Chapter 1

Evolutionary Computation: Historical View and Basic Concepts

Carlos A. Coello Coello[a], Carlos Segura[b], Gara Miranda[c]

[a]Departamento de Computación (Evolutionary Computation Group), CINVESTAV-IPN México D.F., Mexico.

[b]Centro de Investigación en Matemáticas, Área Computación, Callejón Jalisco s/n, Mineral de Valenciana, Guanajuato, Mexico.

[c]Dpto. Estadística, I. O. y Computación, Universidad de La Laguna, Santa Cruz de Tenerife, Spain.

1.1 Introduction

In most real-world problems, achieving optimal (or at least good) solutions results in saved resources, time and expenses. These problems where optimal solutions are desired are known as *optimization problems*. In general, optimization algorithms search among the set of possible solutions (*search space*), evaluating the candidates —perhaps also analyzing their feasibility— and choosing from among them the final solution —or solutions— to the problem. Another perspective is that an optimization problem consists of finding the values of those variables that minimize/maximize the objective functions while satisfying any existing constraints. All optimization problems involve *variables* that somehow define the search space. However, *constraints* are not mandatory: many unconstrained problems have been formulated and studied in the literature. In the case of the *objective function*, most optimization problems have been formulated on the basis of a single objective function. However, in many problems it is actually necessary to optimize several different objectives at once [Coello Coello *et al.* (2007)]. Usually, such objectives are mutually competitive or conflicting, and the values of the variables that optimize one objective may be far from optimal for the others. Thus, in multi-objective optimization there is no single and unique optimal solution, but rather a set of optimal solutions, none of them better than the others.

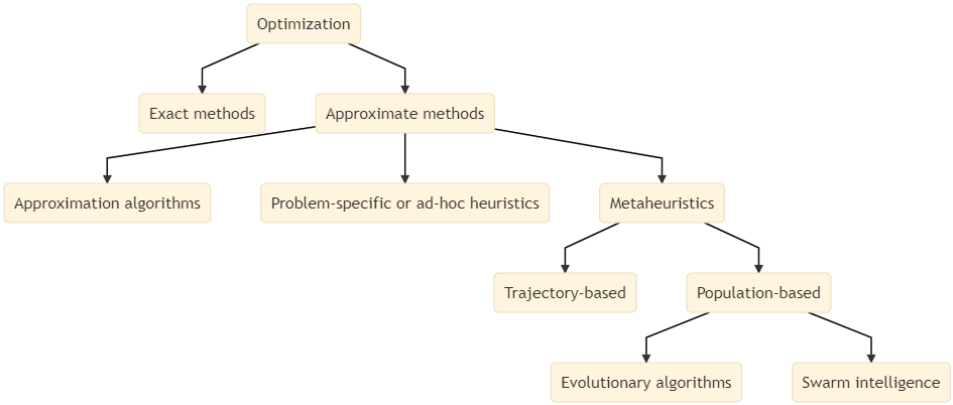

Fig. 1.1 Classification of optimization methods

Optimization problems can be simple or more complex depending on the number of objectives, the solution space, the number of decision variables, the constraints specified, etc. Some optimization problems that satisfy certain properties are relatively easy to solve [Nocedal and Wright (2006)]. However, many optimization problems are quite difficult to solve; in fact many of them fit into the family of *NP-hard problems* [Brassard and Bratley (1996)]. Assuming that $P \neq NP$, there is no approach which guarantees that the optimal solution for these problems is achieved in polynomial time. In the same way, optimization algorithms can be very simple or complex depending on whether they are able to give exact or approximate solutions, whether they are less or more efficient or even whether they need to be specifically designed for a particular problem or not. Moreover, algorithms can be also classified by the amount of completion time required in comparison to their input size. Some problems may have multiple related algorithms of differing complexity, while other problems might have no algorithms, or, at least, no known efficient algorithms. Taking the above into account, when deciding which algorithmic technique to apply, much depends on the properties of the problem to be solved, but also on the expected results: the quality of the solution, efficiency of the approach, flexibility and generality of the method, etc.

Since many optimization algorithms have been proposed, with different purposes and from different perspectives, there is no single criterion that can be used to classify them. We can find different categories of algorithms in the literature depending on the features related to their internal implementation, their general behavior scheme, the field of application, complexity, etc. For the purposes of this chapter, the classification presented in Figure 1.1 provides a meaningful starting point.

In general, most solution techniques involve some form of exploration of the set

of feasible solutions. *Exact approaches* are usually based on an enumerative, exhaustive or brute-force search that yields the optimal solution to the problem. However, in most problems dealing with real or large instances, the search space may be far too large, or there may not even exist a convenient way to enumerate it. In such situations, it is common to resort to some kind of *approximate method*. These approaches do not generally ensure optimal solutions, but some of them at least ensure a certain level of solution quality, e.g. *approximation algorithms*. *Ad-hoc heuristics* are another kind of approximate method that incorporate information about the problem at hand in order to decide which candidate solution should be tested next or how the next candidate can be produced [Weise (2008)]. Since they are based on specific knowledge of the problem, such ad-hoc methods have the drawback of being problem dependent. That is, a wide variety of heuristics can be specifically and successfully designed to optimize a given problem, but unfortunately, they cannot be directly extrapolated to another problem. The specific mechanisms or decisions that work well in one case may not work at all for related problems that share common features, e.g. problems belonging to the same family of problems. As a result, such procedures are unable to adapt to particular problem instances or to extend to different problems. Re-starting and randomization strategies, as well as combinations of simple heuristics, offer only partial and largely unsatisfactory answers to these issues [Glover and Kochenberger (2003)].

Metaheuristics appear as a class of modern heuristics whose main goal is to address these open challenges [Talbi (2009)]. A metaheuristic is an approximate method for solving a very general class of problems. It combines objective functions and heuristics in an abstract and hopefully efficient way, usually without requiring a deeper insight into their structure. Metaheuristics have been defined as master strategies to guide and modify other heuristics to produce solutions beyond those normally identified by ad-hoc heuristics. Compared to exact search methods, metaheuristics are not able to ensure a systematic exploration of the entire solution space, so they cannot guarantee that optimal solutions will be achieved. Instead, with the aim of increasing efficiency, they seek to examine only parts where "good" solutions may be found.

In most metaheuristics, a single solution, or a set of them, is maintained as active solutions and the search procedures are based on movements within the search space. New candidate solutions are built mainly in two ways: from scratch or as an evolution of current solutions. In many cases, these new candidate solutions are generated by means of a neighborhood definition, although different strategies are also used. At each process iteration, the metaheuristic evaluates different kinds of moves, some of which are selected to update the current solutions as determined by certain criteria (objective value, feasibility, statistical measures, etc. [Glover and Kochenberger (2003)]. Well-designed metaheuristics try to avoid being trapped in local optima or to repeatedly cycle through solutions that were already visited. It is also important to provide a reasonable assurance that the search does not overlook

promising regions.

A wide variety of metaheuristics have been proposed in the literature. In many cases, these criteria or combinations of moves are often performed stochastically by utilizing statistics obtained from samples from the search space, or based on a model of some natural phenomenon or physical process [Blum and Roli (2003)]. Another interesting feature that arises when analyzing the internal operation of a metaheuristic involves the number of candidate solutions or states that are being managed at a time. *Trajectory-based metaheuristics* maintain a single current state at any given instant, which is replaced by a new one at each generation. In contrast, *population-based metaheuristics* maintain a current pool with several candidate states instead of a single current state.

As described before, in its search for better solutions and greater quality, the research in the field of optimization has focused its attention on the design of metaheuristics as general purpose techniques to guide the construction of solutions with the aim of improving the resulting quality. In recent decades, research in this field has led to the creation of new hybrid algorithms, by combining different concepts from various fields such as genetics, biology, artificial intelligence, mathematics, physics, and neurology, among others. In this context, Evolutionary Computation (EC) is a research area within Computer Science that studies the properties of a set of algorithms —usually called Evolutionary Algorithms (EAs)— that draw their inspiration from natural evolution. Initially, several algorithms intended to better understand the population dynamics present in evolution were designed [Crosby (1967)]. However, although the design of the EC schemes is based on drawing inspiration from natural evolution, a faithful modeling of biologic processes is not usually incorporated. Thus, since EAs are usually overly simplistic versions of their biological counterparts, using EAs to model population dynamics is not too widely accepted by the evolutionary biology community [Mitchell (1998)]. Instead, the advances achieved in recent decades have shown that the main strength of EC is that it can be successfully applied to numerous practical problems that appear in several areas such as process control, machine learning and function optimization [Fogel (1995)]. For this reason, EC should be seen as a general and robust evolution-inspired framework devoted to problem-solving.

Although it is not easy to classify the kinds of problems that can be solved with EC, some taxonomies have been proposed. For instance, Everett distinguished between two main uses of EAs [Everett (2000)]: optimizing the performance of operating systems, and the testing and fitting of quantitative models. Another view was proposed by Eiben and Smith [Eiben and Smith (2003)] by providing an analogy between problems and systems. In working systems, the three main components are: inputs, outputs and the internal model connecting these two. They distinguished between three kinds of problems that can be solved with EC depending on which of the three components is unknown. In any case, the important fact is that EC has been successfully used in a huge number of applications such as numerical

optimization, design of expert systems, training of neural networks, data mining and many others. In general, EC might be potentially applied to any problem where we can identify a set of candidate solutions and a quality level associated with each of them.

This chapter is devoted to presenting the history and philosophy behind the use of EC, as well as introducing the reader to the design of EAs. In Section 1.2, a brief review of natural evolution theories is provided. In addition, some of the main concepts from the fields of evolution and genetics that have inspired developments in EC are introduced. Then, Section 1.3 presents the history and philosophy of the first EC approaches ever developed. Some of the latest developments, as well as the current unifying view of EC, are summarized in Section 1.4. Section 1.5 offers some observations on the design of EAs and describes some of the most popular components that have been used when tackling optimization problems with EC. Finally, some concluding remarks are given in Section 1.6.

1.2 The Fundamentals of Evolution

The term "*evolution*" is defined by the Oxford Dictionary as *the process by which different kinds of living organisms are thought to have developed and diversified from earlier forms during the history of the Earth.* A much more general definition of this term is *the gradual development of something, especially from a simple to a more complex form.* The word "*evolution*" originated from the Latin word "*evolutio*", which means unrolling, from the verb "*evolvere*". Early meanings related to physical movement, first recorded in describing a tactical "wheeling" maneuver in the realignment of troops or ships. Current senses stem from a notion of "opening out" and "unfolding", giving rise to a general sense of "development". The term appeared a couple of centuries before Darwin wrote "*On the Origin of Species*". In fact, Darwin did not even use the word evolution in his book until the last line:

> *There is grandeur in this view of life, with its several powers, having been originally breathed by the Creator into a few forms or into one; and that, whilst this planet has gone circling on according to the fixed law of gravity, from so simple a beginning endless forms most beautiful and most wonderful have been, and are being* ***evolved****.*

1.2.1 *A Historical Review*

During the eighteenth century, a group of researchers, called naturalists, managed to gather a great deal of information on the flora and fauna in many different areas of our planet. In an attempt to organize and classify this remarkable amount of information, **Carl Linnaeus** (1707-1778) proposed a set of rules to assign genus and species labels to all known living beings. His taxonomy, called *System Nat-*

urae [Linnaeus (1735)], focused solely on the morphological properties of living beings to define the classification. Since this classification criterion was purely morphological, to Linnaeus species identified distinct groups with no relation of origin. This perspective, called *fixity of species*, considered that each species was created as it was, and individuals did not experience changes over time. Linnaeus began his study believing this concept, but later rejected it after observing interbreeding between various species. Due to his extensive work in the field, he is considered the founding father of our current taxonomic system.

The accumulation of information provided by naturalists and the progress achieved in the taxonomies led to the adoption of new approaches, different from the fixity of species, and based on the fact that some species came from other species. This idea required defining a new classification that reflected the relationships among organisms. It was called "*natural classification*". Although **Georges-Louis Leclerc**, Comte de Buffon (1707-1788), was the first to question Linnaeus's fixity of species, the first to propose a hypothesis for how one species could come from another was **Jean-Baptiste Pierre Antoine de Monet**, Chevalier de Lamarck (1744-1829). Lamarck, in his *Zoological Philosophy* [de Monet de Lamarck (1809)] presented a systematic description for the evolution of living beings. For Lamarck, species develop as a result of their reaction and adaptation to the environment. These changes, therefore, must be gradual and will occur over long periods of time. Lamarck believed that certain organs are strengthened by the use that animals make of them, mainly due to the specific nature of their environment. Other organs, in contrast, are atrophied and eventually eliminated because they fall into disuse. For Lamarck, nature developed in the following way: circumstances create a need, that need creates habits, habits produce the changes resulting from the use or disuse of the organ and the means of nature are responsible for setting these modifications. Lamarck believed that these physiological changes acquired over the life of an organism could be transmitted to offspring. This hypothesis is known as the "*inheritance of acquired characteristics*" and is also commonly referred to as *Lamarckism*. We can say, therefore, that Lamarck was the first to formulate a strictly evolutionary hypothesis, although the word "evolution" was at that time reserved for the development of the embryo, so his proposal was referred to as "*transformism*". Despite Lamarck's hypothesis, there was no experimental evidence for the existence of mechanisms by which individuals could transmit the alleged improvements acquired over the course of their lives. In fact, Lamarckism is now considered an obsolete theory. The principles governing the transformation of the individual characters, which are now commonly accepted by science, were first established by Darwin and Wallace. The principles governing the transmission or inheritance of these characteristics were first established by Mendel.

In 1858, **Charles Robert Darwin** (1809-1882) and **Alfred Russel Wallace** (1823-1913) gave a presentation at the Linnean Society of London on a theory of the evolution of species by means of "*natural selection*" [Darwin and Wallace

(1858)]. One year later, Darwin published *On the Origin of Species* [Darwin (1859)], a work in which he provided a detailed explanation of his theory supported by numerous experiments and observations of nature. Darwin's theory - usually known as *Darwinism* - is based on a set of related ideas which try to explain different features of biological reality:

- Living beings are not static; they change continuously: some new organisms are created and some die out. This process of change is gradual, slow and continuous, with no abrupt or discontinuous steps.
- Species diversify, by adapting to different environments or ways of life, thus branching out. The implication of this phenomenon is that all species are somehow related - to varying degrees - and ultimately all species have a single origin in one common and remote ancestor. That is, all living organisms can be traced back to a single and common origin of life.
- Natural selection is the key to the system. It is conceived as a result of two factors: the inherited natural variability of the individuals of a species, and the selection through survival in the struggle for life, i.e., the fittest individuals, who were born with favorable spontaneous modifications better suited to the environment, and are thus more likely to survive, reproduce and leave offspring with these advantages. This implies that each slight variation, if useful, is preserved.

