Evolutionary Optimization

Christian Blum, Raymond Chiong, Maurice Clerc, Kenneth De Jong, Zbigniew Michalewicz, Ferrante Neri, and Thomas Weise

This is a preview version of paper [1] (see page 31 for the reference). It is posted here for your personal use and not for redistribution. The final publication and definite version is available from Springer (who hold the copyright) at http://link.springer.com. See also http://dx.doi.org/10.1007/978-3-642-23424-8_1. (and the book Variants of Evolutionary Algorithms for Real-World Applications).

Abstract The emergence of different metaheuristics and their new variants in recent years has made the definition of the term *Evolutionary Algorithms*

Christian Blum
ALBCOM Research Group, Universitat Politècnica de Catalunya, Barcelona, Spain, e-mail: cblum@lsi.upc.edu

Raymond Chiong
Faculty of Information & Communication Technologies, Swinburne University of Technology, Victoria 3122, Australia, e-mail: rchiong@swin.edu.au

Maurice Clerc
Independent Consultant, Groisy, France, e-mail: Maurice.Clerc@WriteMe.com

Kenneth De Jong
Department of Computer Science, George Mason University, Fairfax, VA 22030, USA, e-mail: kdejong@gmu.edu

Zbigniew Michalewicz
School of Computer Science, University of Adelaide, South Australia 5005, Australia; also at the Institute of Computer Science, Polish Academy of Sciences, ul. Ordona 21, 01-237 Warsaw, Poland, and the Polish-Japanese Institute of Information Technology, ul. Koszykowa 86, 02-008 Warsaw, Poland; e-mail: zbyszek@cs.adelaide.edu.au.

Ferrante Neri
Department of Mathematical Information Technology, P. O. Box 35 (Agora), 40014 University of Jyväskylä, Finland, e-mail: ferrante.neri@jyu.fi

Thomas Weise
Nature Inspired Computation and Applications Laboratory (NICAL), School of Computer Science and Technology, University of Science and Technology of China (USTC), Héféi 230027, Ānhuī, China, e-mail: tweise@ustc.edu.cn (corresponding author)

unclear. Originally, it was coined to put a group of stochastic search algorithms that mimic natural evolution together. While some people would still see it as a specific term devoted to this group of algorithms, including Genetic Algorithms, Genetic Programming, Evolution Strategies, Evolutionary Programming, and to a lesser extent Differential Evolution and Estimation of Distribution Algorithms, many others would regard "Evolutionary Algorithms" as a general term describing population-based search methods that involve some form of randomness and selection. In this chapter, we re-visit the fundamental question of "*what is an Evolutionary Algorithm?*" not only from the traditional viewpoint but also the wider, more modern perspectives relating it to other areas of Evolutionary Computation. To do so, apart from discussing the main characteristics of this family of algorithms we also look at Memetic Algorithms and the Swarm Intelligence algorithms. From our discussion, we see that establishing semantic borders between these algorithm families is not always easy, nor necessarily useful. It is anticipated that they will further converge as the research from these areas cross-fertilizes each other.

1 Introduction

Almost any design or decision task encountered in business, industry, or public services is, by its nature, an optimization problem. How can a ship be designed for *highest* safety and *maximum* cargo capacity at the same time? How should the production in a factory be scheduled in order to satisfy all customer requests *as soon* and timely *as possible*? How can multiple sclerosis lesions on an MRI be identified with the *best* precision? ... Three completely different questions and scenarios, three optimization problems as encountered by practitioners every day.

From the management perspective, an optimization problem is a situation which requires deciding for one choice from a set of possible alternatives in order to reach a predefined/required benefit at minimal costs. From a mathematical point of view, solving an optimization problem requires finding an input value $x^\star$ for which a so-called objective function f takes on the smallest (or largest) possible value (while obeying to some restrictions on the possible values of $x^\star$). In other words, every task that has the goal of approaching certain configurations considered as optimal in the context of pre-defined criteria can be viewed as an optimization problem.

Many optimization algorithms for solving complex real-world problems nowadays are based on metaheuristic methods as opposed to traditional operations research techniques. The reason is simple – this is due to the *complexity* of the problems. Real-world problems are usually difficult to solve for several reasons, some of which include:

1. The number of possible solutions may be too large so that an exhaustive search for the best answer becomes infeasible.
2. The objective function f may be noisy or varies with time, thereby requiring not just a single solution but an entire series of solutions.
3. The possible solutions are so heavily constrained that constructing even one feasible answer is difficult, let alone searching for an optimum solution.

Naturally, this list could be extended to include many other possible obstacles. For example, noise associated with the observations and measurements, uncertainties about the given information, problems that have multiple conflicting objectives, just to mention a few. Moreover, computing the objective values may take much time and thus, the feasible number of objective function invocations could be low. All these reasons are just some of the aspects which can make an optimization problem difficult (see [2]; and also [3**?**] for an in-depth discussion on this topic).

It is worth noting that every time a problem is "solved", in reality what has been discovered is only the solution to a *model* of the problem – and all models are simplification of the real world. When trying to solve the Travelling Salesman Problem (TSP), for example, the problem itself is usually modeled as a graph where the nodes correspond to cities and the edges are annotated with costs representing, e.g., the distances between the cities. Parameters such as traffic, the weather, petrol prices and times of the day are typically omitted.

In view of this, the general process of solving an optimization problem hence consists of two separate steps: (1) creating a model of the problem, and (2) using that model to generate a solution.

Problem $\Rightarrow$ Model $\Rightarrow$ Solution.

Again, the "solution" here is only a solution in terms of the model. If the model has a high degree of fidelity, this "solution" is more likely to be meaningful. In contrast, if the model has too many unfulfilled assumptions and rough approximations, the solution may be meaningless, or worse. From this perspective, there are at least two ways to proceed in solving real-world problems:

1. Trying to simplify the model so that conventional methods might return better answers.
2. Keeping the model with all its complexities and using non-conventional approaches to find a near-optimum solution.

So, the more difficult the problem is, the more appropriate it is to use a metaheuristic method. Here, we see that it will anyway be difficult to obtain precise solutions to a problem, since we have to approximate either the model or the solution. A large volume of experimental evidence has shown that the latter approach can often be used to practical advantages.

In recent years, we have seen the emergence of different types of metaheuristics. This gives rise to many new variants and concepts, making some of the fundamental views in the field no longer clear-cut. In this chapter, our focus is to discuss what Evolutionary Algorithms (EAs) – one of the most popular metaheuristic methods – are and how they differ from other metaheuristics. The aim is not to give a definitive answer to the question "What is an EA?" – it is almost impossible for anyone to do so. Instead, we will systematically explore and discuss the traditional and modern views of this topic. We start by describing what metaheuristics are, followed by the core question of what EAs are. We then present some of the most well-known EAs, such as Genetic Algorithms (GAs), Genetic Programming (GP), Evolution Strategies (ES) and Evolutionary Programming (EP). After that, we extend our discussion to Memetic Computing, taking a look at the relevance/connection between EAs and Memetic Algorithms (MAs). Finally, we also discuss the similarities and differences between EAs and the Swarm Intelligence algorithms such as Particle Swarm Optimization (PSO) and Ant Colony Optimization (ACO).

