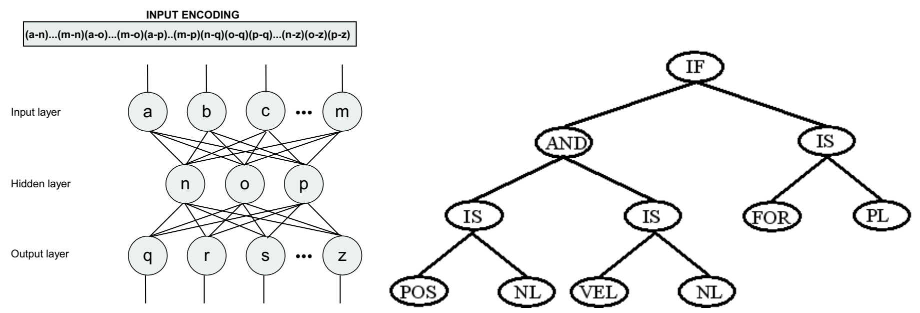

Figure 3: Two examples of complex representations. (Left) A graph representing a neural network. (Right) A tree representing a fuzzy rule.

- *Evolution Strategies* (ES): these techniques were initially developed in Germany by Rechenberg and Schwefel [93, 96]. Their original goal was serving as a tool for solving engineering problems. With this goal in mind, these techniques are characterized by manipulating arrays of floating-point numbers (there exist versions of ES for discrete problems, but they are much more popular for continuous optimization). As EP, mutation is sometimes the unique reproductive operator used in ES; it is not rare to also consider recombination (i.e. the construction of new solutions by combining portions of some indviduals) though. A very important feature of ES is the utilization of self-adaptive mechanisms for controlling the application of mutation. These mechanisms are aimed at optimizing the progress of the search by evolving not only the solutions for the problem being considered, but also some parameters for mutating these solutions (in a typical situation, an ES individual is a pair $(\overrightarrow{x}, \overrightarrow{\sigma})$, where $\overrightarrow{\sigma}$ is a vector of standard deviations used to control the Gaussian mutation exerted on the actual solution $\overrightarrow{x}$).

- *Genetic Algorithms* (GA): GAs are possibly the most widespread variant of EAs. They were conceived by Holland [66]. His work has had a great influence in the development of the field, to the point that some portions –arguably extrapolated– of it were taken almost like dogmas (i.e. the ubiquitous use of binary strings as chromosomes). The main feature of GAs is the use of a recombination (or *crossover*) operator as the primary search tool. The rationale is the assumption that different parts of the optimal solution can be independently discovered, and be later combined to create better solutions. Additionally, mutation is also used, but it was usually considered a secondary background operator whose purpose is merely 'keeping the pot boiling' by introducing new information in the population (this classical interpretation is no longer considered valid though).

These families have not grown in complete isolation from each other. On the contrary, numerous researchers built bridges among them. As a result of this interaction, the borders of these classical families tend to be fuzzy (the reader may check [9] for a unified presentation of EA families), and new variants have emerged. We can cite the following:

- *Evolution Programs*: this term is due to Michalewicz [85], and comprises those techniques that, while using the principles of functioning of GAs, evolve complex data structures, as in EP. Nowadays, it is customary to use the acronym GA –or more generally EA– to refer to such an algorithm, leaving the term 'traditional GA' to denote classical bit-string based GAs.

- *Genetic Programming* (GP): the roots of GP can be traced back to the work of Cramer [40], but it is undisputable that it has been Koza [74] the researcher who promoted GP to its current status. Essentially, GP could be viewed as an evolution program in which the structures evolved represent computer programs. Such programs are typically encoded by trees (see Figure 3-ritgh). The final goal of GP is the automatic design of a program for solving a certain task, formulated as a collection of (input, output) examples.

- *Memetic Algorithms* (MA): these techniques owe their name to Moscato [86]. Some widespread misconception equates MAs to EAs augmented with local search; although such an augmented EA could be indeed considered a MA, other possibilities exist for defining MAs. In general, a MA is problem-aware EA [87]. This problem awareness is typically acquired by combining the EA with existing algorithms such as hill climbing, branch and bound, etc.

In addition to the different EA variants mentioned above, there exist several other techniques that could also fall within the scope of EAs, such as *Ant Colony Optimization* [45], *Distribution Estimation Algorithms* [78], or *Scatter Search* [76] among others. All of them rely on achieving some kind of balance between the exploration of new regions of the search space, and the exploitation of regions known to be promising [18], so as to minimize the computational effort for finding the desired solution. Nevertheless, these techniques exhibit very distinctive features that make them depart from the general pseudocode depicted in Figure 2. The broader term *metaheuristic* (e.g. see [57]) is used to encompass this larger set of modern optimization techniques, including EAs.

4 Dissecting an Evolutionary Algorithm

Once the general structure of an EA has been presented, we will get into more detail on the different components of the algorithm.

4.1 The Fitness Function

This is an essential component of the EA, to the point that some early (and nowadays discredited) views of EAs considered it as the unique point of interaction with the problem that is intended to be solved. This way, the fitness function measured how good a certain tentative solution is for the problem of interest. This interpretation has given rise to several misconceptions, the most important being the equation 'fitness = quality of a solution'. There are many examples in which this is simple not true [87], e.g., tackling the satisfiability problem with EAs (that is, finding the truth assignment that makes a logic formula in conjunctive normal form be satisfied). If quality is used as fitness function, then the search space is divided into solutions with fitness 1 (those satisfying the target formula), and solutions with fitness 0 (those that do not satisfy it). Hence, the EA would be essentially looking for a needle in a haystack (actually, there may be more than one needle in that haystack, but that does not change the situation). A much more reasonable choice is making fitness be the number of satisfied clauses in the formula by a certain solution. This introduces a gradation that allows the EA 'climbing' in search of near-optimal solutions.