Although Darwin knew that there should be a mechanism for transmitting these characteristics from parents to offspring, he was unable to discover the transmission mechanism. It was **Gregor Johann Mendel** (1822 - 1884) who suggested a number of hypotheses which would set the basic underlying principles of heredity. Mendel's research [Mendel (1865)] focused on plants (peas) and, especially on individual features, all unequivocally different from each other and having the peculiarity of not being expressed in a graduated form, i.e., the feature is only present or not present. This research led Mendel to three important conclusions:

- The inheritance of each trait is determined by "units" or "factors" that are passed on to descendants unchanged. These units are now called "*genes*".
- An individual inherits one such unit from each parent for each trait.
- A trait may not show up in an individual but can still be passed on to the next generation. In this sense, the term "*phenotype*" refers to the set of physical or observable characteristics of an organism, while the term "*genotype*" refers to the individual's complete collection of genes. The difference between genotype and phenotype is that the genotype can be distinguished by looking at the deoxyribonucleic acid (DNA) of the cells and the phenotype can be established from observing the external appearance of an organism.

Mendel's laws are the foundation of modern genetics and abolished the idea

that characteristics are transmitted from parents to children through bodily fluids so that, when mixed, cannot be separated, thus causing the offspring to have characteristics that will mix the characteristics of the parents. This theory is called "*pangenesis*" and was mainly based on observations such as how crossing plants with red flowers with plants with white flowers produces plants with pink flowers. Starting from Mendel's advances in genetics and the concept of natural selection, in the thirties and forties the "*modern evolutionary synthesis*" was established [Fisher (1930); Wright (1931); Haldane (1932)]. Basically, this theory [Huxley (1942)] served as a link between the unity of evolution ("the gene") and the mechanism of evolution ("the selection"), i.e., gradual changes and natural selection in populations are the primary mechanism of evolution. According to this theory, genetic variation in populations arises mainly by chance through mutation (alteration or change in the genetic information - genotype - of a living being) and recombination (the mixing of chromosomes produced at meiosis).

Finally note that there are also theories that relate learning and evolution. This is the case of the Baldwin effect[Baldwin (1986)], which was proposed by **James Mark Baldwin**. This theory has also inspired some advances in EC.

1.2.2 *Main Concepts*

In the above description, some important definitions related to evolution and genetics were introduced. This section is devoted to describing some other terms that have been used in some popular books and papers on EC. A complete list of the terms contained in the papers on EC would be too extensive to be included. For this reason, we have selected the most broadly used terms:

- *DNA* (deoxyribonucleic acid): nucleic acid that consists of two long chains of nucleotides twisted into a double helix and joined by hydrogen bonds between the complementary bases adenine and thymine or cytosine and guanine. It is the main constituent of the chromosome and carries the genes as segments along its strands.
- *Chromosome*: structure within the nucleus of eukaryotic cells that bears the genetic material as a threadlike linear strand of DNA.
- *Gene*: as seen in the previous section, Mendel regarded "genes" as inherited factors which determine the external features on living beings. In modern genetics, a "gene" is defined as a sequence of DNA that occupies a specific location on a chromosome.
- *Diploid cell*: cell that contains two sets of paired chromosomes. For example, humans have 2 sets of 23 chromosomes, for a total of 46 chromosomes. Exact replicas of diploid cells are generated through a process denominated mitosis.
- *Haploid cell*: cell that contains only one complete set of chromosomes. The genesis of a haploid cell can occur by meiosis of diploid cells or by mitosis

of haploid cells. A haploid cell will merge with another haploid cell at fertilization, i.e., sperm and ova (also known as "gametes").
- *Locus*: specific position of the chromosome where a gene or other DNA sequence is located.
- *Genetic linkage*: tendency of genes that are located proximal to each other on a chromosome to be inherited together.
- *Alleles*: variant or alternative forms of a gene which are located at the same position - locus - on a chromosome.
- *Genotype*: genetic information of an organism. This information - contained in the chromosomes of the organism - may or may not be manifested or observed in the individual.
- *Phenotype*: property observed in the organism, such as morphology, development, or behavior. It is the expression of the genotype in relation to a particular environment.
- *Epistasis*: type of interaction between genes located at different loci on the same chromosome consisting of one gene masking or suppressing the expression of the other.

1.3 History of Evolutionary Computation

EC is a field of research with a fascinating but complex history. In its origins, several independent researchers proposed numerous approaches with the common feature of using natural evolution to inspire their implementations. Some of the developments were also inspired from other related fields such as artificial life [Conrad and Pattee (1970)] and other areas of artificial intelligence. EC had many independent beginnings, so different terms have been used to refer to concepts that are broadly similar. Given the similarities of the different schemes proposed, the term EC was invented in the early 1990s in an effort to unify the field of evolution-inspired algorithms [Fogel (1995)].

Determining the merits of the various authors is not easy, which is why several different versions of the history of EC have been told [Fogel (1998)]. In this section, we review the origins of EC, focusing on the most important milestones. Many of the bases of EC were developed in the 1960s and 1970s. During that period, three different popular schemes inspired by natural evolution were devised: Evolutionary Programming (EP), Evolution Strategies (ESs) and Genetic Algorithms (GAs). These schemes were inspired by the same ideas but some details, like the nature of the representation and the operators applied, were different. In addition, there were several earlier attempts that highly influenced the advances in this area. Some of these proposals can be considered to be the first designed Evolutionary Algorithms (EAs). In other cases, the schemes were substantially different from what is currently known as an EA. In any case, they are historically important because they had a significant impact on the subsequent studies carried out in the field. This

section reviews the origins along with the three aforementioned types of EAs.

1.3.1 *Origins*

Wright and Cannon [Wright (1932); Cannon (1932)] were probably the first researchers to influence the subsequent development of EC. They viewed natural evolution as a learning process where the genetic information of species is continuously changed through a trial and error mechanism. In this way, evolution can be regarded as a method whose aim is to maximize the adaptiveness of species to their environment. Wright went further and introduced the concept of fitness landscape as a representation of the space of all possible genotypes along with their fitness values. This concept is widely used today to study the performance of EAs [Richter (2014)]. In addition, he developed one of the first studies where selection and mutation were linked, concluding that a proper balance between them is required to proceed successfully. These ideas were extended by Campbell [Campbell (1960)]. He claimed that a blind-variation-and-selective-survival process is one of the main underlying principles of evolution, and hypothesized that "in all processes leading to expansions of knowledge, a blind-variation-and-selective-survival process is involved". This is one of the principles behind Universal Darwinism, a proposal that extends the applicability of Darwin's theory to other areas beyond natural evolution.

Turing can also be considered another EC pioneer. In [Turing (1950)], Turing recognized a connection between machine learning and evolution. Specifically, Turing claimed that in order to develop an intelligent computer that passes the famous Turing test, a learning scheme based on evolution might be used. In this regard, the structure of the evolved machine is related to hereditary material, changes made to the machine are related to mutation and the use of an experimenter to evaluate the generated machines are related to natural selection. In addition, Turing had previously suggested that these kinds of methods might be used to train a set of networks which were akin to current neural networks [Turing (1948)]. Turing used the term "genetical search" to refer to these kinds of schemes. However, this paper was not published until 1968 [Evans and Robertson (1968)] because his supervisor considered it to be a "schoolboy essay" [Burgin and Eberbach (2013)]. By the time of its publication, other EAs had already appeared and the term "genetical search" was never adopted by the EC community.

Some important advances were made in the late 1950s and early 1960s, coinciding with the period in which electronic digital computers became more readily available. In this period, several analyses of the population dynamics appearing in evolution were carried out. A survey of the application of digital computers in this field was presented in [Crosby (1967)]. Crosby identified three kinds of schemes:

- Methods based on studying the mathematical formulations that emerge in the deterministic analysis of population dynamics. These schemes usually

assume an infinite population size and analyze the distributions of genes under different circumstances. An example of this type of scheme is the one developed by Lewontin [Lewontin (1964)], where the interactions between selection and linkage are analyzed.

- Approaches that simulate evolution but do not explicitly represent a population [Crosby (1960)]. In these methods, the frequencies of the different potential genotypes are stored for each generation, resulting in a non-scalable scheme in terms of the number of genes. In addition, a faithful mathematical representation of the evolutionary system is required.
- Schemes that simulate evolution by explicitly maintaining a representation of the population. Fraser and some of his colleagues pioneered these kinds of methods and they analyzed this approach in a set of papers published over the course of a decade [Fraser (1957); Fraser and Burnell (1967)]. The main conclusions drawn from these studies were gathered in their seminal book [Fraser and Burnell (1970)]. From its inception, this model was widely accepted, and several researchers soon adopted it [Gill (1965); Young (1966)]. In fact, even though there are several implementation details that differ from those used in current EAs, several authors regard the works of Fraser as representing the first invention of a GA [Fogel (1995)]. As for the implementation details, the main difference with respect to GAs is that individuals are diploid instead of haploid. However, in our opinion, the key difference is not the implementation but the philosophy behind the use of the scheme. Fraser used its algorithms as a way to analyze population dynamics and not to solve problems, which is the typical current application of EAs.

There were other authors who proposed schemes resembling current EAs. Among them, the works of Friedman, Box and Friedberg are particularly important and often-cited papers. Friedman [Friedman (1956)] hypothesized on the automatic design of control circuits by using a scheme inspired by natural evolution that was termed "selective feedback". His proposals were very different from current ones. For instance, there was no notion of population and/or generations. Moreover, Friedman did not implement his method, and some of his assumptions seem to be overly optimistic [Fogel (1998)]. In any event, his research can be considered as the first efforts in the field of evolvable hardware.

Box proposed a technique called "Evolutionary Operation" (EVOP) [Box (1957)] to optimize the management processes of the chemical industry. His technique essentially used a single parent to generate multiple offspring by modifying a few production parameters at a time. Then, one individual was selected to survive to the next generation by considering certain statistical evidence. The system was not autonomous. Specifically, the variation and selection processes required human intervention, so it can be considered as the first interactive EA.

The most important contribution of Friedberg *et al.* to the field of EC was

their attempt to automatically generate computer programs using an evolutionary scheme [Friedberg (1958); Friedberg *et al.* (1959)]. In the previous papers, Friedberg *et al.* did not identify any relationship between their proposal and natural evolution. However, in subsequent publications by his coauthors, such a relationship was explicitly identified [Dunham *et al.* (1963)]. In their first attempts, the aim was to automatically generate small programs with some simple functionalities that relied on a tailor-made instruction set. In order to direct the search to promising regions, problem-dependent information was considered by defining a variation scheme specifically tailored to the problem at hand. Even with this addition, their success was limited. Even so, some of the analyses they carried out were particularly important to subsequent achievements in EC. Among their contributions, some of the most important are the following:

- The idea of dividing candidate solutions into classes is a precursor of the notions of *intrinsic parallelism* and *schema*, proposed years later by Holland [Holland (1975)].
- Several instruction sets were tested, showing, for the first time, the influence of genotype-to-phenotype mapping.
- A credit assignment algorithm was used to measure the influence of the single instructions. This idea is closely related to the work of Holland on GAs and classifier systems.

There are several more papers that did not elicit much attention when they were first published, but that proposed techniques very similar to others that were later reinvented. For instance, the work by Reed, Toombs and Barricelli [Reed *et al.* (1967)] provides several innovations closely related to self-adaptation, crossover and coevolution[1].

1.3.2 *Evolutionary Programming*

Evolutionary Programming (EP) was devised by Fogel [Fogel (1962)] while he was engaged in basic research on artificial intelligence for the National Science Foundation. At the time, most attempts to generate intelligent behavior used man as a model [Fogel (1999)]. However, Fogel realized that since evolution had been able to create humans and other intelligent creatures, a model mimicking evolution might be successful. Note that, as previously described, some research into this topic had already been published by then. The basic ideas adopted by Fogel are similar: use variation and selection to evolve candidate solutions better adapted to the given goals. However, Fogel was not aware of the existing research and his proposals differed substantially from them, meaning that his models can be considered as a reinvention of evolution-based schemes.

Philosophically, the coding structures utilized in EP are an abstraction of the

[1]Some preliminary work on coevolution was started in [Barricelli (1962)].

phenotype of different species [Fogel (1994)]. As a result, the encoding of candidate solutions can be freely adapted to better support the requirements of the problems at hand. Since there is no sexual communication between different species, this view of candidate solutions also justifies the lack of recombination operators in EP. Thus, the non-application of recombination operators is due to a conceptual rather than a technical view. The lack of recombination, the freedom to adapt the encoding and the use of a probabilistic survivor selection operator —not used in the initial versions of EP— might be considered the main features that distinguished EP from other EC paradigms [Fogel and Chellapilla (1998)].

In his first designs, Fogel reasoned that one of the main features that characterize intelligent behavior is the capacity to predict one's environment, coupled with a translation of said predictions into an adequate response so as to accomplish a given objective. A finite state transducer is a finite state machine (FSM) that allows a sequence of input symbols to be transformed into a sequence of output symbols. Depending on the states and given transitions, they can be used to model different situations and cause different transformations. Thus, Fogel viewed FSMs as a proper mechanism for dealing with the problem of generating intelligent behavior. In his initial experiments, the aim was to develop a FSM capable of predicting the behavior of an environment. The environment was considered to be a sequence of symbols belonging to a given input alphabet. The aim was to develop a FSM that, given the symbols previously generated by the environment, could predict the next symbol to emerge from the environment.