2 Metaheuristics

The field of metaheuristics has a rich history. During the second half of the 20th century, with the development of computational devices and demands of industrial processes, the necessity to solve some optimization problems arose despite the fact that there was not sufficient prior knowledge (hypotheses) on the optimization problem for the application of an exact method. In fact, in the majority of industrial cases, the problems are highly nonlinear, or characterized by a noisy fitness, or without an explicit analytical expression as the objective function might be the result of an experimental or simulation process. In this context, the earliest metaheuristics have been designed. The term metaheuristic, from the greek *meta-euriskein* which means beyond the search, refers to a computational method which progressively attempts to improve one or more candidate solutions while searching for the optimum.

Whenever an optimization problem is to be solved, we expect that there is some kind of utility measure which defines how good a solution is or how high the costs are. Usually this measure is given in the form of a mathematical function f. Then, as stated before, the inputs for which the function takes on the minimal value is sought. Sometimes, multiple such functions have to be optimized simultaneously.

A metaheuristic is a method for solving a *general* class of optimization problems. It combines utility measures in an abstract and hopefully efficient manner, typically without utilizing deeper insights into their inner structure. Metaheuristics do not require hypotheses on the optimization problem nor any kind of prior knowledge on the objective function. The treatment of objective functions as "black boxes" [4–7] is the most prominent and attractive feature of metaheuristics. Metaheuristics obtain knowledge about the structure of an optimization problem by utilizing statistics obtained from the possible solutions (i.e., candidate solutions) evaluated in the past. This knowledge is used to construct new candidate solutions which are likely to have a high utility.

Many different types of metaheuristics emerged during the last 30 years, and the majority of them have been inspired by some aspects of the nature (see [8] for a recent collection of nature-inspired algorithms). These include a variety of Hill Climbing techniques (deterministic and stochastic), the Swarm Intelligence algorithms (PSO and ACO), Artificial Immune Systems, Differential Evolution, Simulated Annealing, Tabu Search, Cultural Algorithms, Iterated Local Search, Variable Neighborhood Search, and – of course – Evolutionary and co-Evolutionary Algorithms.

Metaheuristics can be classified based on different criteria. For example, some of them process a single solution (e.g., Simulated Annealing), whereas some others process a set of solutions and are called population-based methods (e.g., EAs). Some metaheuristics are deterministic (e.g., Tabu Search), others are stochastic (e.g., Simulated Annealing). Some generate complete solutions by *modifying* complete solutions (e.g., EAs), while some others *construct* new solutions at every iteration (e.g., ACO). Many of these metaheuristics offer unique features, but even within a single approach, there are many variants which incorporate different representation of solutions and different modification or construction techniques for new solutions.

3 What are "Evolutionary Algorithms"?

So, what are EAs? Perhaps the best place to start in answering the question is to note that there are at least two possible interpretations of the term *evolution*. It is frequently used in a very general sense to describe something that changes incrementally over time, such as the software requirements for a payroll accounting system. The second meaning is its narrower use in biology, where it describes an evolutionary system that changes from generation to generation via reproductive variation and selection. It is this Darwinian notion of evolutionary change that has been the core idea in EAs.

3.1 Principles Inspired by Nature

From a conventional point of view, an EA is an algorithm that simulates – at some level of abstraction – a Darwinian evolutionary system. To be more specific, a standard EA includes:

1. One or more populations of individuals competing for limited resources.

2. These populations change dynamically due to the birth and death of individuals.
3. A notion of fitness which reflects the ability of an individual to survive and reproduce.
4. A notion of variational reproduction: offspring closely resemble their parents, but are not identical.

In a nutshell, the Darwinian principles of evolution suggest that, on average, species improve their fitness over generations (i. e., their capability of adapting to the environment). A simulation of the evolution based on a set of candidate solutions whose fitness is properly correlated to the objective function to optimize will, on average, lead to an improvement of their fitness and thus steer the simulated population towards the solution.

3.2 The Basic Cycle of EAs

In the following, we try to introduce a very simple EA consisting of a single population of individuals exist in an environment that presents a time-invariant notion of fitness. We will do this from a general perspective, comprising most of the conventional EAs.

Like in nature, an individual may have two different representations: the data structure which is processed by the genetic search procedures and the format in which it is assessed by the environment (and finally handed to the human operator). Like in biology, in the context of EAs, the former representation is referred to as *genotype* and the latter as *phenotype.* EAs usually proceed in principle according to the scheme illustrated in Figure 1. Its steps can be described as follows:

1. In the first generation, a population of $n > 0$ individuals is created. Usually, these individuals have random genotypes but sometimes, the initial population is *seeded* with good candidate solutions either previously known or created according to some other methods.
2. The genotypes, i. e., the points in the search space, are then translated to phenotypes. In the case that search operations directly work on the solution data structures, this genotype-phenotype mapping is the identity mapping.
3. The values of the objective functions are then evaluated for each candidate solution in the population. This evaluation may incorporate complicated simulations and calculations.
4. With the objective functions, the utility of different features of the candidate solutions has been determined. If there is more than one objective function, constraint, or other utility measure, then a scalar fitness value is assigned to each of them.

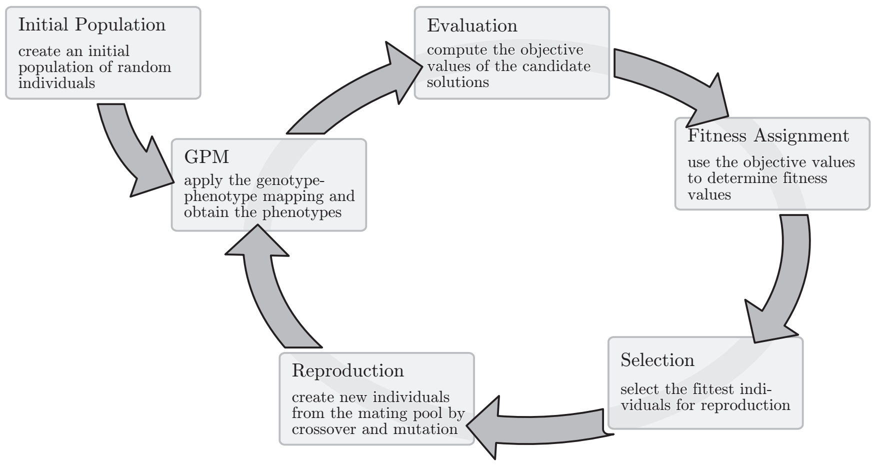
Initial Population
create an initial population of random individuals
GPM
apply the genotype-phenotype mapping and obtain the phenotypes
Evaluation
compute the objective values of the candidate solutions
Fitness Assignment
use the objective values to determine fitness values
Selection
select the fittest individuals for reproduction
Reproduction
create new individuals from the mating pool by crossover and mutation

Figure 1: The basic cycle of EAs.

5. A subsequent selection process filters out the candidate solutions with poor fitness and allows those with good fitness to enter the mating pool with a higher probability.
6. In the reproduction phase, offspring are derived from the genotypes of the selected individuals by applying the search operations (which are called *reproduction operations* in the context of EAs). There are usually two different reproduction operations: mutation, which modifies one genotype, and crossover, which combines two genotypes to a new one. Whether the whole population is replaced by the offspring or whether they are integrated into the population as well as which individuals to recombine with each other depends on the applied population handling strategy.
7. If the termination criterion is met, the evolution stops here. Otherwise, the evolutionary cycle continues with the next generation at point 2.