The existence of this gradation is thus a central feature of the fitness function, and its actual implementation is not that important as long this goal is achieved. Of course, implementation issues are important from a computational point of view, since the cost of the EA is typically assumed to be that of evaluating solutions. In this sense, it must be taken into account that fitness can be measured by means of a simple mathematical expression, or may involve performing a complex simulation of a physical system. Furthermore, this fitness function may incorporate some level of noise, or even vary dynamically. The remaining components of the EA must be defined accordingly so as to deal with these features of the fitness function, e.g., using a non-haploid representation [97] (i.e., having more than one chromosome) so as to have a genetic reservoir of worthwhile information in the past, and thus be capable of tackling dynamic changes in the fitness function.

Notice that there may even exist more than one criterion for guiding the search (e.g., we would like to evolve the shape of a set of pillars, so that their strength is maximal, but so that their cost is also minimal). These criteria will be typically partially conflicting. In this case, a *multiobjective* problem is being faced. This can be tackled in different ways, such as performing an aggregation of these multiple criteria into a single value, or using the notion of Pareto dominance (i.e., solution x dominates solution y if, and only if, $f_i(x)$ yields a better or equal value than $f_i(y)$ for all i, where the f_i's represent the multiple criteria being optimized). See [21, 20] for details.

4.2 Initialization

In order to have the EA started, it is necessary to create the initial population of solutions. This is typically addressed by randomly generating the desired number of solutions. When the alphabet used for representing solutions has low cardinality, this random initialization provides a more or less uniform sample of the solution space. The EA can subsequently start exploring the wide area covered by the initial population, in search of the most promising regions.

In some cases, there exists the risk of not having the initial population adequately scattered all over the search space (e.g., when using small populations and/or large alphabets for representing solutions.) It is then necessary to resort to systematic initialization procedures [94], so as to ensure that all symbols are uniformly present in the initial population.

This random initialization can be complemented with the inclusion of heuristic solutions in the initial population. The EA can thus benefit from the existence of other algorithms, using the solutions they provide. This is termed *seeding*, and it is known to be very beneficial in terms of convergence speed, and quality of the solutions achieved [33, 92]. The potential drawback of this technique is having the injected solutions taking over the whole population in a few iterations, provoking the stagnation of the algorithm. This problem can be remedied by tuning the selection intensity by some means (e.g., by making an adequate choice of the selection operator, as it will be shown below).

4.3 Selection

In combination with replacement, selection is responsible for the competition aspects of individuals in the population. In fact, replacement can be intuitively regarded as the complementary application of the selection operation.

Using the information provided by the fitness function, a sample of individuals from the population is selected for breeding. This sample is obviously biased towards better individuals, i.e., good –according to the fitness function– solutions should be more likely in the sample than bad solutions[2].

The most popular techniques are fitness-proportionate methods. In these methods, the probability of selecting an individual for breeding is proportional to its fitness, i.e.

$$p_i = \frac{f_i}{\sum_{j \in P} f_j} \tag{1}$$

where f_i is the fitness[3] of individual i, and p_i is the probability of i getting into the reproduction stage. This proportional selection can be implemented in a number of ways. For example, *roulette-wheel selection* rolls a dice with $|P|$ sides, such that the ith side has probability p_i. This is repeated as many times as individuals are required in the sample. A drawback of this procedure is that

[2]At least, this is customary in genetic algorithms. In other EC families, selection is less important for biasing evolution, and it is done at random (a typical option in evolution strategies), or exhaustively, i.e., all individuals undergo reproduction (as it is typical in evolutionary programming).

[3]Maximization is assumed here. In case we were dealing with a minimization problem, fitness should be transformed so as to obtain an appropriate value for this purpose, e.g., subtracting it from the highest possible value of the guiding function, or taking the inverse of it.

the actual number of instances of individual i in the sample can largely deviate from the expected $|P| \cdot p_i$. *Stochastic Universal Sampling* [11] (SUS) does not have this problem, and produces a sample with minimal deviation from expected values.

Fitness-proportionate selection faces problems when the fitness values of individuals are very similar among them. In this case, p_i would be approximately $|P|^{-1}$ for all $i \in P$, and hence selection would be essentially random. This can be remedied by using fitness scaling. Typical options are (see [85]):

- Linear scaling: $f'_i = a \cdot f_i + b$, for some real numbers a, b.
- Exponential scaling: $f'_i = (f_i)^k$, for some real number k.
- Sigma truncation: $f'_i = \max\left(0, f_i - \left(\bar{f} - c \cdot \sigma\right)\right)$, where $\bar{f}$ is the mean fitness of individuals, σ is the fitness standard deviation, and c is a real number.

Another problem is the appearance of an individual whose fitness is much better than the remaining individuals. Such *super-individuals* can quickly take over the population. To avoid this, the best option is using a non-fitness-proportionate mechanism. A first possibility is *ranking selection* [105]: individuals are ranked according to fitness (best first, worst last), and later selected –e.g. by means of SUS– using the following probabilities

$$p_i = \frac{1}{|P|}\left[\eta^- + (\eta^+ - \eta^-)\frac{i-1}{|P|-1}\right], \tag{2}$$

where p_i is the probability of selecting the ith best individual, and $\eta^- + \eta^+ = 2$.

Another possibility is using *tournament selection* [17]. In this case, a direct competition is performed whenever an individual needs be selected. To be precise, α individuals are sampled at random, and the best of them is selected for reproduction. This is repeated as many times as needed. The parameter α is termed the *tournament size*; the higher this value, the stronger the selective pressure. These non-proportionate selection methods have the advantage of being unsensitive to fitness scaling problems and to the sense of optimization (maximization or minimization). The reader is referred to e.g. [43, 24] for a theoretical analysis of the properties of different selection operators.

Regardless of the selection operator used, it was implicity assumed in the previous discussion that any two individuals in the population can mate, i.e., all individuals belong to am unstructured centralized population. However, this is not necessarily the case. There exists a long tradition in using structured populations in EC, especially associated to parallel implementations. Among the most widely known types of structured EAs, *distributed* (dEA) and *cellular* (cEA) algorithms are very popular optimization procedures [7].