The initial EP proposal operates as follows. First, a population with N random FSMs is created. Then, each member of the population is mutated, creating as many offspring as parents. Five mutation modes are considered: add a state, delete a state, change the next state of a transition, change the output symbol of a transition, or change the initial state. Mutation operators are chosen randomly —other ways were also tested—, and in some cases the offspring are subjected to more than one mutation. Finally, the offspring are evaluated and the best N FSMs are selected to survive. FSMs are evaluated in light of their capacity to correctly predict the next symbols in known sequences. Initially, the fitness is calculated by considering a small number of symbols, but as the evolution progresses more symbols are attached to the training set.

In the first experiments carried out involving EP, different prediction tasks were tested: periodic sequences of numbers, sequences with noise, non-stationary environments, etc. Subsequently, more difficult tasks were considered. Among other applications, EP was used to tackle pattern recognition, classification and control system design. All this research resulted in the publication of the first book on EC [Fogel *et al.* (1966)]. EP was also used to evolve strategies for gaming [Fogel and Burgin (1969)]. This work is particularly important because it is one of the first applications of coevolution.

It is also important to remark that in most of the initial studies on EP, the

amount of computational results was not ample because of the limited computational power at the time of its inception. Most of these initial experiments were recapitulated and extended in a later period [Fogel and Fogel (1986)], providing a much deeper understanding of EP.

In the 1970s, most of the research into EP was conducted under the guidance of Dearholt. One of the main contributions of his work was the application of EP to practical problems. EP was applied to pattern recognition in regular expressions [Lyle (1972)] and handwritten characters [Cornett (1972)] and to classify different types of electrocardiograms [Dearholt (1976)]. These works incorporated several algorithmic novelties. Among them, the most influential were the use of several simultaneous mutation operators and the dynamic adaptation of the probabilities associated with the different mutation schemes.

Starting in the 1980s, EP diversified by using other arbitrary representations of candidate solutions in order to address different problems. The number of applications that have been addressed with EP is huge. In [Fogel and Fogel (1986)], EP was used to solve routing problems by considering a permutation-based encoding. In order to tackle the generation of computer programs, tree-based encoding was used in [Chellapilla (1997)]. Real-valued vectors were used to deal with continuous optimization problems in [Yao *et al.* (1999)] and to train neural networks in [Porto and Fogel (1995)]. During this period, the feeling was that by designing problem-specific representations with operators specifically tailored to face a given problem, more efficient searches might be performed [Fogel (1999)]. In fact, most EP practitioners defended that by designing intelligent mutation schemes, the use of recombination operators might be avoided [Fogel and Atmar (1990); Chellapilla (1997)].

Among the aforementioned topics, the application of EP to continuous optimization problems saw a large expansion in the 1990s. Over this period, great efforts were made into developing self-adaptive schemes. In self-adaptation, some of the parameters required by the algorithm are bound to each individual and evolved with the original variables. The variables of the problem are called *object parameters*, while those newly attached are called *strategy parameters*. Initial proposals adapted the scale factor of Gaussian mutation [Fogel *et al.* (1991)]. In more advanced variants of EP, several mutation operators are considered simultaneously [Yao *et al.* (1999)]. Since self-adaptive schemes surpassed more traditional EP variants, self-adaptation was adopted by practitioners addressing problems where real encoding was not used, becoming a common mechanism in EP [Fogel *et al.* (1995)]. Self-adaptive EP for continuous optimization presents several similarities with ESs. One of the most important differences is that EP does not include recombination. In addition, some implementation details were also different in the first variants devised [Eiben and Smith (2003)]. For instance, the order in which the object and strategy parameters were subjected to mutation was different in the two schemes, though these differences vanished over time. Moreover, it has been shown that there are cases where mutation alone can outperform schemes with recombination

and vice versa [Richter *et al.* (2008)]. There is also evidence that indicates that it is promising to adapt the parameters associated with crossover [Jain and Fogel (2000)], which further stretches the barriers between EP and ESs. Since self-adaptation for continuous optimization was first proposed in ESs, the history of and advances in self-adaption for continuous optimization is presented in the section devoted to ESs.

1.3.3 *Evolution Strategies*

In the mid-1960s, three students at the Technical University of Berlin —Bienert, Schwefel and Rechenberg— were studying practical problems that arise in fluid mechanics and some related fields. Their desire was to build robots that could autonomously solve engineering problems [Fogel (1998)]. They formulated these engineering tasks as optimization problems and developed autonomous machines that were automatically modified using certain rules. They also made some initial attempts at using traditional optimization schemes. However, since the problems were noisy and multimodal, their schemes failed to provide promising solutions. In order to avoid these drawbacks, they decided to apply mutation and selection methods analogous to natural evolution. Rechenberg published the first report on ESs by applying these ideas to minimizing the total drag of a body in a wind tunnel [Rechenberg (1965)]. Subsequently, other problems like the design of pipes and efficient flashing nozzles were addressed [Fogel (1998)].

Philosophically, the coding structures utilized in ESs are an abstraction of the phenotype of different individuals [Fogel (1994)]. This is why the encoding of candidate solutions can be freely adapted to better support the requirements of the problems at hand. In addition, recombination among individuals is allowed in spite of not being implemented in the initial variants of ESs.

In the first problems tested, a set of quantitative discrete variables had to be adjusted. The initial proposal was very similar to the current (1+1)-ES and works as follows. First, a random solution is created and considered to be the current solution. Then, the current solution is mutated by slightly changing each variable. Specifically, mutation is implemented by emulating a Galton pin-board, so binomial distributions are used. The principle behind the design of this operator is to apply small mutations more frequently than large mutations, as is the case in nature. The current solution is replaced by the mutant only if the mutant solution is at least as good. This process is repeated until the stopping criterion is reached. Empirical evidence revealed the promising behavior of these kinds of schemes with respect to more traditional optimization methods.

In 1965, Schwefel first implemented an ES in a general computer [Schwefel (1965)]. In this case, ESs were mainly applied to continuous optimization. The most important contribution of the new variant was the incorporation of Gaussian mutation with zero mean, which is still the most typical distribution used nowadays. The mutation operator could be controlled by tuning the standard deviations —or

step size— of the Gaussian distribution. Studies on the step size resulted in the first theoretical analyses of ESs.

As previously discussed, initial ES variants only kept a single solution at a time. However, in the late 1960s, the use of populations with several individuals was introduced. The first population-based scheme tested is now known as (μ+1)-ES. This scheme uses a population with μ individuals. These individuals are used to create a new individual through recombination and mutation. Then, the best μ individuals from the $\mu + 1$ individuals are selected to survive. This scheme resembles the steady state selection that was popularized much later in the field of GAs [Whitley and Kauth (1988)].

ESs were extended further to support any number of parents and offspring. In addition, two different selection schemes and a notation that is still in use were proposed. The first selection comprises the (μ+λ)-ES. In this case, λ offspring are created from a population with μ individuals. Then, the best μ individuals from the union of parents and offspring are selected to survive. Schemes that consider the second kind of selection are known as (μ,λ)-ES. In this case, the best μ individuals from the λ offspring are selected to survive.

When ESs were first developed, the papers were written in German, meaning they were accessible to a very small community. However, in 1981 Schwefel published the first book on ESs in English [Schwefel (1981)]. Since then, several researchers have adopted ESs and the number of studies in this area has grown enormously. Schwefel's book focuses on the use of ESs for continuous optimization, which is in fact the field where ESs have been more successful. In his book, Schwefel discusses, among other topics, the use of different kinds of recombinations and the adaptation of the Gaussian distributions adopted for mutation. Regarding recombination, some controversies and misunderstandings have appeared. The use of adaptive mutation is one of the distinguishing features of ESs, and the history of and advances in both topics are described herein. Most of the papers reviewed in this chapter involve continuous single-objective optimization problems. However, it is important to note that the application of ESs has not been limited to these kinds of problems. For instance, multi-objective [Igel *et al.* (2007)], mixed-integer [Li *et al.* (2013)] and constrained problems [Mezura-Montes and Coello Coello (2005)] have also been addressed.

1.3.3.1 *Recombination*

Initially, four different recombination schemes were proposed by combining two different properties [Schwefel (1981)]. First, two choices were given for the number of parents taking part in creating an offspring: *bisexual* or *multisexual*. In bisexual recombination —also known as local— two parents are selected and they are used to recombine each parameter value. In the case of multisexual recombination — also known as global—, different pairs of parents are selected for each parameter value. Thus, more than two individuals can potentially participate in the creation

of each offspring. In addition, two different ways of combining two parameter values were proposed. In the *discrete* case, one of the values is selected randomly. In the *intermediary* case, the mean value of both parameters is calculated.

In the above recombination schemes, the number of parents taking part in each recombination cannot be specified. Rechenberg proposed a generalization that changed this restriction [Rechenberg (1978)]. Specifically, a new parameter, ρ, is used to specify the number of parents taking part in the creation of each individual. The new schemes were called $(\mu/\rho,\lambda)$ and $(\mu/\rho+\lambda)$. Further extensions of the intermediary schemes were developed by weighting the contributions of the different individuals taking part in the recombination [Bäck and Schwefel (1993)]. Different ways of assigning weights have been extensively tested. Previous schemes were not proper generalizations of the initial schemes, in the sense that the intermediary case calculates the mean of ρ individuals, and not the mean of two individuals selected from among a larger set, as was the case in the initial schemes. A more powerful generalization that allowed for both the original operators and those proposed by Rechengerg to be implemented was devised [Eiben and Bäck (1997)]. Thus, using the notation proposed in this last paper is preferable.

The situation became even more complicated because of a misinterpretation of the initial crossover schemes. In the survey presented in [Bäck *et al.* (1991)], the authors showed a formula indicating that in global recombination one parent is chosen and held fixed while the other parent is randomly chosen anew for each component.[2] This new way of recombination was adopted by many authors over the following years, so when using global recombination, the exact definition must be carefully given.

1.3.3.2 *Adaptive Mutation*

One of the main features that characterizes ESs is its mutation operator. In the case of continuous optimization, mutation is usually based on Gaussian distributions with zero mean. In most cases, individuals are disturbed by adding a random vector generated from a multivariate Gaussian distribution, i.e., the candidate solutions x_i are perturbed using equation (1.1). In this equation, C represents a covariance matrix.

$$x_i' = x_i + N_i(0, C) \tag{1.1}$$

From the inception of ESs, it was clear that the features of C might have a significant effect on search performance. As a result, several methods that adapt C during the run have been developed. Many ES variants can be categorized into one of the following groups depending on the how C is adapted [Hansen and Ostermeier (2001)]:

[2]To the best of our knowledge, [Bäck *et al.* (1991)] was the first paper where this definition was given.

- Schemes where the surfaces of equal probability density are hyper-spheres. In these cases, the only free adaptive parameter is the global step size or standard deviation.
- Schemes where the surfaces of equal probability density are axis-parallel hyper-ellipsoids. In these cases, the most typical approach is to have as many free adaptive parameters as there are dimensions.
- Schemes where the surfaces of equal probability density are arbitrarily oriented hyper-ellipsoids. In these cases, any positive-definite matrix can be used, resulting in $(n^2 + n)/2$ free adaptive parameters.

The first adaptive ES considered the global step size as the only free parameter. These schemes originated from the studies presented in [Rechenberg (1973)], which analyzed the convergence properties of ESs depending on the relative frequency of successful mutations. These analyses led to the first adaptive scheme [Schwefel (1981)], where the global step size is adjusted by an online procedure with the aim of producing successful mutations with a ratio equal to 1/5. This rule is generally referred to as the "$\frac{1}{5}$ success rule of Rechenberg". However, this rule is not general so it does not work properly for many functions, which is why self-adaptive ESs were proposed. In the first of such variants, the only strategy parameter was the global step size, and this strategy parameter was subjected to variation using a lognormal distributed mutation.

In the case of schemes where the surfaces of equal probability are axis-parallel hyperellipsoids, a larger number of methods have been devised. A direct extension of previous methods considers the self-adaptation of n strategy parameters, where n is the number of dimensions of the optimization problem at hand. In such a case, each strategy parameter represents the variance in each dimension. Another popular ES variant is the *derandomized* scheme [Ostermeier *et al.* (1994a)]. Ostermeier *et al.* realized that in the original self-adaptive ES, the interaction of the random elements can lead to a drop in performance. The scheme is based on reducing the number of random decisions made by ESs. Another interesting extension is termed *accumulation* [Ostermeier *et al.* (1994b)]. In these variants, the adaptation is performed considering information extracted from the best mutations carried out during the whole run, and not only in the last generation. Note that these last variants favor adaptation instead of self-adaptation.

By adapting the whole covariance matrix, correlations can be induced among the mutations performed involving the different parameters, thus making ESs more suitable for dealing with non-separable functions. The first adaptive scheme where the whole matrix is adapted was presented in [Schwefel (1981)]. In this case, the coordinate system in which the step size control takes place, as well as the step sizes themselves, are self-adapted. In general, this method has not been very successful because it requires very large populations to operate properly. A very efficient and popular scheme is covariance matrix adaptation (CMA-ES) [Hansen and Oster-

meier (2001)]. In this scheme, derandomization and accumulation are integrated in an adaptive scheme whose aim is to build covariance matrices that maximize the creation of mutation vectors that were successful in previous generations of the execution. This scheme is very complex and the practitioner must specify a large number of parameters. This has led to simpler schemes being devised. For instance, the covariance matrix self-adaptation scheme (CMSA) [Beyer and Sendhoff (2008)] reduces the number of parameters required by combining adaptation and self-adaptation. Many other not so popular schemes that are capable of adapting the whole covariance matrix have been devised. The reader is referred to [Rudolph (2012)] for a broader review of these kinds of schemes.

Finally, it is important to note that several schemes that favor some directions without adapting a full covariance matrix have been devised. For instance, in [Poland and Zell (2001)], the main descent direction is adapted. The main advantage of these kinds of schemes is the reduced complexity in terms of time and space. Since the time required for matrix computation is usually negligible in comparison to the function evaluation phase, these schemes have not been very popular. Still, they might be useful especially for large-scale problems, where the cost of computing the matrices is much higher. In addition, several other adaptive and self-adaptive variants that do not fit into the previous categorization have been devised. For instance, some schemes that allow the use of Gaussian distributions with a non-zero mean have been proposed [Ostermeier (1992)]. In addition, some methods not based on Gaussian mutation have been designed [Yao and Liu (1997)]. In this chapter, we have briefly described the fundamentals of different methods. The reader is referred to [Rudolph (2012); Bäck *et al.* (2013)] for a discussion of the implementation details for several ES variants.