Of course, such an algorithm description is too abstract to be executed directly.

3.3 Do all EAs fit to the Basic Cycle?

According to our discussion so far, a simple answer to the question "What are EAs?" would be that EAs are those based on the concepts gleaned from natural evolution and which roughly adhere to the principles and the basic cycle introduced in the previous sections. From a high-level perspective, however, the definition is not so clear.

When solving a new challenging problem, often a new optimization method is designed. It is necessary to specify how the individuals in the population represent the problem solutions, how the fitness is calculated, how parents are selected, how offspring are produced, and how individuals are selected for removal from the population (i. e., to die[1]). Each of these decisions results in an EA variant with different computational properties.

Will these design decisions result in an EA? Before you answer, let us recall that the $(1+1)$ EA does not require a population of solutions but processes just one individual (and is comparing it with its only offspring). Many "Evolutionary Algorithms" assume deterministic selection methods, many "Evolutionary Algorithms" take advantage of smart initialization and problem-specific operators. Some "Evolutionary Algorithms" have been extended with memory structures (e.g., when they operate in dynamic environments) or by a parameter called *temperature* (to control mutation rates). The list could go on.

While there is a well-researched set of "default" EAs which we will introduce in the next section, for many real-world applications it is necessary to derive new, specialized approaches. Examples for this can be found in [9–11] as well as the collection in this book [12].

3.4 Conventional EAs

Historically, computer scientists and engineers had started to consider drawing inspiration from evolutionary principles for solving optimization problems as early as the 1950s (see [13, 14], and [15]). In the 1960s and 1970s, three research lines were developed in parallel [16]: EP [17], ES [18], and GAs [19]. De Jong's PhD thesis [20] further increased the interest in this field, and his PhD student Grefenstette, in turn, started the *International Conference on Genetic Algorithms and their Applications* (ICGA) [21] in 1985. At the 1991 ICGA [22], the three original research streams came together, with Hans-Paul Schwefel presenting the ES. At the same venue, Koza [23] introduced the new concept of Standard GP and Zbigniew Michalewicz outlined the concepts of different data structures which can undergo the evolutionary process in the so-called *Evolutionary Programs* [24]. This was considerably the first time all major areas of Evolutionary Computation were represented at once. As a result, the *Evolutionary Computation* Journal by MIT Press was established, later followed by the *IEEE Transactions on Evolutionary Computation.* The idea of unifying concepts, such as "Evolutionary Algorithms" (or the more general idea of Evolutionary Computation [16]), was then born. Thereafter,

[1] The "death" of an individual, candidate solution, or agent in terms of metaheuristic optimization means that it is removed from the set of elements under investigation and deleted from memory, possibly due to being replaced by a better element.

the first *IEEE Congress on Evolutionary Computation* [25] was initiated in 1994.

From the 1990s onwards, many new ideas have been introduced. One of the most important developments is the discovery that EAs are especially suitable to solve problems with multiple, possibly conflicting optimization criteria – the *Multi-Objective Evolutionary Algorithms* (MOEAs) [26, 27]. Today, the second, improved versions of NSGA [28, 29] and SPEA [30, 31] may be the most popular members of this MOEA family.

There is also growing interest in co-evolutionary EAs, originally introduced by Hillis [32] back in 1989. Potter and De Jong [33, 34] developed *cooperative co-evolution*, which is now regarded as one of the key approaches for tackling large-scale optimization problems because it provides a viable way to decompose the problems and co-evolve solutions for the problem parts which together make up a solution for the original task [35]. Other parallel developments include the works of Grefenstette [36], Deb and Goldberg [37] as well as De Jong [38] who considered the issues *deception*.

Books such as [24, 26, 39] and [40] have always played a major role in opening the field of Evolutionary Computation to a wide audience, with the *Handbook of Evolutionary Computation* [41] one of the most prominent examples.

3.4.1 Genetic Algorithms

GAs are the original prototype of EAs. Here, the genotypes of the search space are strings of primitive elements (usually all of the same type) such as bits, integers or real numbers. Because of the simple structure of the search space of GAs, a genotype-phenotype mapping is often used to translate the genotypes to candidate solutions [19, 39, 42, 43].

The single elements of the string genotypes in GAs are called *genes*. GAs usually apply both the mutation and crossover operators. The mutation operator modifies one or multiple genes whereas the crossover operator takes two genotypes and combines them to form a new one, either by merging or by exchanging the values of the genes. The most common reproduction operations used in GAs, single-point mutation and single-point crossover, are sketched in Figure 2 [39, 44].

3.4.2 Genetic Programming

The term *Genetic Programming* [23, 42, 45] has two possible meanings. First, it can be viewed as a set of EAs that breed programs, algorithms, and similar constructs. Second, it is also often used to subsume EAs that have tree data structures as genotypes. Tree-based GP, usually referred to as the Standard GP, is the most widespread GP variant both for historical reasons and

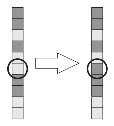
Fig. 2.a: Single-gene mutation.

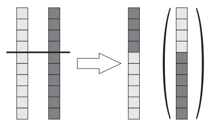
Fig. 2.b: Single-point Crossover (SPX).

Figure 2: Mutation and Crossover in GAs.

because of its efficiency in many problem domains. Here, the genotypes are tree data structures. Generally, a tree can represent a rule set [11, 46, 47], a mathematical expression [23], a decision tree [47, 48], or even the blueprint of an electrical circuit [49].

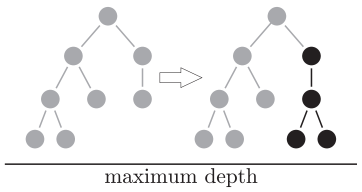

Fig. 3.a: Sub-tree replacement mutation.

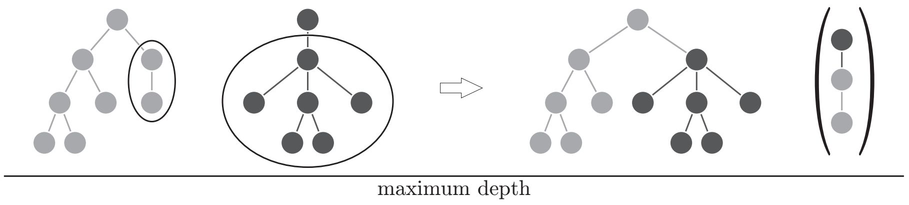

Fig. 3.b: Subtree exchange crossover.

Figure 3: Mutation and Recombination in GP.

Of course, mutation and crossover operators as used in GAs cannot be applied to tree data structures. Instead, new operations have been developed, such as the sub-tree replacement mutation which replaces a sub-tree of a genotype with a randomly created one and sub-tree exchange crossover which exchanges two sub-trees between two parental genotypes, as sketched in Figure 3.