Decentralizing a single population can be achieved by partitioning it into several subpopulations, where component EAs are run performing sparse exchanges of individuals (distributed EAs), or in the form of neighborhoods (cellular EAs). The main difference is that a distributed EA has a large subpopulation, usually much larger than the single individual a cEA has typically in every component algorithm. In a dEA, the subpopulations are loosely coupled, while for a cEA they are tightly coupled. Additionally, in a dEA, there exist only a few subpopulations, while in a cEA there is a large number of them.

The use of decentralized populations has a great influence in te selection intensity, since not all individuals have to compete among them. As a consequence, diversity is often better preserved.

4.4 Recombination

Recombination is a process that models information exchange among several individuals (typically two of them, but a higher number is possible [47]). This is done by constructing new solutions using the information contained in a number of selected *parents*. If it is the case that the resulting individuals (the *offspring*) are entirely composed of information taken from the parents, then the recombination is said to be *transmitting* [39, 91]. This is the case of classical recombination operators for bitstrings such as *single-point crossover*, or *uniform crossover* [99], among others. Figure 4 shows an example of the application of these operators.

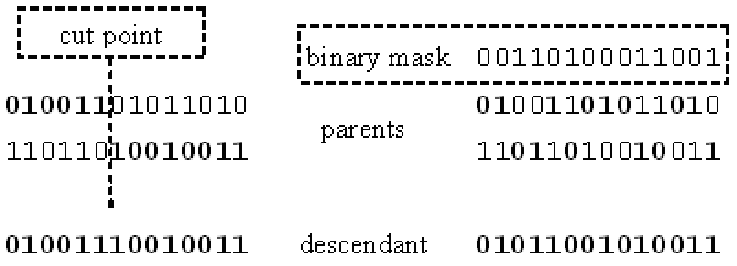

Figure 4: Two examples of recombination on bitstrings: single-point crossover (left) and uniform crossover (right).

This property captures the *a priori* role of recombination: combining good parts of solutions that have been independently discovered. It can be difficult to achieve for certain problem domains though (the *Traveling Salesman Problem* –TSP– is a typical example). In those situations, it is possible to consider other properties of interest such as *respect* or *assortment*. The former refers to the fact that the recombination operator generate descendants carrying all features common to all parents; thus, this property can be seen as a part of the *exploitative* side of the search. On the other hand, *assortment* represents the exploratory side of recombination. A recombination operator is said to be *properly assorting* if, and only if, it can generate descendants carrying any combination of compatible features taken from the parents. The assortment is said to be *weak* if it is necessary to perform several recombinations within the offspring to achieve this effect.

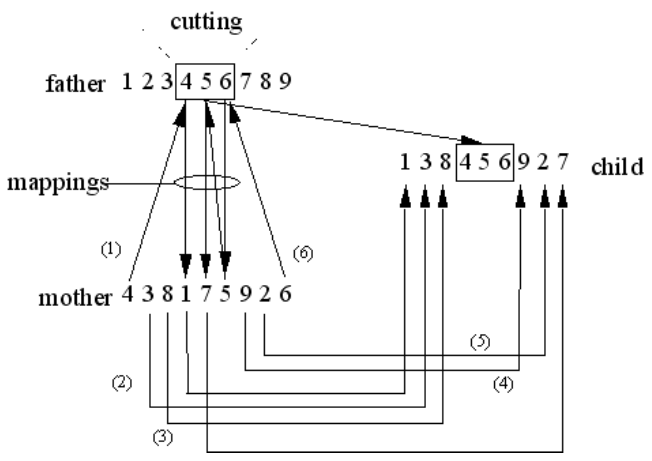

Figure 5: PMX at work. The numbers in brackets indicate the order in which elements are copied to the descendant.

The recombination operator must match the particulars of the representation of solutions chosen. In the GA context, the representation was typically binary, and hence operators such as those depicted in Figure 4 were used. The situation is different in other EA families (and indeed in modern GAs too). Without leaving GAs, another very typical representation is that of permutations. Many *ad hoc* operators have been defined for this purpose, e.g., order crossover (OX) [42], partially mapped crossover (PMX; see Figure 5) [59], and uniform cycle crossover (UCX) [35] among others. The reader may check [35] for a survey of these different operators.

When used in continuous parameter optimization, recombination can exploit the richness of the representation, and utilize a variety of alternate strategies to create the offspring. Let $(x_1, \cdots, x_n)$ and $(y_1, \cdots, y_n)$ be two arrays of real valued elements to be recombined, and let $(z_1, \cdots, z_n)$ be the resulting array. Some possibilities for performing recombination are the following:

- *Arithmetic recombination*: $z_i = (x_i + y_i)/2$, $1 \leq i \leq n$.
- *Geometric recombination*: $z_i = \sqrt{x_i y_i}$, $1 \leq i \leq n$.
- *Flat recombination*: $z_i = \alpha x_i + (1 - \alpha) y_i$, $1 \leq i \leq n$, where α is a random value in [0,1].
- *BLX-α recombination* [49]: $z_i = r_i + \beta(s_i - r_i)$, $1 \leq i \leq n$, where $r_i = \min(x_i, y_i) - \alpha|x_i - y_i|$, $s_i = \max(x_i, y_i) + \alpha|x_i - y_i|$, and β is a random value in [0,1].
- *Fuzzy recombination*: $z_i = Q(x_i, y_i)$, $1 \leq i \leq n$, where Q is a fuzzy connective [64].

In the case of self-adaptive schemes as those typically used in ES, the parameters undergoing self-adaption would be recombined as well, using some of these operators. More details on self-adaption will follow in next subsection.