1.3.4 *Genetic Algorithms*

Holland identified the relationship between the adaptation process that appears in natural evolution and optimization [Holland (1975)], and conjectured that it might have critical roles in several other fields, such as learning, control and/or mathematics. He proposed an algorithmic framework inspired by these ideas, which was initially called the "Genetic Plan" and was subsequently renamed the "Genetic Algorithm". GAs are based on progressively modifying a set of structures with the aim of adapting them to a given environment. The specific way of making these modifications is inspired by genetics and evolution. His initial aim was to formally study the phenomenon of adaptation. However, GAs were soon used as problem solvers. Specifically, GAs have been successfully applied to numerous applications, such as machine learning, control, search and function optimization. A very popular use of GAs appeared in the development of *learning classifier systems*, which is an approach to machine learning based on the autonomous development of rule-based systems capable of generalization and inductive learning. The application of GAs

as function optimizers was also soon analyzed [De Jong (1975)], defining metrics, benchmark problems and methodologies to perform comparisons. This is probably the field where GAs have been most extensively and successfully applied.

The evolutionary schemes devised by Holland are very similar to those previously designed by Bremermann [Bremermann (1962)] and Bledsoe [Bledsoe (1961)]. In fact, "by 1962 there was nothing in Bremermann's algorithm that would distinguish it from what later became known as genetic algorithms" [Fogel and Anderson (2000)]. Moreover, they pioneered several ideas that were developed much later, like the use of multisexual recombination, the use of real-encoding evolutionary approaches and the adaptation of mutation rates. Moreover, since Bremermann's algorithm was very similar to the schemes described earlier by Fraser, some authors regard GAs as having been reinvented at least three times [Fogel (1995)].

Philosophically, the coding structures utilized in GAs are an abstraction of the genotype of different individuals [Fogel (1994)]. Specifically, each trial solution is usually coded as a vector, termed *chromosome*, and each element is denoted with the term *gene*. A candidate solution is created by assigning values to the genes. The set of values that can be assigned are termed *alleles*. In the first variants of GAs, there was an emphasis on using binary representations —although using other encodings is plausible—, so even when continuous optimization problems were addressed, the preferred encoding was binary. Goldberg claimed that "GAs work with a coding of the parameter set, not the parameters themselves" [Goldberg (1989)], which is a distinguishing feature of GAs. This results in the requirement to define a mapping between genotype and phenotype.

The working principles of GAs are somewhat similar to those of EP and ESs. However, unlike other schemes, in most early forms of GAs, recombination was emphasized over mutation. The basic operation of one of the first GAs —denoted as canonical GA or simple GA— is as follows. First, a set of random candidate solutions is created. Then, the performance of each individual is evaluated and a *fitness value* is assigned to each individual. This fitness value is a measure of each individual's quality, and in the initial GA variants it had to be a positive value. Based on the fitness values, a temporary population is created by applying *selection*. This selection process was also called *reproduction* in some of the first works on GAs [Goldberg (1989)]. The best-performing individuals are selected with larger probabilities and might be included several times in the temporary population. Finally, in order to create the population of the next generation, a set of variation operators is applied. The most popular operators are crossover and mutation, but other operators such as inversion, segregation, and translocation have also been proposed [Goldberg (1989)]. In most of the early research on GAs, more attention was paid to the crossover operator. The reason is that crossover was believed to be the major source of the power behind GAs, while the role of mutation was to prevent the loss of diversity [Mitchell (1998)]. This first variant was soon extended by Holland's students. Some of these extensions include the development of game-playing strategies, the use of diploid

representations and the use of adaptive parameters [Bäck *et al.* (1997)].

Among Holland's many contributions, the schema theorem was one of the most important. Holland developed the concepts of schemata and hyperplanes to explain the working principles of GAs. A schema is basically a template that allows grouping candidate solutions. In a problem where candidate solutions are represented with l genes, a schema consists of l symbols. Each symbol can be an allele of the corresponding gene or "*", which is the "don't care" symbol. Each schema designates the subset of candidate solutions in which the corresponding representations match every position in the schema that is different from "*". For instance, the candidate solutions "0 0 1 1" and "0 1 0 1" belong to the schema "0 * * 1". Note that the candidate solutions belonging to a schema form a hyperplane in the space of solutions. Holland introduced the concept of *intrinsic parallelism* — subsequently renamed as *implicit parallelism* —, to describe the fact that the evaluation of a candidate solution provides valuable information on many different schemata. He hypothesized that GAs operate by better sampling schemata with larger mean fitness values.

The concept of schema was also used to justify the use of binary encoding. Specifically, Holland showed that by using binary encoding the number of schemata sampled in each evaluation can be maximized. However, the use of binary encoding was not extensively tested in his preliminary works, and its application has been a source of controversy. One of the first problems detected with the use of binary encoding is that, depending on the length of the vector, precision or efficiency might be sacrificed. It is not always easy to strike a proper balance, so dynamic parameter encoding methods have also been proposed [Schraudolph and Belew (1992)]. Goldberg weakened the binary-encoding requirement by claiming that "the user should select the smallest alphabet that permits a natural expression of the problem" [Goldberg (1989)]. A strict interpretation of this means that the requirement for binary alphabets can be dropped [Bäck *et al.* (1997)]. In fact, several authors have been able to obtain better results by using representations different from the binary encoding, such as floating point representations. For instance, Michalewicz carried out an extensive comparison between binary and floating point representations [Michalewicz (1994)]. He concluded that GAs operating with floating-point representations are faster, more consistent and provide a higher precision. Subsequently, Fogel and Ghozeil demonstrated that there are equivalences between any bijective representation [Fogel and Ghozeil (1997)]. Thus, for GAs with bijective representations, the interactions that take place between the representation and other components are one of the keys to their success.

Holland used the above concepts to develop the *schema theorem*, which provides lower bounds to the change in the sampling rate for a single hyperplane from one generation to the next. For many years, the schema theorem was used to explain the working operation of GAs. However, several researchers have pointed out some weaknesses that call into question some of the implications of the theorem [Altenberg (1995)]. Some of the most well-known weaknesses pointed out by several researchers

are the following. First, the schema theorem only provides lower bounds for the next generation and not for the whole run. Second, the schema theorem does not fully explain the behavior of GAs in problems that present large degrees of inconsistencies in terms of the bit values preferred [Heckendorn *et al.* (1996)]. Finally, considering the typical population sizes used by most practitioners, the number of samples belonging to high-order schemata — those with few "*"— is very low, so in these cases the theorem is not very accurate. In general, the controversies are not with the formulae, but with the implications that can be derived from them. The reader is referred to [Whitley and Sutton (2012)] for a more detailed discussion of this topic. Finally, we should note that other theoretical studies not based on the concept of schema have also been proposed to explain the behavior of GAs. In some cases, the model is simplified by assuming infinitely large population sizes [Vose and Liepins (1991)]. In other cases, the Markov model has been used to analyze simple GAs with finite population sizes [Nix and Vose (1992)].

Due to the complexity of analyzing the mathematics behind GAs, there is a large amount of work focused on the practical use of GAs. For instance, the initial GA framework has been extended by considering several selection and variation operators. Regarding the former, note that initially, selection was only used to choose the individuals that were subjected to variation. This operator is now known as *parent selection* or *mating selection* [Eiben and Smith (2003)]. However, in most current GAs, a *survivor selection* or *environmental selection* is also carried out. The aim of this last selection —also called *replacement*— is to choose which individuals survive to the next generation. In its most general form, survivor selection can operate on the union of parents and offspring. The number of selection operators defined in recent decades is very large, ranging from deterministic to stochastic schemes. Parent selection is usually carried out with stochastic operators. Some of the most well-known stochastic selectors are: *fitness proportional selection*, *ranking selection* and *tournament selection*. Alternatively, survivor selection is usually based on deterministic schemes. In the case of the survivor selection schemes, not only is the fitness used, but the age of individuals might also be considered. Some of the most-well known schemes are *age-based replacement* and *replace-worst*. Also of note is the fact that *elitism* is included in most current GAs. Elitist selectors ensure that the best individual from among parents and offspring is always selected to survive. In these kinds of schemes, and under some assumptions, asymptotic convergence is ensured [Eiben *et al.* (1991)] and, in general, there is empirical evidence of its advantages. A topic closely related to replacement is the *steady state model* [Whitley and Kauth (1988)]. In the steady state model, after the creation of one or maybe a few offspring, the replacement phase is executed. Note that this model is closely related to the *generation gap* concept previously defined in [De Jong (1975)]. The generation gap was introduced into GAs to permit overlapping populations.

In the same way, several crossover schemes have been proposed in the literature. The first GAs proposed by Holland operated with one-point crossover, whose oper-

ation is justified in light of the schemata theorem. Specifically, Goldberg [Goldberg (1989)] claimed that in GAs with one-point crossover, short, low-order and highly fit schemata —which received the name of building blocks[3]— are sampled, recombined and resampled to form strings of potentially higher fitness. However, this hypothesis has been very controversial. One of the basic features of one-point crossover is that bits from a single parent that are close together in the encoding are inherited together, i.e., it suffers from *positional bias*. This idea was borrowed in part from the biological concept of *coadapted alleles*. However, several authors have questioned the importance of this property [Whitley and Sutton (2012)]. In fact, Goldberg claimed that since many encoding decisions are arbitrary, it is not clear that a GA operating in this way might obtain the desired improvements [Goldberg (1989)]. The inversion operator might in some way alleviate this problem by allowing the reordering of genes so that linkages between arbitrary positions can be created. Another popular attempt to identify linkages was carried out in the messy GA [Goldberg *et al.* (1989)], which explicitly manipulates schemata and allows linking non-adjacent positions. More advanced linkage learning mechanisms have been depicted recently [Chen and Lim (2008)]. In addition, given the lack of a theory that fully justifies the use of one-point crossover, several different crossover operators, as well as alternative hypotheses to explain the effectiveness of crossover, have been proposed. Among these hypotheses, some of the widely accepted consider crossover as a macro-mutation or as an adaptive mutation [Sivanandam and Deepa (2007)]. Regarding alternative definitions of crossover operators, it has been shown empirically that depending on the problem, operators different from one-point crossover might be preferred. For instance, Syswerda showed the proper behavior of uniform crossover with a set of benchmark problems [Syswerda (1989)]. In the case of binary encoding, some of the most well-known crossover operators are the two-point crossover, multi-point crossover, segmented crossover and uniform crossover [Eiben and Smith (2003)]. Note that some multisexual crossover operators have also been provided. In fact, well before the popularization of GAs, Bremermann had proposed their use [Fogel (1998)]. For other representations, a large set of crossover operators have been devised [Eiben and Smith (2003)].

As mentioned earlier, several components and/or parameters in GAs have to be tuned. The first GA variants lacked procedures to automatically adapt these parameters and components. However, several adaptive GAs have also been proposed [Lobo *et al.* (2007)]. In these schemes, some of the parameters and/or components are adapted by using the feedback obtained during the search. In addition, most current GAs do not operate on binary strings, relying instead on encodings that fit naturally with the problem at hand. As a result, several variation operators specific to different chromosome representations have been defined. The reader is referred to EAs [Eiben and Smith (2003)] for a review of these operators.

[3]The term building block has also been used with slightly different definitions [Radcliffe (1997)].

1.4 A Unified View of Evolutionary Computation

The three main branches of EC —(EP, GAs, and ESs)— developed quite independently of each other over the course of about 25 years, during which several conferences and workshops devoted to specific types of EAs were held, such as the Evolutionary Programming Conference. However, in the early 1990s, it was clear to some researchers that these approaches had several similarities, and that findings in one kind of approach might also be useful for the other types of EAs. Since these researchers considered that the topics covered by the conferences at the time period were too narrow, they decided to organize a workshop called "Parallel Problem Solving From Nature" that would accept papers on any type of EA, as well as papers based on other metaphors of nature. This resulted in a growth in the number of interactions and collaborations among practitioners of the various EC paradigms, and as a result the different schemes began to merge naturally. This unification process continued at the *Fourth International Conference on Genetic Algorithms*, where the creators of EP, GAs and ESs met. At this conference, the terms EC and EA were proposed and it was decided to use these terms as the common denominators of their approaches. Basically, an EA was defined as any approach towards solving problems that mimics evolutionary principles.[4] Similar efforts were undertaken at other conferences, such as in the *Evolutionary Programming Conference*. Finally, these unifying efforts resulted in the establishment in 1993 of the journal *Evolutionary Computation*, published by MIT Press. Furthermore, the original conferences were soon replaced by more general ones, like the *Genetic and Evolutionary Computation Conference*[5] and the *IEEE Congress on Evolutionary Computation*. In addition, other journals specifically devoted to EC have appeared, like the *IEEE Transactions on Evolutionary Computation*.

It is worth noting that since the different types of EAs were developed in an independent way, many ideas have been reinvented and explored more than once. In addition, there was no clear definition for each kind of scheme, so the boundaries between EC paradigms become blurred. Thus, for many contemporary EAs it is very difficult and even unfair to claim that they belong to one or another type of EA. For instance, a scheme that adopts proportional parent selection, self-adaptation and stochastic replacement merges ideas that were originally depicted in each of the initial types of EAs. For this reason, when combining ideas that were originated by practitioners of the different EC paradigms — which is very typical —, it is preferable to use the generic terms EC and EA.

In recent decades, several papers comparing the differences and similarities of the different types of EAs have appeared [Bäck *et al.* (1993)]. In addition, some authors have proposed unifying and general frameworks that allow for the implementation

[4]See `http://ls11-www.cs.uni-dortmund.de/rudolph/ppsn`

[5]The conference originally known as "International Conference on Genetic Algorithms" was renamed "Genetic and Evolutionary Computation Conference".

Algorithm 1.1 Pseudocode of an Evolutionary Algorithm - A Unified View

1: **Initialization**: Generate an initial population with N individuals
2: **Evaluation**: Evaluate every individual in the population
3: **while** (not fulfilling the stopping criterion) **do**
4: **Mating selection**: select parents to generate the offspring
5: **Variation**: Apply variation operators to the mating pool to create a child population
6: **Evaluation**: Evaluate the child population
7: **Survivor selection**: Select individuals for the next generation
8: **end while**

of any of these schemes [Bäck (1996); De Jong (2006)]. In fact, note that with a pseudocode as simple as the one shown in Algorithm 1.1, any of the original schemes can be implemented. Although this simple pseudocode does not fit with every contemporary EA, it does capture the main essence of EC by combining population, random variation and selection. Two interesting books where the unifying view is used are: [Eiben and Smith (2003); De Jong (2006)].