3.4.3 Evolution Strategies

ES, introduced by Rechenberg [18, 50, 51] and Schwefel [52, 53, 54], is a heuristic optimization technique based on the ideas of adaptation and evolution – a special form of EAs [18, 42, 50, 51, 55–58]. The search space of today's ES usually consists of vectors from the $\mathbb{R}^n$, but bit strings or integer strings are common as well [56]. Mutation and selection are the primary reproduction operators and recombination is used less often in ES. Most often, normal distributed random numbers are used for mutation. The parameter of the mutation is the standard deviation of these random numbers. ES may either:

1. maintain a single standard deviation parameter and use identical normal distributions for generating the random numbers added to each element of the solution vectors,
2. use a separate standard deviation (from a standard deviation vector) for each element of the genotypes, i. e., create random numbers from different normal distributions for mutations in order to facilitate different strengths and resolutions of the decision variables, or
3. use a complete covariance matrix for creating random vectors distributed in a hyperellipse and thus also taking into account binary interactions between the elements of the solution vectors.

The standard deviations are governed by self-adaptation [59–61] and may result from a stochastic analysis of the elements in the population [62–65]. They are often treated as *endogenous* strategy parameters which can directly be stored in the individual records and evolve along with them [56].

3.4.4 Evolutionary Programming

EP is less precisely defined as the other conventional EAs. There is though a semantic difference: while single individuals of a species are the biological metaphor for candidate solutions in other EAs, in EP a candidate solution is thought of as a species itself. Hence, mutation and selection are the only operators used in EP and recombination is usually not applied. The selection scheme utilized in EP is normally quite similar to the $(\mu + \lambda)$ method in ES.

EP was pioneered by Fogel [66] in his PhD thesis back in 1964. Fogel et al. [17] experimented with the evolution of finite state machines as predictors for data streams [67]. One of the most advanced EP algorithms for numerical optimization today has been developed by Yao et al. [68].

4 Memetic Computing

Memetic Computing is a growing area in computational intelligence closely related to EAs. During the creation of the initial population in an EA, a set of candidate solutions is generated, usually randomly within the decision space. Other sampling systems that include a certain degree of determinism for selecting the initial set of solutions are also widely used. The latter, usually known as intelligent initial sampling, is often considered as a memetic component within an EA framework [69].

4.1 MAs as an Extension of EAs

The main idea in 1980s and 1990s was to propose EAs with superior performance with respect to all the other algorithms present in the literature. This approach is visible in many famous texts and papers published in those years (see Section 3.4). After the publication of the No Free Lunch Theorem (NFLT) [70], however, researchers in the field have to dramatically change their view about the subject. The NFLT mathematically proves that the average performance of any pair of algorithms A and B across all possible problems with finite search spaces is identical. Thus, if an algorithm performs well on a certain class of problems, then it necessarily pays for that with degraded performance on other sets of problems. The concept that there is no universal optimizer has a significant impact on the scientific community.[2] In light of increasing interest in general-purpose optimization algorithms, it has become important to understand the relationship between the performance of an algorithm A and a given optimization problem f. The problem hence becomes the starting point for building up a suitable algorithm.

In this context, the term *Memetic Algorithms* was coined, representing an efficient alternative (or maybe a modification) of EAs. This term was first introduced in [72] with reference to an algorithm proposed in [73, 74] to indicate an approach that integrates a local search operator within an evolutionary structure. The metaphor of the term "memetic" was inspired by modern philosophy, more specifically by the meme's theory of Richard Dawkins [75]. The meme is an idea, a "unit of cultural transmission", the basic unit of knowledge. Although in Dawkins' studies the focus was to prove that evolution was based on the individual choices rather than collective choices (the selfish gene), in Computer Science another concept has been taken and adapted to computational problem-solving. By interpreting Dawkins' philosophy, it can be deduced that the collective culture is the result of an evolutionary process where ideas (memes) interact and diffuse over individuals modifying

[2] Note, however, that it is possible to find algorithms which are best over large sets of (practically-relevant) problems; see [71].

and getting enriched. Transferring this to the computing environment, different search operators (e.g., evolutionary framework and local search) compete and cooperate as different memes and process the solutions, by means of their harmonic interaction, towards the detection of the global optimum.

A definition which characterizes the structural features of MAs has been given in [69]. In general, an MA is a population-based hybrid optimization algorithm composed of an evolutionary framework and a list of local search algorithms activated within the generation cycle of the framework. In other words, MAs can be considered as specific hybrid algorithms which combine an EA framework and local search for enhancing the quality of some solutions of the population during the EA generation cycle. The sense of MAs is to compensate, for some specific problems, the limitations of EAs. As with all other metaheuristics, the functioning of an EA is due to the proper balance between exploration and exploitation. The generally optimal balance, in accordance with the NFLT, does not exist but it should be found for each fitness landscape. In addition, MAs contain multiple search components which can explore the fitness landscape from complementary perspectives and mitigate the typical undesired effects of stagnation and premature convergence.

Obviously, in MAs the coordination between the EA framework and local search operators can be hard to design. For this reason, a lot of research studies on MAs have been performed by paying great attention to the coordination logic of the various search operators. By updating the classification given in [76], MAs can be subdivided as: 1) Adaptive Hyper-Heuristic, see e.g., [77–79] and [80], where the coordination of the memes is performed by means of heuristic rules; 2) Self-Adaptive and Co-Evolutionary, see e.g., [81, 82] and [83], where the memes, either directly encoded within the candidate solutions or evolving in parallel to them, take part in the evolution and undergo recombination and selection in order to select the most promising operators; 3) Meta-Lamarckian Learning, see e.g., [84–86] and [87], where the success of the memes biases their activation probability, thus performing an on-line algorithmic design which can flexibly adapt to various optimization problems; 4) Diversity-Adaptive, see e.g., [88–93] and [94], where a measure of the diversity is used to select and activate the most appropriate memes. In addition, it is worth to mention about the Baldwinian systems, i. e., those MAs that do not modify the solutions after the employment of local search, see [95] and [96]. The latter are basically EAs where the selection process is influenced by the search potential of each solution.

4.2 Can all MAs be considered EAs?

Generally speaking, MAs are population-based algorithms that evolve solutions under the same rules/logic as conventional EAs while additionally applying local search. In this sense, if the local search algorithms are to be considered as special operators, e.g., a hill-climb is seen as a mutation, then MAs can be considered as a subset of EAs. On the other hand, MAs can be considered as EAs that allow plenty of unconventional operators. To this end, MAs can be seen as an extension of EAs.

Regardless of the labeling, it is important to note that all these optimization algorithms are de facto the combination of two kinds of operators, i. e., search and selection, respectively. In conventional EAs, the search is performed by crossover and mutation operators, which are also known as variation operators, while the selection is included into the parent and survivor selection phases. Similarly, the combination of these two kinds of operators can be spotted within an MA by analyzing its evolutionary framework and each of its local search components. In this context, the more modern (and at the same time primitive) concept of Memetic Computing has been recently defined in a structured and systematic way. Specifically, Memetic Computing is a broad subject which studies complex and dynamic computing structures composed of interacting operators (memes) whose evolution dynamics is inspired by the diffusion of ideas.

Research in evolutionary optimization has always been closely tied to self-adaptation, i. e., the development of approaches which can adapt their parameters to the optimization problem at hand. An important research goal in this area would thus be to develop an intelligent unit which can choose, during the optimization run, the most suitable combination of operators for a given problem. Since a high degree of flexibility is necessary for solving a wide range of problems, Memetic Computing is strictly connected to the concept of modularity and an evolutionary structure that can be seen as a collection of interactive modules whose interaction, in an evolutionary sense, leads to the generation of the solution of the problem.