Solutions can be also represented by means of some complex data structure, and the recombination operator must be adequately defined to deal with these (e.g., [4, 6, 37]). In particular, the field of GP normally uses trees to represent LISP programs [74], rule-bases [5], mathematical expressions, etc. Recombination is usually performed here by swapping branches of the trees involved, as exemplified in Figure 6.

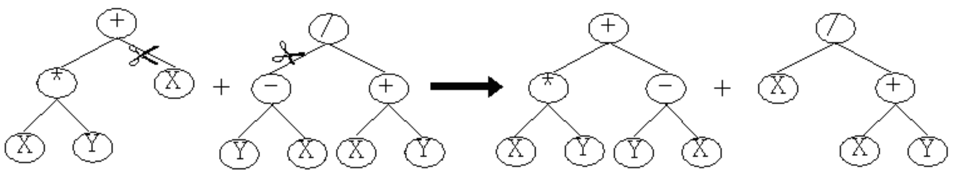

Figure 6: A example of branch-swapping recombination, as it is typically used in GP.

4.5 Mutation

From a classical point of view (at least in the GA arena [58]), this was a secondary operator whose mission is to *keep the pot boiling*, continuously injecting new material in the population, but at a low rate (otherwise the search would degrade to a random walk in the solution space). Evolutionary-programming practitioners [55] would disagree with this characterization, claiming a central role for mutation. Actually, it is considered the crucial part of the search engine in this context. This later vision has nowadays propagated to most EC researchers (at least in the sense of considering mutation as important as recombination).

As it was the case for recombination, the choice of a mutation operator depends on the representation used. In bitstrings (and in general, in linear strings spanning Σ^n, where Σ is arbitrary alphabet) mutation is done by randomly substituting the symbol contained at a certain position by a different symbol. If a permutation representation is used, such a procedure cannot be used for it would not produce a valid permutation. Typical strategies in this case are swapping two randomly chosen positions, or inverting a segment of the permutation. The interested reader may check [48] or [85] for an overview of different options.

If solutions are represented by complex data structures, mutation has to be implemented accordingly. In particular, this is the case of EP, in which e.g., finite automata [29], layered graphs [107], directed acyclic graphs [106], etc., are often evolved. In this domain, it is customary to use more than one mutation operator, making for each individual a choice of which operators will be deployed on it.

In the case of ES applied to continuous optimization, mutation is typically done using Gaussian perturbations, i.e.,

$$z_i = x_i + N_i(0, \sigma_i) \tag{3}$$

where σ_i is a parameter controlling the amplitude of the mutation, and $N(a, b)$ is a random number drawn from a normal distribution with mean a and standard deviation b. The parameters σ_i usually undergo self-adaption. In this case, they are mutated prior to mutating the x_i's as follows:

$$\sigma_i' = \sigma_i \cdot e^{N(0,\tau')+N_i(0,\tau)} \tag{4}$$

where τ and τ' are two parameters termed the *local* and *global learning rate* respectively. Advanced schemes have been also defined in which a covariance matrix is used rather than independent σ_i's. However, these schemes tend to be unpractical if solutions are highly dimensional. For a better understanding of ES mutation see [16].

4.6 Replacement

The role of replacement is keeping the population size constant[4]. To do so, some individuals from the population have to be substituted by some of the individuals created during reproduction. This can be done in several ways:

- *Replacement-of-the-worst*: the population is sorted according to fitness, and the new individuals replace the worst ones from the population.
- *Random replacement*: the individuals to be replaced are selected at random.
- *Tournament replacement*: a subset of α individuals is selected at random, and the worst one is selected for replacement. Notice that if $\alpha = 1$ we have random replacement.
- *Direct replacement*: the offspring replace their parents.

Some variants of these strategies are possible. For example, it is possible to consider the *elitist* versions of these, and only perform replacement if the new individual is better than the individual it has to replace.

Two replacement strategies (*comma* and *plus*) are also typically considered in the context of ES and EP. Comma replacement is analogous to replacement of the worst, with the addition that the number of new individuals $|P''|$ (also denoted by λ) can be larger that the population size $|P|$ (also denoted by μ). In this case, the population is constructed using the best μ out of the λ new individuals. As to the plus strategy, it would be the elitist counterpart of the former, i.e., pick the best μ individuals out of the μ old individuals plus the λ

[4]Although it is not mandatory to do so [50], it is common practice to use populations of fixed size.

new ones. The notation $(\mu, \lambda)-$EA and $(\mu + \lambda)-$EA is used to denote these two strategies.

It must be noted that the term 'elitism' is often used as well to denote replacement-of-the-worst strategies in which $|P''| < |P|$. This strategy is very commonly used, and ensures that the best individual found so far is never lost. An extreme situation takes place when $|P''| = 1$, i.e. just a single individual is generated in each iteration of the algorithm. This is known as *steady-state* reproduction, and it is usually associated with faster convergence of the algorithm. The term *generational* is used to designate the classical situation in which $|P''| = |P|$.

5 Fields of Application of EAs

EAs have been thoroughly used in many domains. One of the most conspicuous fields in which these techniques have been utilized is combinatorial optimization (CO). This way, EAs have been used to solve classical $NP-$hard problems such as the Travelling Salesman Problem [26, 52, 82], the Multiple Knapsack Problem [36, 72], Number Partitioning [15, 69], Max Independent Set [1, 65], and Graph Coloring [32, 51], among others.

Other non-classical –yet important– CO problems to which EAs have been applied are scheduling (in many variants [25, 31, 35, 80, 88]), timetabling [19, 89], lot-sizing [63], vehicle routing [13, 14], quadratic assignment [83, 84], placement problems [67, 75], and transportation problems [56].

Telecommunications is another field that has witnessed the successful application of EAs. For example, EAs have been applied to the placement of antennas and converters [23, 102], frequency assignment [38, 46, 71], digital data network design [28], predicting bandwidth demands in ATM networks [98], error code design [27, 44], etc. See also [30].