The last two decades have seen an impressive growth in the number of EC practitioners. Thus, the amount of research that has been conducted in the area is huge. For instance, EAs have been applied to several kinds of optimization problems, such as constrained [Mezura-Montes (2009)] and multi-objective [Coello Coello *et al.* (2007)] optimization problems. Also, several efforts have been made to address the problem of premature convergence [Črepinšek *et al.* (2013)]. Studies conducted on this topic include the use of specific selection schemes such as fitness sharing [Goldberg and Richardson (1987)], crowding schemes [De Jong (1975)], restarting mechanisms [Eshelman (1991)] and the application of multi-objective concepts to tackle single-objective problems [Segura *et al.* (2016)]. Other highly active topics include the design of parallel EAs [Alba (2005)] and memetic [Moscato (1989)] algorithms—related to Lamarck's hypothesis and Baldwin effect which were discussed before—which allows hybridizing EAs with more traditional optimization schemes. Note that for most of the topics that have been studied over these past decades, there were some preliminary works that were developed in the 1960s. For instance, the island-based model — which is one of the most popular parallel models — was proposed in [Bossert (1967)], while some preliminary works on hybrid models were presented as early as 1967 [Kaufman (1967)]. The aforementioned works represent only a very small cross-section of the topics that have been covered in recent years. It is beyond the scope of this chapter to present an extensive review of current research, so readers are referred to some of the latest papers published in some of the most popular conferences and journals in the field.

Interestingly, it is also remarkable that while considerable efforts have been made to unify the different EC paradigms, some new terms for referring to specific classes of EAs have also appeared in recent decades. However, in these last cases, their

distinguishing features are clear, so there is no ambiguity in the use of such terms. For instance, Differential Evolution [Storn and Price (1995)] is a special type of EA where the mutation is guided by the differences appearing in the current population. Another popular variant is Genetic Programming [Koza (1990)], which focuses on the evolution of computer programs.

Finally, we would like to state that while there has been considerable research into EC in the last decade, the number of topics that have yet to be addressed and further explored in the field is huge. Moreover, in recent years there has been a remarkable increase in the number of proposals based on alternative nature-inspired phenomena. In some cases, they have been merged with evolutionary schemes. Thus, similarly to the period where synergies were obtained by combining ideas that arose within different evolutionary paradigms, the interactions among practitioners of different nature-inspired algorithms might be beneficial for advancing this field.

1.5 Design of Evolutionary Algorithms

EC is a general framework that can be applied to a large number of practical applications. Among them, the use of EC to tackle optimization problems is probably the most popular one [Eiben and Smith (2003)]. The generality of EC and the large number of different components that have been devised imply that when facing new problems, several design decisions must be made. These design decisions have an important impact on the overall performance [Rothlauf (2011)], meaning they must be made very carefully. Thus, while EAs are usually referred to as general solvers, most successful EAs do not treat problems as black-box functions. Instead, information on the problem is used to alter the definition of the different components of the EA [Grefenstette (1987)]. This section discusses some of the main decisions involved in the design of EAs and describes some of the most well-known components that have been used to handle optimization problems.

One of the first decisions to make when applying EAs is the way in which individuals are represented. The representation of a solution involves selecting a data structure to encode solutions [Ashlock *et al.* (2012)] and, probably more importantly, a way to transform this encoding (genotype) into the phenotype [Rothlauf (2006)]. Note that in direct representations, there is no transformation between genotype and phenotype, whereas in cases where such a transformation is required, the representation is said to be indirect. Note that while in many cases a direct representation is effective, indirect representations allow for the introduction of problem-specific knowledge, the use of standard genetic operators and, in some cases, it facilitates the treatment of constraints, among other benefits. Several efforts to facilitate the process of designing, selecting and comparing representations have been developed. The recommendations given by Goldberg (1990), Radcliffe (1992), Palmer (1994) and Ronald (1997) are interesting but too general, and in some cases there is a lack of theory behind the recommendations. A more formal framework is given by

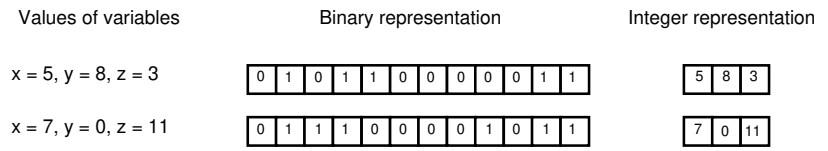

Fig. 1.2 Individuals with binary and integer encoding

Rothlauf [Rothlauf (2006)] that introduces some important features that should be taken into account, such as redundancy, scaling and locality. The aim of this last framework is to reduce the black art behind the selection of proper representations. However, since analyses and recommendations are based on simplifications of EAs, there is usually a need to resort to trial and error mechanisms, which are combined with the analyses of these recommendations and features. Finally, it is important to note that the representation chosen influences other design decisions. For instance, variation operators are applied to the genotype, so its design depends on the representation.

In order to illustrate the encoding process, let us consider a simple optimization problem such as the minimization of the function $f(x, y, z)$ (Equation 1.2), where each variable is an integer number in the range $[0, 15]$.

$$f(x, y, z) = (x - 1)^2 + (y - 2)^2 + (z - 7)^2 - 2 \tag{1.2}$$

Considering some of the first GAs designed, an alternative is to represent each number in base-2 with 4 binary digits. In such a case, individuals consist of a sequence of 12 binary digits or genes (based on the GAs nomenclature). Another possible choice — which is more widespread nowadays — is to use a gene for each variable, where each gene can have any value between 0 and 15. Figure 1.2 shows some candidate solutions considering both the binary and integer representations. Each cell represents a gene of the chromosome. Note that for this case, the base-2 encoding is an example of an indirect representation, whereas the second encoding is a direct representation. Another example of an indirect representation appeared at the *Second Edition of the Wind Farm Layout Optimization Competition* [Wilson *et al.* (2018)]. The purpose of this contest was to design optimizers to select the positions of turbines. Several of the top-ranked contestants noted that most high-quality solutions aligned several of the turbines in specific directions. Thus, instead of directly optimizing the positions, having problem-dependent information —relationship among directions and positions— can be used to design an indirect encoding, which focuses the search on promising regions of the search space. Consequently, instead of looking for thousands of positions, just a few parameters had to be optimized.

Another important component of EAs is the fitness function. The role of the fitness function is to assign a quality measure to each individual. Note that some

problems might involve several requirements and objectives. In order to deal with these multiple aims, they might be combined into the fitness function to generate a global notion of quality. A taxonomy regarding typical kinds of requirements and methods to combine them are discussed in [Wilkerson and Tauritz (2011)]. In simple EAs, the fitness function is used mainly in the selection stages. However, in more complex algorithms, such as memetic algorithms, other components also take the fitness function into account. The term *fitness function* is usually associated with maximization [Eiben and Smith (2003)], but it has caused some controversy because in some papers minimization fitness functions are employed. Changing minimization into maximization or vice versa in the implementation of EAs is in many cases trivial, but when connecting selection operators and fitness functions, both components should agree in the optimization direction. Moreover, as discussed later, some selection operators only make sense when the fitness function is maximized and every candidate solution attains a positive fitness. Thus, when designing the fitness function, the way in which it will be used must be taken into account.

In order to illustrate the definition of a fitness function, let us consider an even simpler optimization problem, such as the minimization of the function $f(x)$ (Equation 1.3), where x is a real number in the range $[0, 15]$. In this case, a plausible fitness function for the minimization of $f(x)$ that guarantees that the fitness value of any candidate solution is positive is given by f_1 (Equation 1.4). Note that 191 is the maximum value of the function $f(x)$ in the range considered. Usually, this value is not known. Any value greater than 191 might also be used if positive values are desired. However, depending on the other components — especially the selection operators — this specific constant might affect the performance of the optimization process. Alternatively, a fitness function that might produce negative values is given by the simpler function $f_2(x) = -f(x)$. In this last case, estimating the maximum value of the function is not a requirement. However, not every selection operator can be used with f_2.

$$f(x) = (x - 1)^2 - 5 \tag{1.3}$$

$$f_1(x) = -f(x) + 191 \tag{1.4}$$

Regarding the task of designing a fitness function, this is a straightforward process in some cases because it is just equal to (or a simple transformation of) the objective function. In these cases, the objective function is said to be self-sufficient [Talbi (2009)]. For instance, in the case of the Traveling Salesman Problem, most effective optimizers just consider the minimization of the distance traveled, so the fitness function is viewed as the inverse of the distance. However, in other cases, this task is more complex. For instance, let us consider the Satisfiability Problem. In this case, each solution is mapped in the original problem to a 0 or 1 value, and the aim is to find a solution whose value is 1. Using just the binary

value is not enough to guide the optimization process, so there is a need to define a more complex fitness function.

The definition of the fitness function heavily impacts the expected performance of EAs. The analysis of the fitness landscape is complex, especially for combinatorial optimization, but it is useful to identify drawbacks of the proposed fitness function [Reeves (2000)]. Recent advances such as local optima networks [Adair *et al.* (2019)] facilitate the analyses, especially when dealing with high-dimensional problems. As an example, in the case of maximizing the non-linearity of Boolean functions, the direct use of the non-linearity as the fitness function causes large plateaus that hinder the proper performance of EAs. Extending the fitness function by including additional information from the Walsh-Hadamard transform provides a better guide that results in improved performance [Clark *et al.* (2004); López-López *et al.* (2020)]. Finally, note that in some cases, the fitness function is adapted online once certain issues with the optimization process are detected [Majig and Fukushima (2008)].

An additional issue that might emerge with the design of fitness functions is that they might be too computationally expensive. This would result in just a few generations being evolved, with potentially poor results. Surrogate models can be used to alleviate this difficulty [Shi and Rasheed (2010)]. The main idea behind surrogate models is to approximate the fitness function with a more inexpensive process than the one initially designed. Different ways of building surrogate models have been proposed. Problem-dependent models rely on ad-hoc knowledge of the problem and on the fact that in order to distinguish between very poor and high quality solutions, very accurate functions are not required. Thus, functions with different trade-offs between accuracy and computational cost might be designed and applied at different stages of the optimization process [Jin (2011)]. In contrast, problem-independent models usually rely on machine learning methods and other statistical procedures [Jin (2005)]. Note that while the most typical use of surrogate models is to reduce the computational cost by providing inexpensive fitness functions, they have also been applied to redesign other components of an EA [Shi and Rasheed (2010)].

The selection operator is another important component that has a large impact on the overall performance of EAs [Blickle and Thiele (1996)]. One of the aims of selection is to focus the search on the most promising regions. Some of the first selectors designed made their decisions considering only the individuals' fitness values. However, other factors, such as age or diversity, are usually considered in state-of-the-art EAs [Segura *et al.* (2013)]. Selection operators are applied in two different stages: parent selection and replacement. In both cases, stochastic and deterministic operators can be used, though the most popular choice is to apply stochastic operators in the parent selection stage and deterministic operators in the replacement phase [Eiben and Smith (2003)].

A very popular stochastic selection operator is the *fitness proportional* operator.

Table 1.1 Fitness values and selection probabilities induced by the fitness proportional selection

Individual	Fitness	Selection Prob.
x = 2	195	0.43
x = 5	180	0.40
x = 12	75	0.17

In the fitness proportional selection, the total fitness F is first calculated as the sum of the fitness values of the individuals in the current population. Then, the selection probability of an individual I_i (p_i) is calculated using Equation 1.5, where fit represents the fitness function. In order to apply this selection scheme, the fitness function of every individual must be positive. Let us assume that we are using the fitness function given in Equation 1.4 and that our current population consists of three individuals with $x = 2$, $x = 5$ and $x = 12$. Table 1.1 shows the fitness value and the selection probability associated with each individual. One of the weaknesses of this operator is that it is susceptible to function transposition, which means that its behavior changes if the fitness function of every individual is transposed by adding a constant value. This is illustrated in Figure 1.3. In this figure, a population consisting of individuals A, B and C is used. Two fitness functions are considered: the first one (f_1) is shown in Equation 1.4, while the second one (f_2) is generated by replacing the value 191 in Equation 1.4 with the value 1000. Figure 1.3 shows the fitness values of the different individuals, as well as the selection probabilities associated with the individuals when f_1 and f_2 are applied. Since the selection scheme is susceptible to function transposition, these probabilities differ, as can be seen in the pie charts. In addition, note that with f_2, the selection pressure is very low, i.e. all individuals are selected with very similar probabilities. The reason is that when the differences between the fitness values are small with respect to their absolute values, the selection pressure vanishes. This last effect is another of the known weaknesses that can affect fitness proportional selection.

$$p_i = \frac{fit(I_i)}{F} \tag{1.5}$$

Another quite popular stochastic selection operator is the *tournament selection*. In this case, each time an individual must be selected, there is a competition between k randomly selected individuals, with the best one being chosen. Tournament selection has very different features than *fitness proportional* selection. For instance, it is invariant to transposition of the fitness function. However, this does not mean that it is superior to *fitness proportional* selection. For instance, when facing some fitness landscapes, *fitness proportional* selection can induce a larger focus on exploitation than *tournament selection*. Depending on the problem at hand, on the stopping criterion and on the other components applied, this might be desired

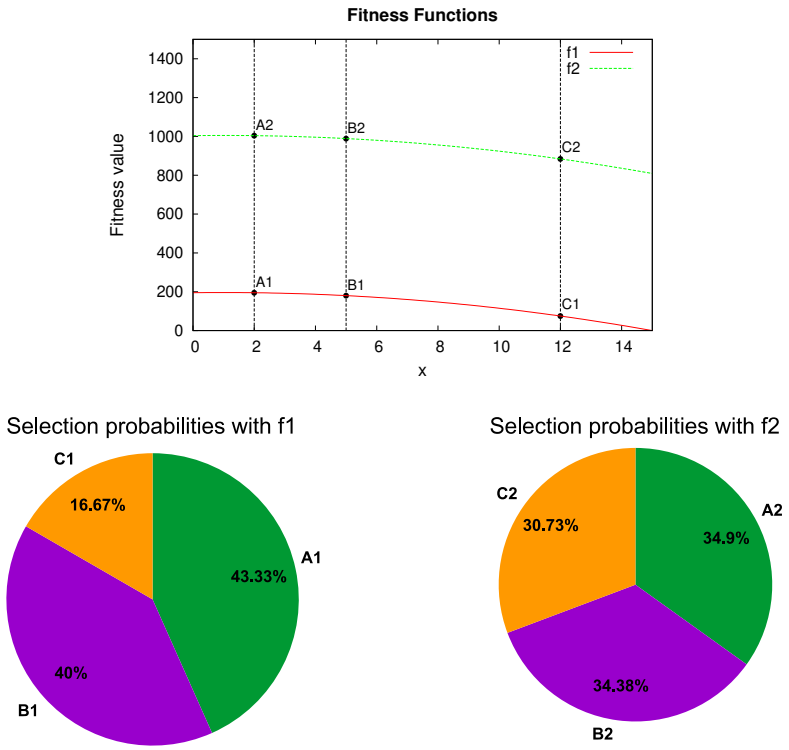

Fig. 1.3 Effect of transposition on fitness proportional selection

or counterproductive, so it is the task of the designer to analyze the interactions between the selectors and the fitness landscape.