Concerning the structure of Memetic Computing approaches, there are two philosophies. On one hand, Memetic Computing can be seen as a broad subject which includes various kinds of algorithms. In order to solve optimization problems, a structure consisting of multiple operators, each of which performing a simple action, must be composed. Depending on the underlying algorithms used, a Memetic Computing approach may or may not be an EA (or its extension).

On the other hand, Memetic Computing can be considered from a bottom-up perspective. Here, the optimization algorithm would start as a blank slate to which components are added one by one. One interesting stream of research is the automatic design of algorithmic structures. Here, three aspects should be considered: 1) the memes should be simple operators, 2) the role and effect of each meme should be clearly understood so that this knowledge can be encoded and used by the automated designer in a flexible way, and 3) the algorithmic structure should be built up from scratch by means of the aforementioned bottom-up strategy which aims at including only the necessary memes and the simplest possible coordination rules.

5 Swarm Intelligence

Swarm Intelligence, another area closely related to EAs, is concerned with the design of algorithms or distributed problem-solving devices inspired by the collective behavior of social insects or animals. Two of the most popular Swarm Intelligence algorithms are PSO and ACO. Other representative examples include those for routing in communication networks based on the foraging behavior of bees [97], and those for dynamic task allocation inspired by the behavior of wasp colonies [98].

The natural role model of Particle Swarm Optimization, originally proposed by Eberhart and Kennedy [99–101], is the behavior of biological social systems like flocks of birds or schools of fish. PSO simulates a swarm of particles (individuals) in an n-dimensional search space, where each particle has its own position and velocity [102–104]. The velocity vector of an individual determines in which direction the search will continue and if it has an explorative (high velocity) or an exploitative (low velocity) character. This velocity vector represents an endogenous parameter – while the endogenous information in ES is used for an undirected mutation, the velocity in PSO is used to perform a directed modification of the genotypes (particles' positions).

ACO, developed by Dorigo et al. [105], is an optimization technique inspired by the capability of ant colonies to find short paths between encountered food sources and their ant hill [106, 107]. This capability is a result of the collective behavior of locally interacting ants. Here, the ants communicate indirectly via chemical pheromone trails which they deposit on the ground. This behavior can be simulated as a multi-agent system using a pheromone model in order to construct new solutions in each iteration.

In the following sections, we will take a closer look at PSO and ACO, and discuss their similarities and differences with EAs.

5.1 Particle Swarm Optimization

In what we refer to as the Standard PSO (SPSO), whose code is freely available on the Particle Swarm Central `http://www.particleswarm.info`, there is typically a unique swarm of agents, called particles, in which each particle P_i is defined as

$$P_i = (p_i, v_i, b_i) \tag{1}$$

where p_i is the position, v_i the velocity (more precisely the displacement), and b_i the best position ever found so far by the particle. Each particle is informed by a set $\mathcal{N} = \{P_j\}$ of other particles called "neighborhood". The metaphor is that each particle "moves", and the process at each time step can be described as follows:

1. each particle asks its neighbors, and chooses the best one (the one that has the best b_j)
2. it computes a new velocity v'_i by taking into account v_i, p_i, b_j; the precise formula is not important here, as it differs from version to version of SPSO (e.g., compare SPSO 2007 and SPSO 2011) – the most important key feature is that it contains some randomness and that its general form is

$$v'_i = a(v_i) + b(p_i) + c(b_i) + kd(b_j) \tag{2}$$

3. each particle "moves", by computing the new position as $p'_i = p_i + v'_i$
4. if the new position is better, then b_i is updated, by $b'_i = p'_i$

There also exists another possible, formally equivalent but more flexible, point of view. That is, one may consider three kinds of agents:

1. position agents p_i
2. velocity agents v_i
3. memory agents m_i

Here, m_i is in fact the b_i of the previous process description. Now, there are three populations, $\mathcal{P} = \{p_i\}$, $\mathcal{V} = \{v_i\}$, and $\mathcal{M} = \{m_i\}$. Each v_i has a "neighbourhood" of informants, which is a subset of $\mathcal{M}$, and the process at each time step can be described as follows:

1. the velocity agent v_i updates its components, thanks to the function a of Equation 2
2. then it combines them with some information coming from p_i, m_i, and from its best informant m_j, thanks to the functions b, c, d, in order to define a new velocity v'_i (note that the order of the operations may be equivalently 2 then 1, as Equation 2 is commutative)
3. a new position agent p'_i is generated, by $p'_i = p_i + v'_i$
4. if the new agent is better than m_i, the agent m_i "dies", and is replaced by a better one, by using the formula $m'_i = p'_i$
5. p_i "dies"

Mathematically speaking, the behavior here is exactly the same as the previously described one, but as the metaphor is different, it is now easier to answer some of the relevant questions we want to address in this chapter.

5.2 Is PSO an EA?

A classical definition of an EA, given in Section 3, states that it uses mechanisms such as reproduction, mutation, recombination, and selection. Quite often, it is also added that some of these mechanisms have to make use of randomness. It is clear that randomness is used in all stochastic algorithms, including PSO, so we will not proceed on this point any further. In the following, let us consider, one by one, the four mechanisms of EAs – mutation, recombination, reproduction and selection – from a PSO point of view.

In molecular biology and genetics, mutations are changes in a genomic sequence. In a D-dimensional search space, a velocity agent can be written $v_i = (v_{i,1}, \cdots, v_{i,D})$. It is worth noting that, on a digital computer the search space is necessarily finite and discrete (even if the number of possible $v_{i,k}$ values is huge). Therefore, v_i can be seen as a "genomic sequence". According to point 1 in the algorithm description above, the velocity agent can be said to be "mutated". Here, however, the mutation rate is almost always equal to 100% (all components are modified). Also, mutation occurs before the reproduction.

Genetic recombination is a process by which a molecule of nucleic acid is broken and then joined to a different one. Point 2 in the PSO algorithm description can be seen as a recombination of the genomic sequences of three agents.

Reproduction (or procreation) is the biological process by which new "offspring" individual organisms are produced from their "parents". According to point 3 of the PSO description, a part of the process can be symbolically described by

$$(p_i, v_i') \Rightarrow p_i' \tag{3}$$

which can be interpreted as procreation with two "parents".

Natural selection is the process by which traits become more or less common in a population due to consistent effects upon the survival or reproduction of their bearers. We can see that point 4 of the PSO algorithm is a selection mechanism: the agent m_i may die or survive, according to its "quality". Also, it can be proved (see more comments about this in [108]) that there is always convergence. It means that the m_i agents (and also the p_i agents) become more and more similar. In the optimisation context, this phenomenon is called stagnation, and is not very desirable. In other words, there is a kind of selection, but it has to be carefully controlled for good performance.