EAs have been actively used in electronics and engineering as well. For example, work has been done in structure optimization [108], aeronautic design [90], power planning [101], circuit design [62] computer aided design [12], analogue network synthesis [61], and service restoration [8] among other areas.

Besides the precise application areas mentioned before, EAs have been also utilized in many other fields such as, e.g, medicine [22, 103], economics [2, 79], mathematics [95, 104], biology [34, 54, 100], etc. The reader may try querying any bibliographical database or web search engine for "evolutionary algorithm application" to get an idea of the vast number of problems that have been tackled with EAs.

6 Conclusions

EC is a fascinating field. Its optimization philosophy is appealing, and its practical power is striking. Whenever the user is faced with a hard search/optimization task that she cannot solve by classical means, trying EAs is a must. The extremely brief overview of EA applications presented before can convince the reader that a “killer approach” is in her hands.

EC is also a very active research field. One of the main weaknesses of the field is the absence of a conclusive general theoretical basis, although great advances are being made in this direction, and in-depth knowledge is available about certain idealized EA models.

Regarding the more practical aspects of the paradigm, two main streamlines can be identified: parallelizing and hybridizing. The use of decentralized EAs in the context of multiprocessors or networked systems can result in enormous performance improvement [3], and constitutes an ideal option for exploiting the availability of distributed computing resources. As to hybridization, it has become evident in the last years that it constitutes a crucial factor for the successful use of EAs in real-world endeavors. This can be achieved by hard-wiring problem-knowledge within the EA, or by combining it with other techniques. In this sense, the reader is encouraged to read other essays in this volume to get valuable ideas on suitable candidates for this hybridization.

Acknowledgements

This work has been partially funded by the Ministry of Science and Technology (MCYT) and Regional Development European Found (FEDER) under contract TIC2002-04498-C05-02 (the TRACER project) `http://tracer.lcc.uma.es`.

References

[1] C.C. Aggarwal, J.B. Orlin, and R.P. Tai. Optimized crossover for the independent set problem. *Operations Research*, 45(2):226–234, 1997.

[2] J. Alander. Indexed bibliography of genetic algorithms in economics. Technical Report 94-1-ECO, University of Vaasa, Department of Information Technology and Production Economics, 1995.

[3] E. Alba. Parallel evolutionary algorithms can achieve super-linear performance. *Information Processing Letters*, 82(1):7–13, 2002.

[4] E. Alba, J.F. Aldana, and J.M. Troya. Full automatic ann design: A genetic approach. In J. Cabestany J. Mira and A. Prieto, editors, *New Trends in Neural Computation*, volume 686 of *Lecture Notes in Computer Science*, pages 399–404. Springer-Verlag, 1993.

[5] E. Alba, C. Cotta, and J.M. Troya. Evolutionary design of fuzzy logic controllers using strongly-typed GP. *Mathware & Soft Computing*, 6:109–124, 1999.

[6] E. Alba and J.M. Troya. Genetic algorithms for protocol validation. In H.M. Voigt, W. Ebeling, I. Rechenberg, and H.-P. Schwefel, editors,

Parallel Problem Solving From Nature IV, pages 870–879, Berlin, 1996. Springer-Verlag.

[7] E. Alba and J.M. Troya. A survey of parallel distributed genetic algorithms. *Complexity*, 4(4):31–52, 1999.

[8] A. Augugliaro, L. Dusonchet, and E. Riva-Sanseverino. Service restoration in compensated distribution networks using a hybrid genetic algorithm. *Electric Power Systems Research*, 46(1):59–66, 1998.

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

[10] T. Bäck, D.B. Fogel, and Z. Michalewicz. *Handbook of Evolutionary Computation.* Oxford University Press, New York NY, 1997.

[11] J.E. Baker. Reducing bias and inefficiency in the selection algorithm. In J.J. Grefenstette, editor, *Proceedings of the Second International Conference on Genetic Algorithms*, pages 14–21, Hillsdale NJ, 1987. Lawrence Erlbaum Associates.

[12] B. Becker and R. Drechsler. Ofdd based minimization of fixed polarity Reed-Muller expressions using hybrid genetic algorithms. In *Proceedings IEEE International Conference on Computer Design: VLSI in Computers and Processor*, pages 106–110, Los Alamitos, CA, 1994. IEEE.

[13] J. Berger and M. Barkaoui. A hybrid genetic algorithm for the capacitated vehicle routing problem. In E. Cantú-Paz, editor, *Proceedings of the Genetic and Evolutionary Computation Conference 2003*, number 2723 in Lecture Notes in Computer Science, pages 646–656, Berlin Heidelberg, 2003. Springer-Verlag.

[14] J. Berger, M. Salois, and R. Begin. A hybrid genetic algorithm for the vehicle routing problem with time windows. In R.E. Mercer and E. Neufeld, editors, *Advances in Artificial Intelligence. 12th Biennial Conference of the Canadian Society for Computational Studies of Intelligence*, pages 114–127, Berlin, 1998. Springer-Verlag.

[15] R. Berretta, C. Cotta, and P. Moscato. Enhancing the performance of memetic algorithms by using a matching-based recombination algorithm: Results on the number partitioning problem. In M. Resende and J. Pinho de Sousa, editors, *Metaheuristics: Computer-Decision Making*, pages 65–90. Kluwer Academic Publishers, Boston MA, 2003.

[16] H.-G. Beyer. *The Theory of Evolution Strategies.* Springer-Verlag, Berlin Heidelberg, 2001.

[17] T. Bickle and L. Thiele. A mathematical analysis of tournament selection. In L.J. Eshelman, editor, *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 9–16, San Francisco CA, 1995. Morgan Kaufmann.

[18] C. Blum and A. Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, 35(3):268–308, 2003.

[19] E.K. Burke, J.P. Newall, and R.F. Weare. Initialisation strategies and diversity in evolutionary timetabling. *Evolutionary Computation*, 6(1):81–103, 1998.