The variation phase is another quite important component that is usually adapted to the problem at hand. Among the different operators, the crossover and mutation operators are the most popular, and a large number of different choices have been devised for each of them. For instance, [Eiben and Smith (2003)] presents several operators that can be applied with binary, integer, floating-point and permutation representations. In order to better illustrate the working operation of crossover, two popular crossover operators that can be applied with binary, integer and floating-point representations are presented: the one-point and the uniform crossover. One-point crossover (Figure 1.4) operates by choosing a random gene, and then splitting both parents at this point and creating the two children by exchanging the tails. Alternatively, uniform crossover (Figure 1.5) works by treating each gene independently, and the parent associated with each gene is selected randomly. This is implemented by generating a string with L random variables from a uniform distribution over $[0, 1]$, where L is the number of genes. In each position, if the value is below 0.5, the gene is inherited from the first parent; otherwise, it

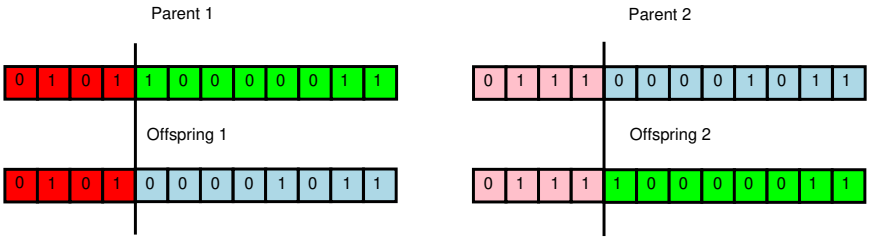

Fig. 1.4 Operation of one-point crossover

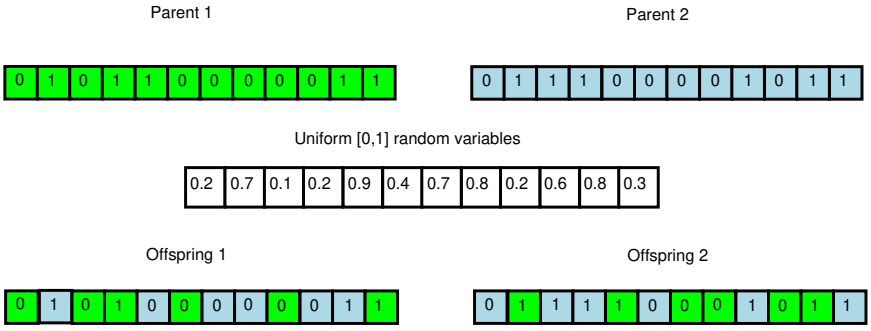

Fig. 1.5 Operation of uniform crossover

is inherited from the second parent. The second offspring is created using inverse mapping.

It is important to note that the crossover schemes discussed induce different linkages between the genes. For instance, in the one-point crossover, genes that are close in the chromosome are usually inherited together, while this is not the case in the uniform crossover. Depending on the meaning of each gene and on its position in the chromosome, introducing this kind of linkage might make sense or not. Thus, when selecting the crossover operator, the representation of the individuals must be taken into account. Finally, operators that try to learn a proper linkage between genes have been devised [Chen and Lim (2008)]. These kinds of operators have yielded significant benefits in several cases.

Note also that in many cases, problem-dependent crossover and mutation operators are designed. Properly designed crossover operators detect characteristics that are important for the problem and that are present in the parents. They then create offspring by combining these features. For instance, in the Graph Coloring Problem, the authors noted that the important feature is not the specific color assigned to each node, but the groups of nodes sharing the same color. Thus, ad-hoc operators that inherit these kinds of features have been designed [Galinier and Hao (1999)]. Additionally, ad-hoc operators might establish a linkage between genes by considering problem-dependent knowledge. For instance, in the Frequency As-

signment Problem, antennas that interact strongly with one another are inherited and/or mutated together with a larger probability than unrelated antennas [Segura *et al.* (2017)]. Finally, in the case of crossover operators, it is also important to distinguish between the parent-centric and mean-centric operators because of the impact that this feature has on the dynamics of the population [Deb *et al.* (2002)]. Properly designed operators that adapt this feature to the needs of the problem at hand have excelled [Jain and Deb (2011)].

Note that while we have discussed different components involved in the design of EAs independently, there are important interactions among them. Thus, regarding their performance, it is not possible to make categorical assertions. Depending on the components used, different balances between exploitation and exploration can be induced. It is therefore very important to recognize how to modify this balance, since it is one of the keys to success. In some cases, there is no explicit mechanism for managing the trade-off between exploration and exploitation. Note that due to the way EAs work, the diversity maintained in the population is one way to control this trade-off. In fact, many authors regard the proper management of diversity as one of the cornerstones of proper performance [Črepinšek *et al.* (2013)]. Thus, several explicit strategies for managing diversity have been proposed [Pandey *et al.* (2014)]. In some way, this can be viewed as another component of EAs, but it is not a completely independent component because it is highly coupled to some of the components already discussed. For instance, in [Črepinšek *et al.* (2013)], diversity management methods are classified as selection-based, population-based, crossover/mutation-based, fitness-based and replacement-based, depending on the sort of component that is modified. Additionally, some implicit mechanisms that alter this degree, such as multi-objectivization, have been devised [Segura *et al.* (2016)].

It is also important to note that for many problems, trajectory-based strategies are usually considered more powerful than EAs in terms of their intensification capabilities. As a result, several state-of-the-art optimizers are hybrids that incorporate trajectory-based strategies and/or other mechanisms to promote intensification [Talbi (2009); Neri *et al.* (2011)]. Thus, knowing the different ways of hybridizing algorithms is an important aspect of mastering the design of EAs.

Finally, given the difficulties in understanding and theoretically analyzing the interactions that appear in EAs, several components and parameterizations are typically tested during the design phase. By analyzing the performance of different components and the reasons for their good or poor performance, some alternative designs might be tested until a good enough scheme is obtained. Moreover, note that it is not just the components that have to be selected. A set of parameters must also be adapted to the problem at hand, and it is important that it is done systematically [Lobo *et al.* (2007); López-Ibáñez *et al.* (2016)]. In this regard, two kinds of methodologies have been devised [Eiben *et al.* (1999)]. *Parameter tuning* can be thought of as searching for good parameters and components before the

Table 1.2 Some aspects to consider when designing EAs

Component or aspect	Related decisions
Individual	Representation
	Evaluation (*fitness function*)
Population	Size
	Population Sizing Scheme
	Initialization
Evolution	Mutation operator
	Crossover operator
	Repair operator
	Intensification operator
	Probabilities for applying operators
	Parent selection
	Replacement scheme
	Diversity Management Strategy
	Hybridization
Stopping criterion	Number of evaluations of individuals
	Generations
	Time
	Given signs of stagnation
	A certain solution quality is achieved
Parameterization	Parameter Control
	Parameter Tuning

algorithm is executed, whereas *parameter control* strategies alter the components and parameters during the run. This can be used, for instance, with the aim of applying an ensemble of variation operators whose probabilities and other internal parameters are changed during the optimization process [Mallipeddi *et al.* (2011)].

Table 1.2 summarizes some of the most important aspects to define when designing and running EAs. Note that in addition to all of these aspects, we also have to consider the internal parameters associated with many of the operators mentioned above. We would like to remark that in the design of proper EAs, it is very important to ascertain the features and implications of using the different components and parameter values and recognize the drawbacks that each of them can circumvent. This is because of the impossibility of testing every parameter and component combination, given how time and computationally consuming this task is. Thus, designing efficient EAs for new problems is a complex task that requires knowledge in several areas that has been developed in recent decades through both experimental and theoretical studies. Generic tools for developing metaheuristics can facilitate this entire process [Luke *et al.* (2007); León *et al.* (2009); Liefooghe *et al.* (2011); Durillo and Nebro (2011)].

1.6 Concluding Remarks

EC is a very active area of research that studies the design and application of a set

of algorithms that draw their inspiration from natural evolution. The first studies that influenced the development of EC date back to at least the 1930s. However, it was during the 1960s when the roots of EC were established. During the first stages of EC, several independent researchers devised different schemes that were applied to a diverse number of applications. Three different types of schemes were developed: EP, ESs and GAs. Although each type of scheme had its own features, further developments made it clear that these schemes had a lot in common, so they began to merge in a natural way. In the 1990s, the terms EC and EA were proposed with the aim of unifying the terminology and they were soon widely adopted by the community. In fact, it is now very difficult to claim that a given state-of-the-art EA belongs to any of the previous classes because they usually combine components from several types of schemes. The number of studies that have been carried out in the last decades is vast. These studies have yielded significant advances in EC; as a result, the number of problems that have been addressed with EAs has grown enormously.

One of the main handicaps of EC is that mastering EAs is a very complex task. One of the reasons is that despite the huge amount of research conducted in recent years, the discipline cannot yet be considered as fully mature. For instance, we are still in a period where different researchers have opposing views with respect to several aspects of EAs, and, therefore, much more research is still required to fully understand EAs and the large number of different components that have been devised by various authors. Thus, designing EAs to tackle new problems is a complex task that involves trial-and-error processes and in-depth analyses in order to understand the interactions and implications of the different components proposed for the problem at hand. Fortunately, many tools that facilitate the use of EAs are now available and recent years have seen significant efforts made to facilitate the application of EAs. In the near future we expect this area to grow even more as it continues to develop on its path to maturity.

Acknowledgements

The first author gratefully acknowledges financial support from CONACyT grant no. 2016-01-1920 (*Investigación en Fronteras de la Ciencia 2016*) and from a project from the 2018 SEP-Cinvestav Fund (application no. 4). The second author acknowledges the financial support from CONACyT through the "Ciencia Básica" project no. 285599. The third author acknowledges the financial support from the Spanish Ministry of Economy, Industry and Competitiveness as part of the programme "I+D+i Orientada a los Retos de la Sociedad" [contract number TIN2016-78410-R].

Bibliography

Adair, J., Ochoa, G., and Malan, K. M. (2019). Local optima networks for continuous fitness landscapes, in *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, GECCO '19 (Association for Computing Machinery, New York, NY, USA), ISBN 9781450367486, p. 14071414, doi:10.1145/3319619.3326852.

Alba, E. (2005). *Parallel Metaheuristics: A New Class of Algorithms* (John Wiley & Sons, NJ, USA).

Altenberg, L. (1995). The Schema Theorem and Price's Theorem, in D. Whitley and M. D. Vose (eds.), *Foundations of Genetic Algorithms 3* (Morgan Kaufmann, San Francisco), pp. 23–49.

Ashlock, D., McGuinness, C., and Ashlock, W. (2012). Representation in Evolutionary Computation, in J. Liu, C. Alippi, B. Bouchon-Meunier, G. W. Greenwood, and H. A. Abbass (eds.), *Advances in Computational Intelligence: IEEE World Congress on Computational Intelligence, WCCI 2012, Brisbane, Australia, June 10-15, 2012. Plenary/Invited Lectures* (Springer, Berlin, Germany), pp. 77–97, iSBN 978-3-642-30687-7.

Bäck, T. (1996). *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms* (Oxford University Press, Oxford, UK).

Bäck, T., Fogel, D. B., and Michalewicz, Z. (1997). *Handbook of Evolutionary Computation*, 1st edn. (IOP Publishing Ltd., Bristol, UK, UK).

Bäck, T., Foussette, C., and Krause, P. (2013). *Contemporary Evolution Strategies*, Natural Computing Series (Springer).

Bäck, T., Hoffmeister, F., and Schwefel, H.-P. (1991). A Survey of Evolution Strategies, in *Proceedings of the Fourth International Conference on Genetic Algorithms* (Morgan Kaufmann), pp. 2–9.

Bäck, T., Rudolph, G., and Schwefel, H.-P. (1993). Evolutionary Programming and Evolution Strategies: Similarities and Differences, in D. B. Fogel and J. W. Atmar (eds.), *Second Annual Conference on Evolutionary Programming* (San Diego, CA), pp. 11–22.

Bäck, T. and Schwefel, H.-P. (1993). An Overview of Evolutionary Algorithms for Parameter Optimization, *Evol. Comput.* **1**, 1, pp. 1–23.

Baldwin, J. (1986). A new factor in evolution, *The American Naturalist* **30**, pp. 441 – 451.

Barricelli, N. A. (1962). Numerical testing of evolution theories, *Acta Biotheoretica* **16**, 1-2, pp. 69–98.

Beyer, H.-G. and Sendhoff, B. (2008). Covariance Matrix Adaptation Revisited - The CMSA Evolution Strategy -, in G. Rudolph, T. Jansen, S. Lucas, C. Poloni, and

N. Beume (eds.), *Parallel Problem Solving from Nature - PPSN X*, *Lecture Notes in Computer Science*, Vol. 5199 (Springer Berlin Heidelberg), pp. 123–132.