So, is PSO an EA or not? The answer to the question itself is not really interesting. It is just a matter of classification. By studying the question, however, a new point of view on PSO could be defined, which may suggest some fruitful variants (not studied in detail here). For instance:

1. The "mutation" rate may be smaller than 100%. In that case, not all velocity components are modified. In particular, if it is zero, there is no "generation", and, as the position agent "dies", the swarm size decreases.
2. Instead of being informed by always the same memory agent m_i, the velocity agent v_i may be informed by some others. The "combination" may make use of more than two memory agents, or even all (for this case, see [109]). Actually, we may also define a population $\mathcal{L}$ of "link agents". The existence of a (i, j) agent means there is an information link between

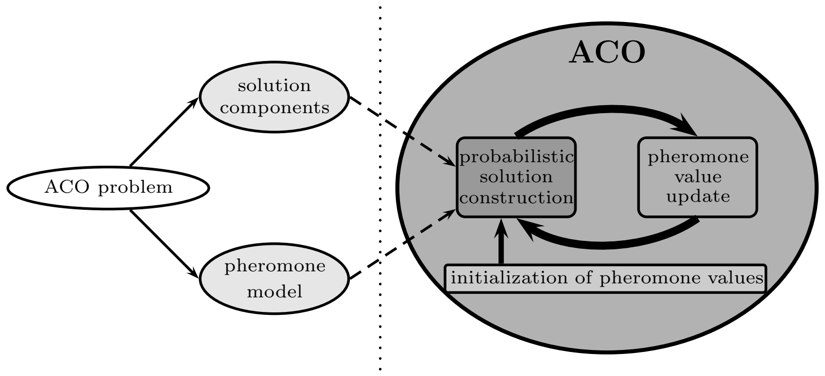

Figure 4: A schematic view of ACO algorithms.

the velocity agent v_i and the memory agent m_j. It is even possible to design an algorithm that works by co-evolution of the four populations $\mathcal{P}$, $\mathcal{V}$, $\mathcal{M}$, and $\mathcal{L}$.

3. The position agent may not die. In that case, and if the velocity agent is not null, the swarm size increases.

... and so on.

5.3 Ant Colony Optimization

Like EAs, ACO algorithms [110, 111] are bio-inspired techniques for optimization. A schematic view of ACO algorithms is shown in Figure 4. They are based on a so-called pheromone model, which is a set of numerical values that are associated to opportunely defined solution components. In the case of the well-known TSP, for example, the edges of the underlying graph are the solution components. The pheromone model is used to generate – at each iteration – a fixed number of solutions to the considered problem. Again considering the case of the TSP, edges with a high pheromone value have a greater chance to be chosen during the solution construction process. In this way, the pheromone model – together with the mechanism for constructing solutions – implies a parameterized probability distribution over the search space. In general, the ACO approach attempts to solve an optimization problem by iterating the following two steps:

1. candidate solutions are constructed in a probabilistic way by using the pheromone model;
2. the candidate solutions are used to modify the pheromone values in a way that is deemed to bias future sampling toward high quality solutions.

In other words, the pheromone update aims to concentrate the search in regions of the search space containing high quality solutions. In particular, the reinforcement of solution components depending on the solution quality is an important ingredient of ACO algorithms. It implicitly assumes that good solutions consist of good solution components. To learn which components contribute to good solutions can help assembling them into better solutions.

5.4 Is ACO an EA?

While there are some similarities between EAs and ACO algorithms, there also exist some fundamental differences. Concerning the similarities, ACO algorithms are – just like EAs – population-based techniques. At each iteration a number of new solutions is generated. In both cases new solutions are generated based on search experience. However, while most EAs store their search experience in the explicit form of a population of solutions, ACO algorithms store their search experience in the values of the pheromone model. Accordingly, there are also differences in updating the stored information. While standard EAs perform an explicit update of the population – that is, at each iteration some solutions are replaced by new ones – ACO algorithms use some of the generated solutions for making an update of the pheromone values.

Despite the differences, ACO algorithms and certain types of EAs can be studied under a common framework known as *model-based search* [112]. Apart from ACO algorithms, this framework also covers stochastic gradient ascent, the cross-entropy method, and EAs that can be labeled as Estimation of Distribution Algorithms (EDAs) [113]. According to [112], *"in model-based search algorithms, candidate solutions are generated using a parametrized probabilistic model that is updated using the previously seen solutions in such a way that the search will concentrate in the regions containing high quality solutions."*

The development of EDAs was initiated by mainly two observations. The first one concerns the fact that standard crossover operators were often observed to destroy good *building blocks*, i.e., partial solutions that are present in most, if not all, high quality solutions. The second observation is the one of *genetic drift*, i.e., a loss of diversity in the population due to its finite size. As a result of genetic drift, EAs may prematurely converge to sub-optimal solutions. One of the earliest EDAs is *Population-Based Incremental Learning* (PBIL) [114], developed with the idea of removing the genetics from the standard GA. In fact, for problems with independent decision variables, PBIL using only the best solution of each iteration for the update is equivalent to a specific version of ACO known as the *hyper-cube framework* with iteration-best update [115].

Summarizing, while ACO algorithms may be seen as model-based search algorithms, just like some EAs, ACO algorithms should not be labeled as "Evolutionary Algorithms".

6 Concluding Remarks

In this chapter, we have discussed the term "Evolutionary Algorithms" from various different perspectives. As seen in Section 3, there are at least two ways to define EAs. Traditionally, "Evolutionary Algorithms" is considered as a term identifying a set of algorithms (e.g., GAs, GP, ES and EP) which work according to the same basic cycle. Today, even these terms became mere names for large algorithm families which consist of many different sub-algorithms. The justification for such a variety of algorithms has been pointed out in Section 4.1: the NFLT which signifies that there may be an algorithm which is best for a certain family of optimization problems, but not for all possible ones. The variety of "Evolutionary Algorithms" has led to the controversy about what is an EA and what it is not.

One of the factors contributing to this situation is that there exist many new metaheuristics that share the characteristic traits of EAs but differ significantly in their semantics. Hybrid EAs incorporating local search algorithms and other Memetic Computing approaches, for instance, possess a different algorithmic structure. EDAs are population-based randomized algorithms and involve selection and possibly mutation – but are not related to any process in nature.

Another possible factor is that researchers nowadays tend to pay more efforts into defining common frameworks which can unite different algorithms, such as the already mentioned work in [112] or the recent framework proposed in [116] that unites both the traditional EAs and EDAs. Generally speaking, metaheuristics can be viewed as the combination of components for search and selection, i. e., a set of operations for generating one or more trial solutions and/or a set of operations to perform the selection of the solution and thus of the search directions.

Furthermore, the research communities working on particular algorithms pursue a process of generalization and formalization during which more similarities between formerly distinct approaches are discovered. These processes make it easier to construct versatile algorithms and also provide the chance of obtaining more generally applicable theoretical results.

Besides these reasons, there is the basic fact that researchers themselves are the ones who decide the name of their algorithms. It may indeed be argued whether a $(1 + 1)$ ES is actually an EA or just a Hill Climbing method, or whether those very first MAs were special EAs or not. Approaching this issue from the opposite direction, it is indeed possible to develop algorithms which improve a set of solutions with a process of choosing the best ones and slightly modifying them in an iterative way, e.g., by using unary and binary search operations, without utilizing any inspiration from nature. Would the term "Evolutionary Algorithm" appropriate for such an algorithm?

The meaning of the term is thus subject to interpretation, and we put three other metaheuristics, the MA, PSO and ACO, into the context of this controversy. The sections on PSO and ACO in particular symbolize very well how different researchers may either tend to generalize an algorithm's definition to make it more compatible to the evolutionary framework or may emphasize more on its individual features in favor of more distinct semantics.

A simple strategy to avoid ambiguity would be to use terms like *Nature-Inspired Algorithms* or *Evolutionary Computation Techniques* for general methods inspired by nature or evolution and to preserve the term "Evolutionary Algorithm" for GAs, GP, ES, EP and, to a lesser extent, Differential Evolution and EDAs.