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

[21] Coello C.A. and A.D. Christiansen. An approach to multiobjective optimization using genetic algorithms. In C.H. Dagli, M. Akay, C.L.P. Chen, B. R. Fernández, and J. Ghosh, editors, *Intelligent Engineering Systems Through Artificial Neural Networks*, volume 5, pages 411–416. ASME Press, St. Louis, Missouri, 1995.

[22] M. Sipper C.A. Peña Reyes. Evolutionary computation in medicine: An overview. *Artificial Intelligence in Medicine*, 19(1):1–23, 2000.

[23] P. Calegar, F. Guidec, P. Kuonen, and D. Wagner. Parallel island-based genetic algorithm for radio network design. *Journal of Parallel and Distributed Computing*, 47(1):86–90, 1997.

[24] E. Cantú-Paz. Order statistics and selection methods of evolutionary algorithms. *Information Processing Letters*, 82(1):15–22, 2002.

[25] S. Cavalieri and P. Gaiardelli. Hybrid genetic algorithms for a multiple-objective scheduling problem. *Journal of Intelligent Manufacturing*, 9(4):361–367, 1998.

[26] S. Chatterjee, C. Carrera, and L. Lynch. Genetic algorithms and traveling salesman problems. *European Journal of Operational Research*, 93(3):490–510, 1996.

[27] H. Chen, N.S. Flann, and D.W. Watson. Parallel genetic simulated annealing: A massively parallel SIMD algorithm. *IEEE Transactions on Parallel and Distributed Systems*, 9(2):126–136, 1998.

[28] C.H. Chu, G. Premkumar, and H. Chou. Digital data networks design using genetic algorithms. *European Journal of Operational Research*, 127:140–158, 2000.

[29] C.H. Clelland and D.A. Newlands. PFSA modelling of behavioural sequences by evolutionary programming. In R.J. Stonier and X.H. Yu, editors, *Complex Systems: Mechanism for Adaptation*, pages 165–172. IOS Press, 1994.

[30] D. W. Corne, M. J. Oates, and G. D. Smith. *Telecommunications Optimization: Heuristic and Adaptive Techniques.* Wiley, 2000.

[31] D. Costa. An evolutionary tabu search algorithm and the NHL scheduling problem. *INFOR*, 33(3):161–178, 1995.

[32] D. Costa, N. Dubuis, and A. Hertz. Embedding of a sequential procedure within an evolutionary algorithm for coloring problems in graphs. *Journal of Heuristics*, 1(1):105–128, 1995.

[33] C. Cotta. On the evolutionary inference of temporal boolean networks. In J. Mira and J.R. Álvarez, editors, *Computational Methods in Neural Modeling*, volume 2686 of *Lecture Notes in Computer Science*, pages 494–501, Berlin Heidelberg, 2003. Springer-Verlag.

[34] C. Cotta and P. Moscato. Inferring phylogenetic trees using evolutionary algorithms. In J.J. Merelo, P. Adamidis, H.-G. Beyer, J.-L. Fernández-Villacañas, and H.-P. Schwefel, editors, *Parallel Problem Solving From Nature VII*, volume 2439 of *Lecture Notes in Computer Science*, pages 720–729. Springer-Verlag, Berlin, 2002.

[35] C. Cotta and J.M. Troya. Genetic forma recombination in permutation flowshop problems. *Evolutionary Computation*, 6(1):25–44, 1998.

[36] C. Cotta and J.M. Troya. A hybrid genetic algorithm for the 0-1 multiple knapsack problem. In G.D. Smith, N.C. Steele, and R.F. Albrecht, editors, *Artificial Neural Nets and Genetic Algorithms 3*, pages 251–255, Wien New York, 1998. Springer-Verlag.

[37] C. Cotta and J.M. Troya. Analyzing directed acyclic graph recombination. In B. Reusch, editor, *Computational Intelligence: Theory and Applications*, volume 2206 of *Lecture Notes in Computer Science*, pages 739–748. Springer-Verlag, Berlin Heidelberg, 2001.

[38] C. Cotta and J.M. Troya. A comparison of several evolutionary heuristics for the frequency assignment problem. In J. Mira and A. Prieto, editors, *Connectionist Models of Neurons, Learning Processes, and Artificial Intelligence*, volume 2084 of *Lecture Notes in Computer Science*, pages 709–716. Springer-Verlag, Berlin Heidelberg, 2001.

[39] C. Cotta and J.M. Troya. Information processing in transmitting recombination. *Applied Mathematics Letters*, 16(6):945–948, 2003.

[40] M.L. Cramer. A representation for the adaptive generation of simple sequential programs. In J.J. Grefenstette, editor, *Proceedings of the First International Conference on Genetic Algorithms*, Hillsdale NJ, 1985. Lawrence Erlbaum Associates.

[41] C. Darwin. *On the Origin of Species by Means of Natural Selection.* John Murray, London, 1859.

[42] L. Davis. *Handbook of Genetic Algorithms*. Van Nostrand Reinhold Computer Library, New York, 1991.

[43] K. Deb and D. Goldberg. A comparative analysis of selection schemes used in genetic algorithms. In G.J. Rawlins, editor, *Foundations of Genetic Algorithms*, pages 69–93, San Mateo, CA, 1991.

[44] K. Dontas and K. De Jong. Discovery of maximal distance codes using genetic algorithms. In *Proceedings of the Second International IEEE Conference on Tools for Artificial Intelligence*, pages 905–811, Herndon, VA, 1990. IEEE Press.

[45] M. Dorigo and G. Di Caro. The ant colony optimization meta-heuristic. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization*, pages 11–32. McGraw-Hill, 1999.

[46] R. Dorne and J.K Hao. An evolutionary approach for frequency assignment in cellular radio networks. In *1995 IEEE International Conference on Evolutionary Computation*, pages 539–544, Perth, Australia, 1995. IEEE Press.