Bledsoe, W. (1961). The use of biological concepts in the analytical study of "systems", Tech. Rep. Technical report of talk presented to ORSA-TIMS national meeting held in San Francisco, California, The University of Texas at Austin, USA.

Blickle, T. and Thiele, L. (1996). A comparison of selection schemes used in evolutionary algorithms, *Evol. Comput.* **4**, 4, pp. 361–394.

Blum, C. and Roli, A. (2003). Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison, *ACM Computing Surveys* **35**, 3, pp. 268–308.

Bossert, W. (1967). Mathematical Optimization: Are There Abstract Limits on Natural Selection? in P. S. Moorehead and M. M. Kaplan (eds.), *Mathematical Challenges to the Neo-Darwinian Interpretation of Evolution* (The Wistar Institute Press, Philadelphia, PA), pp. 35–46.

Box, G. E. P. (1957). Evolutionary Operation: A Method for Increasing Industrial Productivity, *Applied Statistics* **6**, 2, pp. 81–101.

Brassard, G. and Bratley, P. (1996). *Fundamentals of Algorithms* (Prentice-Hall, New Jersey).

Bremermann, H. J. (1962). Optimization through evolution and recombination, in M. C. Yovits, G. T. Jacobi, and G. D. Golstine (eds.), *Proceedings of the Conference on Self-Organizing Systems* (Spartan Books, Washington, DC), pp. 93–106.

Burgin, M. and Eberbach, E. (2013). Recursively Generated Evolutionary Turing Machines and Evolutionary Automata, in X.-S. Yang (ed.), *Artificial Intelligence, Evolutionary Computing and Metaheuristics*, *Studies in Computational Intelligence*, Vol. 427 (Springer Berlin Heidelberg), pp. 201–230.

Campbell, D. T. (1960). Blind Variation and Selective Survival as a General Strategy in Knowledge-Processes, in M. C. Yovits and S. Cameron (eds.), *Self-Organizing Systems* (Pergamon Press, New York), pp. 205–231.

Cannon, W. B. (1932). *The wisdom of the body* (W. W. Norton & Company, inc.).

Chellapilla, K. (1997). Evolving Computer Programs Without Subtree Crossover, *IEEE Trans. Evol. Comput.* **1**, 3, pp. 209–216.

Chen, Y. and Lim, M. H. (2008). *Linkage in Evolutionary Computation*, Studies in Computational Intelligence (Springer).

Clark, J. A., Jacob, J. L., and Stepney, S. (2004). Searching for cost functions, in *Proceedings of the 2004 Congress on Evolutionary Computation (IEEE Cat. No.04TH8753)*, Vol. 2, pp. 1517–1524 Vol.2, doi:10.1109/CEC.2004.1331076.

Coello Coello, C. A., Lamont, G. B., and Van Veldhuizen, D. A. (2007). *Evolutionary Algorithms for Solving Multi-Objective Problems*, 2nd edn. (Springer, New York), iSBN 978-0-387-33254-3.

Conrad, M. and Pattee, H. H. (1970). Evolution Experiments with an Artificial Ecosystem, *Journal of Theoretical Biology* **28**, 3, pp. 393 – 409.

Cornett, F. N. (1972). *An Application of Evolutionary Programming to Pattern Recognition*, Master's thesis, New Mexico State University, Las Cruces, New Mexico, USA.

Crosby, J. L. (1960). The Use of Electronic Computation in the Study of Random Fluctuations in Rapidly Evolving Populations, *Philosophical Transactions of the Royal Society of London. Series B, Biological Sciences* **242**, 697, pp. 550–572.

Crosby, J. L. (1967). Computers in the study of evolution, *Science Progress Oxford* **55**, pp. 279–292.

Darwin, C. (1859). *On the Origin of Species by Means of Natural Selection* (Murray), or the Preservation of Favored Races in the Struggle for Life.

Darwin, C. and Wallace, A. (1858). On the Tendency of Species to form Varieties; and on

the Perpetuation of Varieties and Species by Natural Means of Selection, *Zoological Journal of the Linnean Society* **3**, pp. 46–50.

De Jong, K. A. (1975). *An Analysis of the Behavior of a Class of Genetic Adaptive Systems.*, Ph.D. thesis, University of Michigan, Ann Arbor, Michigan, USA.

De Jong, K. A. (2006). *Evolutionary Computation - A Unified Approach* (MIT Press).

de Monet de Lamarck, J. B. P. A. (1809). *Philosophie zoologique: ou Exposition des considérations relative à l'histoire naturelle des animaux* (Chez Dentu).

Dearholt, D. W. (1976). Some experiments on generalization using evolving automata, in *9th Hawaii Internation Conference on System Sciences* (Western Periodicals, Honolulu), pp. 131–133.

Deb, K., Anand, A., and Joshi, D. (2002). A computationally efficient evolutionary algorithm for real-parameter optimization, *Evolutionary Computation* **10**, 4, pp. 371–395, doi:10.1162/106365602760972767.

Dunham, B., Fridshal, D., Fridshal, R., and North, J. H. (1963). Design by natural selection, *Synthese* **15**, 1, pp. 254–259.

Durillo, J. J. and Nebro, A. J. (2011). jmetal: A java framework for multi-objective optimization, *Advances in Engineering Software* **42**, pp. 760–771, doi:DOI:10.1016/j.advengsoft.2011.05.014.

Eiben, A. E., Aarts, E. H. L., and Hee, K. M. (1991). Global Convergence of Genetic Algorithms: a Markov Chain Analysis, in H.-P. Schwefel and R. Männer (eds.), *Parallel Problem Solving from Nature*, *Lecture Notes in Computer Science*, Vol. 496 (Springer Berlin Heidelberg), pp. 3–12.

Eiben, A. E. and Bäck, T. (1997). Empirical Investigation of Multiparent Recombination Operators in Evolution Strategies, *Evol. Comput.* **5**, 3, pp. 347–365.

Eiben, A. E., Hinterding, R., and Michalewicz, Z. (1999). Parameter control in evolutionary algorithms, *IEEE Transactions on Evolutionary Computation* **3**, 2, pp. 124–141, doi:10.1109/4235.771166.

Eiben, A. E. and Smith, J. E. (2003). *Introduction to Evolutionary Computing*, Natural Computing Series (Springer).

Eshelman, L. (1991). The CHC Adaptive Search Algorithm : How to Have Safe Search When Engaging in Nontraditional Genetic Recombination, *Foundations of Genetic Algorithms* , pp. 265–283.

Evans, C. R. and Robertson, A. D. J. (eds.) (1968). *Cybernetics: key papers* (University Park Press, Baltimore, MD, USA).

Everett, J. (2000). Model Building, Model Testing and Model Fitting, in L. D. Chambers (ed.), *The Practical Handbook of Genetic Algorithms: Applications, Second Edition* (CRC Press, Inc., Boca Raton, Florida, USA), pp. 1–30.

Fisher, R. (1930). *The Genetical Theory of Natural Selection* (Oxford University Press), ISBN 0-19-850440-3.

Fogel, D. B. (1994). An introduction to simulated evolutionary optimization, *IEEE Trans. Neural Netw.* **5**, 1, pp. 3–14.

Fogel, D. B. (1995). *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence* (IEEE Press, Piscataway, NJ, USA).

Fogel, D. B. (1998). *Evolutionary Computation: The Fossil Record* (Wiley-IEEE Press).

Fogel, D. B. and Anderson, R. W. (2000). Revisiting Bremermann's genetic algorithm. I. Simultaneous mutation of all parameters, in *2000 IEEE Congress on Evolutionary Computation*, Vol. 2, pp. 1204–1209 vol.2.

Fogel, D. B. and Atmar, J. W. (1990). Comparing genetic operators with gaussian mutations in simulated evolutionary processes using linear systems, *Biological Cybernetics* **63**, 2, pp. 111–114.

Fogel, D. B. and Chellapilla, K. (1998). Revisiting evolutionary programming, in S. K. Rogers, D. B. Fogel, J. C. Bezdek, and B. Bosacchi (eds.), *Applications and Science of Computational Intelligence*, Vol. 3390 (SPIE), pp. 2–11.

Fogel, D. B., Fogel, L. J., and Atmar, J. W. (1991). Meta-Evolutionary Programming, in *Asilomar Conference in Signals Systems and Computers, 1991*, pp. 540–545.

Fogel, D. B. and Ghozeil, A. (1997). A note on representations and variation operators, *IEEE Trans. Evol. Comput.* **1**, 2, pp. 159–161.

Fogel, L. J. (1962). Autonomous automata, *Industrial Research* **4**, pp. 14–19.

Fogel, L. J. (1999). *Intelligence through simulated evolution: forty years of evolutionary programming*, Wiley series on intelligent systems (Wiley).

Fogel, L. J., Angeline, P. J., and Fogel, D. B. (1995). An Evolutionary Programming Approach to Self-Adaptation on Finite State Machines, in *Annual Conference on Evolutionary Programming IV*, pp. 355–365.

Fogel, L. J. and Burgin, G. H. (1969). Competitive Goal-seeking Through Evolutionary Programming, Tech. Rep. Contract AF 19(628)-5927, Air Force Cambridge Research Labs.

Fogel, L. J. and Fogel, D. B. (1986). Artificial intelligence through evolutionary programming, Tech. Rep. PO-9-X56-1102C-1, U.S. Army Research Institute, San Diego, CA.

Fogel, L. J., Owens, A. J., and Walsh, M. J. (1966). *Artificial Intelligence Through Simulated Evolution* (John Wiley & Sons).

Fraser, A. S. (1957). Simulation of genetic systems by automatic digital computers. I. Introduction, *Australian Journal of Biological Science* **10**, pp. 484–491.

Fraser, A. S. and Burnell, D. (1967). Simulation of Genetic Systems XII. Models of Inversion Polymorphism, *Genetics* **57**, pp. 267–282.

Fraser, A. S. and Burnell, D. (1970). *Computer models in genetics* (McGraw-Hill, NY).

Friedberg, R. M. (1958). A Learning Machine: Part I, *IBM Journal of Research and Development* **2**, 1, pp. 2–13.

Friedberg, R. M., Dunham, B., and North, J. H. (1959). A Learning Machine: Part II, *IBM Journal of Research and Development* **3**, 3, pp. 282–287.

Friedman, G. J. (1956). *Selective Feedback Computers for Engineering Synthesis and Nervous System Analogy*, Master's thesis, University of California, Los Angeles.

Galinier, P. and Hao, J.-K. (1999). Hybrid evolutionary algorithms for graph coloring, *Journal of Combinatorial Optimization* **3**, pp. 379–397.

Gill, J. L. (1965). Effects of finite size on selection advance in simulated genetic populations, *Australian J. Biological Sciences* **18**, pp. 599–617.

Glover, F. and Kochenberger, G. A. (eds.) (2003). *Handbook of Metaheuristics* (Kluver Academic Publishers), ISBN 1-4020-7263-5.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*, Artificial Intelligence (Addison-Wesley).

Goldberg, D. E. (1990). Real-coded genetic algorithms, virtual alphabets, and blocking, *Complex Systems* **5**, pp. 139–167.

Goldberg, D. E., Korb, B., and Deb, K. (1989). Messy Genetic Algorithms: Motivation, Analysis, and First Results, *Complex Systems* **3**, 5, pp. 493–530.

Goldberg, D. E. and Richardson, J. (1987). Genetic algorithms with sharing for multimodal function optimization, in *Proceedings of the Second International Conference on Genetic algorithms and their Application* (L. Erlbaum Associates Inc., Hillsdale, NJ, USA), pp. 41–49.

Grefenstette, J. (1987). Incorporating Problem Specific Knowledge into Genetic Algorithms, in L. Davis (ed.), *Genetic Algorithms and Simulated Annealing*, pp. 42–60.

Haldane, J. B. S. (1932). *The Causes of Evolution* (Longman, Green and Co.), ISBN

0-691-02442-1.

Hansen, N. and Ostermeier, A. (2001). Completely Derandomized Self-Adaptation in Evolution Strategies, *Evol. Comput.* **9**, 2, pp. 159–195.

Heckendorn, R. B., Whitley, D., and Rana, S. (1996). Nonlinearity, Hyperplane Ranking and the Simple Genetic Algorithm, in R. K. Belew and M. D. Vose (eds.), *Foundations of Genetic Algorithms 4*, pp. 181–201.

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems* (University of Michigan Press, Ann Arbor, MI, USA).

Huxley, J. S. (1942). *Evolution: the modern synthesis* (Allen and Unwin).

Igel, C., Hansen, N., and Roth, S. (2007). Covariance Matrix Adaptation for Multi-objective Optimization, *Evol. Comput.* **15**, 1, pp. 1–28.

Jain, A. and Fogel, D. B. (2000). Case studies in applying fitness distributions in evolutionary algorithms. II. Comparing the improvements from crossover and Gaussian mutation on simple neural networks, in *2000 IEEE Symposium on Combinations of Evolutionary Computation and Neural Networks*, pp. 91–97.

Jain, H. and Deb, K. (2011). Parent to mean-centric self-adaptation in sbx operator for real-parameter optimization, in B. K. Panigrahi, P. N. Suganthan, S. Das, and S. C. Satapathy (eds.), *Swarm, Evolutionary, and Memetic Computing* (Springer Berlin Heidelberg, Berlin, Heidelberg), ISBN 978-3-642-27172-4, pp. 299–306.

Jin, Y. (2005). A comprehensive survey of fitness approximation in evolutionary computation, *Soft Comput.* **9**, 1, p. 312, doi:10.1007/s00500-003-0328-5.

Jin, Y. (2011). Surrogate-assisted evolutionary computation: Recent advances and future challenges, *Swarm and Evolutionary Computation* **1**, 2, pp. 61 – 70, doi:https://doi.org/10.1016/j.swevo.2011.05.001.

Kaufman, H. (1967). An Experimental Investigation of Process Identification by Competitive Evolution, *IEEE Trans. Syst. Sci. Cybern.* **3**, 1, pp. 11–16.

Koza, J. R. (1990). Genetic Programming: A Paradigm for Genetically Breeding Populations of Computer Programs to Solve Problems, Tech. rep., Stanford University, Stanford, California, USA.

León, C., Miranda, G., and Segura, C. (2009). Metco: a Parallel Plugin-Based Framework for Multi-Objective Optimization, *International Journal on Artificial Intelligence Tools* **18**, 4, pp. 569–588.