Another idea would be to more strictly divide the theoretical algorithm structure from its inspirational roots and history, i. e., to totally abandon terms such as "genetics", "evolutionary", "mutation" or "crossover" from the naming conventions. Of course, this would probably not happen since these terms have already entered the folklore. However, more frequently using words such as "unary search operation" instead of "mutation" or "candidate solution" instead of "phenotype" in favor of a clearer ontology would lead to more precise definitions, inspire more rigorous analyses, and may reduce the quack aura sometimes wrongly attributed by industrial engineers to the so-called Evolutionary Computation techniques.

Yet, it is likely that "Evolutionary Algorithms" would suffer the same fate as the term "agent" and blur into a state of, on one hand, almost universal applicability and, on the other hand, lesser semantic value. Then again, this does not necessarily need to be bad – since it may open the door for even more cross-discipline interaction and cross-fertilization of ideas, as can be observed in the agent community too during the past 20 years.

Acknowledgement. Ferrante Neri's work is supported by Academy of Finland, Akatemiatutkija 130600, Algorithmic Design Issues in Memetic Computing. Christian Blum acknowledges support from grant TIN2007-66523 (FORMLISM) and from the Ram n y Cajal program of the Spanish Government. Thomas Weise is supported by the Chinese Academy of Sciences (CAS) Fellowship for Young International Scientists 2011 and the China Postdoctoral Science Foundation Grant Number 20100470843.

References

1. Christian Blum, Raymond Chiong, Maurice Clerc, Kenneth Alan De Jong, Zbigniew Michalewicz, Ferrante Neri, and Thomas Weise. Evolutionary optimization. In Raymond Chiong, Thomas Weise, and Zbigniew Michalewicz, editors, *Variants of Evo-*

lutionary Algorithms for Real-World Applications, chapter 1, pages 1–29. Springer, Berlin Heidelberg, 2011. ISBN 978-3-642-23423-1. 10.1007/978-3-642-23424-8_1.

2. Zbigniew Michalewicz and David B. Fogel. *How to Solve It: Modern Heuristics.* Springer-Verlag: Berlin/Heidelberg, 2nd edition, 2004. ISBN 3-540-22494-7.
3. Thomas Weise, Michael Zapf, Raymond Chiong, and Antonio Jesús Nebro Urbaneja. Why Is Optimization Difficult? In Raymond Chiong, editor, *Nature-Inspired Algorithms for Optimisation*, volume 193/2009 of *Studies in Computational Intelligence*, chapter 1, pages 1–50. Springer-Verlag: Berlin/Heidelberg, 2009. 10.1007/978-3-642-00267-0_1. URL `http://www.it-weise.de/documents/files/WZCN2009WIOD.pdf`.
4. Fred Glover and Gary A. Kochenberger, editors. *Handbook of Metaheuristics*, volume 57 of *International Series in Operations Research & Management Science.* Kluwer Academic Publishers: Norwell, MA, USA and Springer Netherlands: Dordrecht, Netherlands, 2003. ISBN 0-306-48056-5 and 1-4020-7263-5. 10.1007/b101874. Series Editor Frederick S. Hillier.
5. Teofilo F. Gonzalez, editor. *Handbook of Approximation Algorithms and Metaheuristics.* Chapmann & Hall/CRC Computer and Information Science Series. Chapman & Hall/CRC: Boca Raton, FL, USA, 2007. ISBN 1584885505.
6. Christian Blum and Andrea Roli. Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison. *ACM Computing Surveys (CSUR)*, 35(3):268–308, September 2003. 10.1145/937503.937505. URL `http://iridia.ulb.ac.be/~meta/newsite/downloads/ACSUR-blum-roli.pdf`.
7. Thomas Weise. *Global Optimization Algorithms – Theory and Application.* it-weise.de (self-published): Germany, 2009. URL `http://www.it-weise.de/projects/book.pdf`.
8. Raymond Chiong, editor. *Nature-Inspired Algorithms for Optimisation*, volume 193/2009 of *Studies in Computational Intelligence.* Springer-Verlag: Berlin/Heidelberg, April 30, 2009. ISBN 3-642-00266-8 and 3-642-00267-6. 10.1007/978-3-642-00267-0.
9. Thomas Weise, Alexander Podlich, and Christian Gorldt. Solving Real-World Vehicle Routing Problems with Evolutionary Algorithms. In Raymond Chiong and Sandeep Dhakal, editors, *Natural Intelligence for Scheduling, Planning and Packing Problems*, volume 250 of *Studies in Computational Intelligence*, chapter 2, pages 29–53. Springer-Verlag: Berlin/Heidelberg, 2009. 10.1007/978-3-642-04039-9_2. URL `http://www.it-weise.de/documents/files/WPG2009SRWVRPWEA.pdf`.
10. Thomas Weise, Alexander Podlich, Kai Reinhard, Christian Gorldt, and Kurt Geihs. Evolutionary Freight Transportation Planning. In Mario Giacobini, Penousal Machado, Anthony Brabazon, Jon McCormack, Stefano Cagnoni, Michael O'Neill, Gianni A. Di Caro, Ferrante Neri, Anikó Ekárt, Mike Preuß, Anna Isabel Esparcia-Alcázar, Franz Rothlauf, Muddassar Farooq, Ernesto Tarantino, Andreas Fink, and Shengxiang Yang, editors, *Applications of Evolutionary Computing – Proceedings of EvoWorkshops 2009: EvoCOMNET, EvoENVIRONMENT, EvoFIN, EvoGAMES, EvoHOT, EvoIASP, EvoINTERACTION, EvoMUSART, EvoNUM, EvoSTOC, EvoTRANSLOG (EvoWorkshops'09)*, volume 5484/2009 of *Theoretical Computer Science and General Issues (SL 1), Lecture Notes in Computer Science (LNCS)*, pages 768–777. Springer-Verlag GmbH: Berlin, Germany, 2009. 10.1007/978-3-642-01129-0_87. URL `http://www.it-weise.de/documents/files/WPRGG2009EFTP.pdf`.
11. Thomas Weise and Ke Tang. Evolving Distributed Algorithms with Genetic Programming. *IEEE Transactions on Evolutionary Computation (IEEE-EC)*, 16(2), 2011. 10.1109/TEVC.2011.2112666.
12. Raymond Chiong, Thomas Weise, and Zbigniew Michalewicz, editors. *Variants of Evolutionary Algorithms for Real-World Applications.* Springer-Verlag: Berlin/Heidelberg, 2011. 10.1007/978-3-642-23424-8.