[47] A.E. Eiben, P.-E. Raue, and Zs. Ruttkay. Genetic algorithms with multi-parent recombination. In Y. Davidor, H.-P. Schwefel, and R. Männer, editors, *Parallel Problem Solving From Nature III*, volume 866 of *Lecture Notes in Computer Science*, pages 78–87. Springer-Verlag, 1994.

[48] A.E. Eiben and J.E. Smith. *Introduction to Evolutionary Computing*. Springer-Verlag, Berlin Heidelberg, 2003.

[49] L.J. Eshelman and J.D. Schaffer. Real-coded genetic algorithms and interval-schemata. In D. Whitley, editor, *Foundations of Genetic Algorithms 2*, pages 187–202, San Mateo CA, 1993. Morgan Kaufman Publishers.

[50] F. Fernandez, L. Vanneschi, and M. Tomassini. The effect of plagues in genetic programming: A study of variable-size populations. In C. Ryan et al., editors, *Genetic Programming, Proceedings of EuroGP'2003*, volume 2610 of *Lecture Notes in Computer Science*, pages 320–329, Berln Heidelberg, 2003. Springer-Verlag.

[51] C. Fleurent and J.A. Ferland. Genetic and hybrid algorithms for graph coloring. *Annals of Operations Research*, 63:437–461, 1997.

[52] D.B. Fogel. An evolutionary approach to the traveling salesman problem. *Biological Cybernetics*, 60(2):139–144, 1988.

[53] D.B. Fogel. *Evolutionary Computation: The Fossil Record*. Wiley-IEEE Press, Piscataway, NJ, 1998.

[54] G.B. Fogel and D.W. Corne. *Evolutionary Computation in Bioinformatics.* Morgan Kaufmann, 2003.

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

[56] M. Gen, K. Ida, and L. Yinzhen. Bicriteria transportation problem by hybrid genetic algorithm. *Computers & Industrial Engineering*, 35(1-2):363–366, 1998.

[57] F. Glover and G. Kochenberger. *Handbook of Metaheuristics.* Kluwer Academic Publishers, Boston MA, 2003.

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

[59] D.E. Goldberg and R. Lingle Jr. Alleles, loci and the traveling salesman problem. In J.J. Grefenstette, editor, *Proceedings of an International Conference on Genetic Algorithms*, Hillsdale NJ, 1985. Lawrence Erlbaum Associates.

[60] S.J. Gould and N. Elredge. Punctuated equilibria: The tempo and mode of evolution reconsidered. *Paleobiology*, 32:115–151, 1977.

[61] J.B. Grimbleby. Hybrid genetic algorithms for analogue network synthesis. In *Proceedings of the 1999 Congress on Evolutionary Computation*, pages 1781–1787, Washington D.C., 1999. IEEE.

[62] M. Guotian and L. Changhong. Optimal design of the broadband stepped impedance transformer based on the hybrid genetic algorithm. *Journal of Xidian University*, 26(1):8–12, 1999.

[63] K. Haase and U. Kohlmorgen. Parallel genetic algorithm for the capacitated lot-sizing problem. In Kleinschmidt et al., editors, *Operations Research Proceedings*, pages 370–375. Springer-Verlag, 1996.

[64] F. Herrera, M. Lozano, and J.L. Verdegay. Dynamic and heuristic fuzzy connectives-based crossover operators for controlling the diversity and convengence of real coded genetic algorithms. *Journal of Intelligent Systems*, 11:1013–1041, 1996.

[65] M. Hifi. A genetic algorithm-based heuristic for solving the weighted maximum independent set and some equivalent problems. *Journal of the Operational Research Society*, 48(6):612–622, 1997.

[66] J.H. Holland. *Adaptation in Natural and Artificial Systems.* University of Michigan Press, Ann Harbor, 1975.

[67] E. Hopper and B. Turton. A genetic algorithm for a 2d industrial packing problem. *Computers & Industrial Engineering*, 37(1-2):375–378, 1999.

[68] J. Huxley. *Evolution, the Modern Synthesis.* Harper, New York NY, 1942.

[69] D.R. Jones and M.A. Beltramo. Solving partitioning problems with genetic algorithms. In R.K Belew and L.B. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 442–449, San Mateo, CA, 1991. Morgan Kaufmann.

[70] T.C. Jones. *Evolutionary Algorithms, Fitness Landscapes and Search.* PhD thesis, University of New Mexico, 1995.

[71] A. Kapsalis, V.J. Rayward-Smith, and G.D. Smith. Using genetic algorithms to solve the radio link frequency assigment problem. In D.W. Pearson, N.C. Steele, and R.F. Albretch, editors, *Artificial Neural Nets and Genetic Algorithms*, pages 37–40, Wien New York, 1995. Springer-Verlag.

[72] S. Khuri, T. Bäck, and J. Heitkötter. The zero/one multiple knapsack problem and genetic algorithms. In E. Deaton, D. Oppenheim, J. Urban, and H. Berghel, editors, *Proc. of the 1994 ACM Symposium of Applied Computation proceedings*, pages 188–193. ACM Press, 1994.

[73] M. Kimura. Evolutionary rate at the molecular level. *Nature*, 217:624–626, 1968.

[74] J.R. Koza. *Genetic Programming.* MIT Press, Cambridge MA, 1992.

[75] R.M. Krzanowski and J. Raper. Hybrid genetic algorithm for transmitter location in wireless networks. *Computers, Environment and Urban Systems*, 23(5):359–382, 1999.

[76] M. Laguna and R. Martí. *Scatter Search. Methodology and Implementations in C.* Kluwer Academic Publishers, Boston MA, 2003.

[77] C.G. Langton. Artificial life. In C.G. Langton, editor, *Artificial Life 1*, pages 1–47, Santa Fe NM, 1989. Addison-Wesley.

[78] P. Larrañaga and J.A. Lozano. *Estimation of Distribution Algorithms. A New Tool for Evolutionary Computation.* Kluwer Academic Publishers, Boston MA, 2001.