Lewontin, R. C. (1964). The Interaction of Selection and Linkage. I. General Considerations; Heterotic Models, *Genetics* **49**, 1, pp. 49–67.

Li, R., Emmerich, M. T. M., Eggermont, J., Bäck, T., Schütz, M., Dijkstra, J., and Reiber, J. H. C. (2013). Mixed Integer Evolution Strategies for Parameter Optimization, *Evol. Comput.* **21**, 1, pp. 29–64.

Liefooghe, A., Jourdan, L., and Talbi, E.-G. (2011). A software framework based on a conceptual unified model for evolutionary multiobjective optimization: Paradiseo-moeo, *European Journal of Operational Research* **209**, 2, pp. 104 – 112.

Linnaeus, C. (1735). *Systema naturae, sive regna tria naturae systematice proposita per secundum classes, ordines, genera, & species, cum characteribus, differentiis, synonymis, locis* (Theodorum Haak).

Lobo, F. G., Lima, C. F., and Michalewicz, Z. (2007). *Parameter Setting in Evolutionary Algorithms*, 1st edn. (Springer Publishing Company, Incorporated).

López-Ibáñez, M., Dubois-Lacoste, J., Pérez Cáceres, L., Stützle, T., and Birattari, M. (2016). The irace package: Iterated racing for automatic algorithm configuration, *Operations Research Perspectives* **3**, pp. 43–58, doi:10.1016/j.orp.2016.09.002.

López-López, I., Gómez, G. S., Segura, C., Oliva, D., and Rojas, O. (2020). Metaheuristics in the optimization of cryptographic boolean functions, *Entropy* **22**, 9, pp. 1–25, doi:

10.3390/e22091052.

Luke, S., Panait, L., Balan, G., and Et (2007). ECJ: A Java-based Evolutionary Computation Research System, http://cs.gmu.edu/˜eclab/projects/ecj/.

Lyle, M. (1972). *An Investigation Into Scoring Techniques in Evolutionary Programming*, Master's thesis, New Mexico State University, Las Cruces, New Mexico, USA.

Majig, M. and Fukushima, M. (2008). Adaptive fitness function for evolutionary algorithm and its applications, in *International Conference on Informatics Education and Research for Knowledge-Circulating Society (icks 2008)*, pp. 119–124, doi: 10.1109/ICKS.2008.12.

Mallipeddi, R., Suganthan, P., Pan, Q., and Tasgetiren, M. (2011). Differential evolution algorithm with ensemble of parameters and mutation strategies, *Applied Soft Computing* **11**, 2, pp. 1679 – 1696, doi:https://doi.org/10.1016/j.asoc.2010.04.024, the Impact of Soft Computing for the Progress of Artificial Intelligence.

Mendel, G. (1865). Experiments in Plant Hibridization, in *Brünn Natural History Society*, pp. 3–47.

Mezura-Montes, E. (2009). *Constraint-Handling in Evolutionary Optimization*, 1st edn. (Springer).

Mezura-Montes, E. and Coello Coello, C. A. (2005). A simple multimembered evolution strategy to solve constrained optimization problems, *IEEE Trans. Evol. Comput.* **9**, 1, pp. 1–17.

Michalewicz, Z. (1994). *Genetic Algorithms + Data Structures = Evolution Programs* (Springer-Verlag New York, Inc., New York, NY, USA).

Mitchell, M. (1998). *An Introduction to Genetic Algorithms* (MIT Press, Cambridge, MA, USA).

Moscato, P. (1989). On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts: Towards Memetic Algorithms, Tech. Rep. C3P Report 826, California Institute of Technology.

Neri, F., Cotta, C., and Moscato, P. (2011). *Handbook of Memetic Algorithms* (Springer Publishing Company, Incorporated), ISBN 3642232469.

Nix, A. and Vose, M. D. (1992). Modeling genetic algorithms with Markov chains, *Annals of Mathematics and Artificial Intelligence* **5**, 1, pp. 79–88.

Nocedal, J. and Wright, S. J. (2006). *Numerical Optimization*, 2nd edn. (Springer, New York, NY, USA).

Ostermeier, A. (1992). An Evolution Strategy with Momentum Adaptation of the Random Number Distribution, in R. Männer and B. Manderick (eds.), *Parallel Problem Solving from Nature 2, PPSN-II* (Elsevier, Brussels, Belgium), pp. 199–208.

Ostermeier, A., Gawelczyk, A., and Hansen, N. (1994a). A Derandomized Approach to Self-adaptation of Evolution Strategies, *Evol. Comput.* **2**, 4, pp. 369–380.

Ostermeier, A., Gawelczyk, A., and Hansen, N. (1994b). Step-size adaptation based on non-local use of selection information, in Y. Davidor, H.-P. Schwefel, and R. Männer (eds.), *Parallel Problem Solving from Nature PPSN III*, *Lecture Notes in Computer Science*, Vol. 866 (Springer Berlin Heidelberg), pp. 189–198.

Palmer, C. (1994). An approach to a problem in network design using genetic algorithms unpublished phd thesis, *Polytechnic University, Troy, NY* .

Pandey, H. M., Chaudhary, A., and Mehrotra, D. (2014). A comparative review of approaches to prevent premature convergence in ga, *Applied Soft Computing* **24**, pp. 1047–1077.

Poland, J. and Zell, A. (2001). Main Vector Adaptation: A CMA Variant with Linear Time and Space Complexity, in L. Spector, E. D. Goodman, A. Wu, W. B. Langdon, H.-M. Voigt, M. Gen, S. Sen, M. Dorigo, S. Pezeshk, M. H. Garzon, and E. Burke (eds.),

Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001) (Morgan Kaufmann, San Francisco, California, USA), pp. 1050–1055.

Porto, V. W. and Fogel, D. B. (1995). Alternative neural network training methods [active sonar processing], *IEEE Expert* **10**, 3, pp. 16–22.

Radcliffe, N. J. (1992). Non-linear genetic representations, in R. Männer and B. Manderick (eds.), *Parallel Problem Solving from Nature 2, PPSN-II, Brussels, Belgium, September 28-30, 1992* (Elsevier), pp. 261–270.

Radcliffe, N. J. (1997). Schema processing, in T. Bäck, D. B. Fogel, and Z. Michalewicz (eds.), *Handbook of Evolutionary Computation* (Institute of Physics Publishing and Oxford University Press, Bristol, New York), pp. B2.5:1–10.

Rechenberg, I. (1965). Cybernetic solution path of an experimental problem, Tech. Rep. Library Translation 1122, Royal Air Force Establishment.

Rechenberg, I. (1973). *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution* (Frommann-Holzboog, Stutgart).

Rechenberg, I. (1978). Evolutionsstrategin, in B. Schneider and U. Ranft (eds.), *Simulationsmethoden in der Medizin und Biologie* (Springer-Verlag, Berlin, Germinay), pp. 83–114.

Reed, J., Toombs, R., and Barricelli, N. A. (1967). Simulation of biological evolution and machine learning: I. Selection of self-reproducing numeric patterns by data processing machines, effects of hereditary control, mutation type and crossing, *Journal of Theoretical Biology* **17**, 3, pp. 319 – 342.

Reeves, C. R. (2000). Fitness landscapes and evolutionary algorithms, in C. Fonlupt, J.-K. Hao, E. Lutton, M. Schoenauer, and E. Ronald (eds.), *Artificial Evolution* (Springer Berlin Heidelberg, Berlin, Heidelberg), ISBN 978-3-540-44908-9, pp. 3–20.

Richter, H. (2014). Fitness landscapes: From evolutionary biology to evolutionary computation, in H. Richter and A. Engelbrecht (eds.), *Recent Advances in the Theory and Application of Fitness Landscapes*, *Emergence, Complexity and Computation*, Vol. 6 (Springer Berlin Heidelberg), pp. 3–31.

Richter, J. N., Wright, A., and Paxton, J. (2008). Ignoble Trails - Where Crossover Is Provably Harmful, in G. Rudolph, T. Jansen, S. Lucas, C. Poloni, and N. Beume (eds.), *Parallel Problem Solving from Nature PPSN X*, *Lecture Notes in Computer Science*, Vol. 5199 (Springer Berlin Heidelberg), pp. 92–101.

Ronald, S. (1997). Robust encodings in genetic algorithms: a survey of encoding issues, in *Proceedings of 1997 IEEE International Conference on Evolutionary Computation (ICEC '97)*, pp. 43–48, doi:10.1109/ICEC.1997.592265.

Rothlauf, F. (2006). *Representations for Genetic and Evolutionary Algorithms* (Springer-Verlag, Berlin, Heidelberg), ISBN 354025059X.

Rothlauf, F. (2011). *Design of Modern Heuristics: Principles and Application*, 1st edn. (Springer Publishing Company, Incorporated), ISBN 3540729615.

Rudolph, G. (2012). Evolutionary Strategies, in G. Rozenberg, T. Bäck, and J. Kok (eds.), *Handbook of Natural Computing* (Springer Berlin Heidelberg), pp. 673–698.

Schraudolph, N. and Belew, R. (1992). Dynamic Parameter Encoding for genetic algorithms, *Machine Learning* **9**, 1, pp. 9–21.

Schwefel, H.-P. (1965). *Kybernetische Evolution als Strategie der experimentellen Forschung in der Strömungstechnik*, Dipl.-Ing. Thesis, Technical University of Berlin, Hermann Föttinger–Institute for Hydrodynamics.

Schwefel, H.-P. (1981). *Numerical Optimization of Computer Models* (John Wiley & Sons, Inc., New York, NY, USA).

Segura, C., Coello, C., Miranda, G., and León, C. (2016). Using multi-objective evolutionary algorithms for single-objective constrained and unconstrained optimization,

Annals of Operations Research **240**, pp. 217–250.

Segura, C., Coello, C., Segredo, E., Miranda, G., and Leon, C. (2013). Improving the diversity preservation of multi-objective approaches used for single-objective optimization, in *2013 IEEE Congress on Evolutionary Computation (CEC)*, pp. 3198–3205.

Segura, C., Hernndez-Aguirre, A., Luna, F., and Alba, E. (2017). Improving diversity in evolutionary algorithms: New best solutions for frequency assignment, *IEEE Transactions on Evolutionary Computation* **21**, 4, pp. 539–553, doi:10.1109/TEVC.2016.2641477.

Shi, L. and Rasheed, K. (2010). A Survey of Fitness Approximation Methods Applied in Evolutionary Algorithms, in Y. Tenne and C.-K. Goh (eds.), *Computational Intelligence in Expensive Optimization Problems* (Springer, Berlin, Germany), pp. 3–28, iSBN 978-3-642-10701-6.

Sivanandam, S. N. and Deepa, S. N. (2007). *Introduction to Genetic Algorithms* (Springer).

Storn, R. and Price, K. (1995). Differential Evolution: A simple and efficient adaptive scheme for global optimization over continuous spaces, Tech. rep., International Computer Science Institute, Berkeley, Tech. Rep. TR95012.

Syswerda, G. (1989). Uniform Crossover in Genetic Algorithms, in *Proceedings of the 3rd International Conference on Genetic Algorithms* (Morgan Kaufmann Publishers Inc., San Francisco, CA, USA), pp. 2–9.

Talbi, E.-G. (2009). *Metaheuristics: From Design to Implementation* (John Wiley & Sons), ISBN 978-0470278581.

Turing, A. M. (1948). Intelligent Machinery, Report, National Physical Laboratory, Teddington, UK.

Turing, A. M. (1950). Computing Machinery and Intelligence, *Mind* **59**, pp. 433–460.

Črepinšek, M., Liu, S.-H., and Mernik, M. (2013). Exploration and Exploitation in Evolutionary Algorithms: A Survey, *ACM Comput. Surv.* **45**, 3, pp. 35:1–35:33.

Vose, M. D. and Liepins, G. (1991). Punctuated equilibria in genetic search, *Complex Systems* **5**, pp. 31–44.

Weise, T. (2008). *Global Optimization Algorithms - Theory and Application* (http://www.it-weise.de/).

Whitley, D. and Kauth, J. (1988). *GENITOR: A different genetic algorithm* (Colorado State University, Department of Computer Science).

Whitley, D. and Sutton, A. (2012). Genetic Algorithms - A Survey of Models and Methods, in G. Rozenberg, T. Bäck, and J. Kok (eds.), *Handbook of Natural Computing* (Springer Berlin Heidelberg), pp. 637–671.

Wilkerson, J. L. and Tauritz, D. R. (2011). A guide for fitness function design, in *Proceedings of the 13th Annual Conference Companion on Genetic and Evolutionary Computation*, GECCO '11 (Association for Computing Machinery, New York, NY, USA), ISBN 9781450306904, p. 123124, doi:10.1145/2001858.2001929.

Wilson, D., Rodrigues, S., Segura, C., Loshchilov, I., Hutter, F., Buenfil, G. L., Kheiri, A., Keedwell, E., Ocampo-Pineda, M., zcan, E., Pea, S. I. V., Goldman, B., Rionda, S. B., Hernndez-Aguirre, A., Veeramachaneni, K., and Cussat-Blanc, S. (2018). Evolutionary computation for wind farm layout optimization, *Renewable Energy* **126**, pp. 681 – 691, doi:https://doi.org/10.1016/j.renene.2018.03.052.

Wright, S. (1931). Evolution in Mendelian Populations, *Genetics* **16**, pp. 97–159.

Wright, S. (1932). The roles of mutation, inbreeding, crossbreeding and selection in evolution, in *VI International Congress of Genetics*, Vol. 1, pp. 356–366.

Yao, X. and Liu, Y. (1997). Fast evolution strategies, in P. Angeline, R. Reynolds, J. McDonnell, and R. Eberhart (eds.), *Evolutionary Programming VI*, *Lecture Notes in Computer Science*, Vol. 1213 (Springer Berlin Heidelberg), pp. 149–161.

Yao, X., Liu, Y., and Lin, G. (1999). Evolutionary programming made faster, *IEEE Trans. Evol. Comput.* **3**, 2, pp. 82–102.

Young, S. S. (1966). Computer simulation of directional selection in large populations. i. the programme, the additive and the dominance models, *Genetics* **53**, pp. 189–205.