[79] F. Li, R. Morgan, and D. Williams. Economic environmental dispatch made easy with hybrid genetic algorithms. In *Proceedings of the International Conference on Electrical Engineering*, volume 2, pages 965–969, Beijing, China, 1996. Int. Acad. Publishers.

[80] C.F. Liaw. A hybrid genetic algorithm for the open shop scheduling problem. *European Journal of Oprational Research*, 124(1):28–42, 2000.

[81] G. Mendel. Versuche über pflanzen-hybriden. *Verhandlungen des Naturforschendes Vereines in Brünn*, 4:3–47, 1865.

[82] P. Merz and B. Freisleben. Genetic Local Search for the TSP: New Results. In *Proceedings of the 1997 IEEE International Conference on Evolutionary Computation*, pages 159–164. IEEE Press, 1997.

[83] P. Merz and B. Freisleben. Genetic algorithms for binary quadratic programming. In W. Banzhaf et al., editors, *Proceedings of the 1999 Genetic and Evolutionary Computation Conference*, pages 417–424. Morgan Kauffman, 1999.

[84] P. Merz and B. Freisleben. Fitness landscape analysis and memetic algorithms for the quadratic assignment problem. *IEEE Transactions on Evolutionary Computation*, 4(4):337–352, 2000.

[85] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs.* Springer-Verlag, Berlin, 1992.

[86] P. Moscato. On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts: Towards Memetic Algorithms. Technical Report Caltech Concurrent Computation Program, Report. 826, California Institute of Technology, Pasadena, California, USA, 1989.

[87] P. Moscato and C. Cotta. A gentle introduction to memetic algorithms. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, pages 105–144. Kluwer Academic Publishers, Boston MA, 2003.

[88] L. Ozdamar. A genetic algorithm approach to a general category project scheduling problem. *IEEE Transactions on Systems, Man and Cybernetics, Part C (Applications and Reviews)*, 29(1):44–59, 1999.

[89] B. Paechter, R.C. Rankin, and A. Cumming. Improving a lecture timetabling system for university wide use. In E.K. Burke and M. Carter, editors, *The Practice and Theory of Automated Timetabling II*, volume 1408 of *Lecture Notes in Computer Science*, pages 156–165. Springer Verlag, 1998.

[90] D. Quagliarella and A. Vicini. Hybrid genetic algorithms as tools for complex optimisation problems. In P. Blonda, M. Castellano, and A. Petrosino, editors, *New Trends in Fuzzy Logic II. Proceedings of the Second Italian Workshop on Fuzzy Logic*, pages 300–307, Singapore, 1998. World Scientific.

[91] N.J. Radcliffe. The algebra of genetic algorithms. *Annals of Mathematics and Artificial Intelligence*, 10:339–384, 1994.

[92] C. Ramsey and J.J. Grefensttete. Case-based initialization of genetic algorithms. In S. Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 84–91, San Mateo CA, 1993. Morgan Kauffman.

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

[94] C.R. Reeves. Using genetic algorithms with small populations. In S. Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 92–99, San Mateo CA, 1993. Morgan Kaufmann.

[95] C. Reich. Simulation if imprecise ordinary differential equations using evolutionary algorithms. In J. Carroll, E. Damiani, H. Haddad, and D. Oppenheim, editors, *ACM Symposium on Applied Computing 2000*, pages 428–432. ACM Press, 2000.

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

[97] R.E. Smith. Diploid genetic algorithms for search in time varying environments. In *Annual Southeast Regional Conference of the ACM*, pages 175–179, New York NY, 1987. ACM Press.

[98] N. Swaminathan, J. Srinivasan, and S.V. Raghavan. Bandwidth-demand prediction in virtual path in atm networks using genetic algorithms. *Computer Commnunications*, 22(12):1127–1135, 1999.

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

[100] R. Thomsen, G.B. Fogel, and T. Krink. A clustal alignment improver using evolutionary algorithms. In David B. Fogel, Xin Yao, Garry Greenwood, Hitoshi Iba, Paul Marrow, and Mark Shackleton, editors, *Proceedings of the Fourth Congress on Evolutionary Computation (CEC-2002)*, volume 1, pages 121–126, 2002.

[101] A.J. Urdaneta, J.F. Gómez, E. Sorrentino, L. Flores, and R. Díaz. A hybrid genetic algorithm for optimal reactive power planning based upon successive linear programming. *IEEE Transactions on Power Systems*, 14(4):1292–1298, 1999.

[102] C. Vijayanand, M. S. Kumar, K. R. Venugopal, and P. S. Kumar. Converter placement in all-optical networks using genetic algorithms. *Computer Communications*, 23:1223–1234, 2000.

[103] R. Wehrens, C. Lucasius, L. Buydens, and G. Kateman. HIPS, A hybrid self-adapting expert system for nuclear magnetic resonance spectrum interpretation using genetic algorithms. *Analytica Chimica ACTA*, 277(2):313–324, May 1993.

[104] X. Wei and F. Kangling. A hybrid genetic algorithm for global solution of nondifferentiable nonlinear function. *Control Theory & Applications*, 17(2):180–183, 2000.

[105] D.L. Whitley. Using reproductive evaluation to improve genetic search and heuris-tic discovery. In J.J. Grefenstette, editor, *Proceedings of the Second International Conference on Genetic Algorithms*, pages 116–121, Hillsdale NJ, 1987. Lawrence Erlbaum Associates.

[106] M.L. Wong, W. Lam, and K.S. Leung. Using evolutionary programming and minimum description length principle for data mining of bayesian networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(2):174–178, 1999.

[107] X. Yao and Y. Liu. A new evolutionary system for evolving artificial neural networks. *IEEE Transactions on Neural Networks*, 8(3):694–713, 1997.

[108] I.C. Yeh. Hybrid genetic algorithms for optimization of truss structures. *Computer Aided Civil and Infrastructure Engineering*, 14(3):199–206, 1999.