13. Nils Aaall Barricelli. Esempi Numerici di Processi di Evoluzione. *Methodos*, 6(21–22): 45–68, 1954.
14. Nils Aaall Barricelli. Symbiogenetic Evolution Processes Realized by Artificial Methods. *Methodos*, 9(35–36):143–182, 1957.
15. Alex S. Fraser. Simulation of Genetic Systems by Automatic Digital Computers. I. Introduction. *Australian Journal of Biological Science (AJBS)*, 10:484–491, 1957.
16. Zbigniew Michalewicz. A Perspective on Evolutionary Computation. In Xin Yao, editor, *Progress in Evolutionary Computation, AI'93 (Melbourne, Victoria, Australia, 1993-11-16) and AI'94 Workshops (Armidale, NSW, Australia, 1994-11-22/23) on Evolutionary Computation, Selected Papers*, volume 956/1995 of *Lecture Notes in Computer Science (LNCS)*, pages 73–89. Springer-Verlag GmbH: Berlin, Germany, 1993. 10.1007/3-540-60154-6_49.
17. Lawrence Jerome Fogel, Alvin J. Owens, and Michael J. Walsh. *Artificial Intelligence through Simulated Evolution*. John Wiley & Sons Ltd.: New York, NY, USA, 1966. ISBN 0471265160.
18. Ingo Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. PhD thesis, Technische Universität Berlin: Berlin, Germany, 1971.
19. John Henry Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. University of Michigan Press: Ann Arbor, MI, USA, 1975. ISBN 0-472-08460-7.
20. Kenneth Alan De Jong. *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*. PhD thesis, University of Michigan: Ann Arbor, MI, USA, August 1975. URL `http://cs.gmu.edu/~eclab/kdj_thesis.html`.
21. John J. Grefenstette, editor. *Proceedings of the 1st International Conference on Genetic Algorithms and their Applications (ICGA'85)*, June 24–26, 1985. Lawrence Erlbaum Associates: Hillsdale, NJ, USA. ISBN 0-8058-0426-9.
22. Richard K. Belew and Lashon Bernard Booker, editors. *Proceedings of the Fourth International Conference on Genetic Algorithms (ICGA'91)*, July 13–16, 1991. Morgan Kaufmann Publishers Inc.: San Francisco, CA, USA. ISBN 1-55860-208-9.
23. John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Bradford Books. MIT Press: Cambridge, MA, USA, December 1992. ISBN 0-262-11170-5. 1992 first edition, 1993 second edition.
24. Zbigniew Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag GmbH: Berlin, Germany, 1996. ISBN 3-540-58090-5 and 3-540-60676-9.
25. Zbigniew Michalewicz, J. David Schaffer, Hans-Paul Schwefel, David B. Fogel, and Hiroaki Kitano, editors. *Proceedings of the First IEEE Conference on Evolutionary Computation (CEC'94)*, June 27–29, 1994. IEEE Computer Society: Piscataway, NJ, USA, IEEE Computer Society: Piscataway, NJ, USA. ISBN 0-7803-1899-4 and 0-7803-1900-1.
26. Carlos Artemio Coello Coello, Gary B. Lamont, and David A. van Veldhuizen. *Evolutionary Algorithms for Solving Multi-Objective Problems*, volume 5 of *Genetic and Evolutionary Computation*. Springer US: Boston, MA, USA and Kluwer Academic Publishers: Norwell, MA, USA, 2nd edition, 2002. ISBN 0306467623 and 0387332545. 10.1007/978-0-387-36797-2.
27. Kalyanmoy Deb. *Multi-Objective Optimization Using Evolutionary Algorithms*. Wiley Interscience Series in Systems and Optimization. John Wiley & Sons Ltd.: New York, NY, USA, May 2001. ISBN 047187339X.
28. N. Srinivas and Kalyanmoy Deb. Muiltiobjective Optimization Using Nondominated Sorting in Genetic Algorithms. *Evolutionary Computation*, 2(3):221–248, 1994. 10.1162/evco.1994.2.3.221. URL `http://www.eng.auburn.edu/~smithae/publications/journal/Multi-objective%20optimization%20using%20genetic%20a`

29. Kalyanmoy Deb, Amrit Pratab, Samir Agrawal, and T Meyarivan. A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation (IEEE-EC)*, 6(2):182–197, April 2002. 10.1109/4235.996017. URL `http://dynamics.org/~altenber/UH_ICS/EC_REFS/MULTI_OBJ/DebPratapAgarwalMeyarivan.pdf`.
30. Eckart Zitzler and Lothar Thiele. An Evolutionary Algorithm for Multiobjective Optimization: The Strength Pareto Approach. TIK-Report 43, Eidgenössische Technische Hochschule (ETH) Zürich, Department of Electrical Engineering, Computer Engineering and Networks Laboratory (TIK): Zürich, Switzerland, May 1998. URL `http://www.tik.ee.ethz.ch/sop/publicationListFiles/zt1998a.pdf`.
31. Eckart Zitzler, Marco Laumanns, and Lothar Thiele. SPEA2: Improving the Strength Pareto Evolutionary Algorithm. TIK-Report 101, Eidgenössische Technische Hochschule (ETH) Zürich, Department of Electrical Engineering, Computer Engineering and Networks Laboratory (TIK): Zürich, Switzerland, May 2001. URL `http://www.tik.ee.ethz.ch/sop/publicationListFiles/zlt2001a.pdf`. Errata added 2001-09-27.
32. W. Daniel Hillis. Co-Evolving Parasites Improve Simulated Evolution as an Optimization Procedure. *Physica D: Nonlinear Phenomena*, 42(1-2):228–234, June 1990. 10.1016/0167-2789(90)90076-2.
33. Mitchell A. Potter and Kenneth Alan De Jong. A Cooperative Coevolutionary Approach to Function Optimization. In Yuval Davidor, Hans-Paul Schwefel, and Reinhard Männer, editors, *Proceedings of the Third Conference on Parallel Problem Solving from Nature; International Conference on Evolutionary Computation (PPSN III)*, volume 866/1994 of *Lecture Notes in Computer Science (LNCS)*, pages 249–257. Springer-Verlag GmbH: Berlin, Germany, 1994. 10.1007/3-540-58484-6_269.
34. Mitchell A. Potter and Kenneth Alan De Jong. Cooperative Coevolution: An Architecture for Evolving Coadapted Subcomponents. *Evolutionary Computation*, 8(1):1–29, 2000. 10.1162/106365600568086. URL `http://mitpress.mit.edu/journals/EVCO/Potter.pdf`.
35. Wenxiang Chen, Thomas Weise, Zhenyu Yang, and Ke Tang. Large-Scale Global Optimization Using Cooperative Coevolution with Variable Interaction Learning. In Robert Schaefer, Carlos Cotta, Joanna Kołodziej, and Günter Rudolph, editors, *Proceedings of the 11th International Conference on Parallel Problem Solving From Nature, Part 2 (PPSN'10-2)*, volume 6239 of *Theoretical Computer Science and General Issues (SL 1), Lecture Notes in Computer Science (LNCS)*, pages 300–309. Springer-Verlag GmbH: Berlin, Germany, 2010. 10.1007/978-3-642-15871-1_31. URL `http://mail.ustc.edu.cn/~chenwx/ppsn10Chen.pdf`.
36. John J. Grefenstette. Deception Considered Harmful. In L. Darrell Whitley, editor, *Proceedings of the Second Workshop on Foundations of Genetic Algorithms (FOGA'92)*, pages 75–91. Morgan Kaufmann Publishers Inc.: San Francisco, CA, USA, 1992.
37. Kalyanmoy Deb and David Edward Goldberg. Analyzing Deception in Trap Functions. In L. Darrell Whitley, editor, *Proceedings of the Second Workshop on Foundations of Genetic Algorithms (FOGA'92)*, pages 93–108. Morgan Kaufmann Publishers Inc.: San Francisco, CA, USA, 1992.
38. Kenneth Alan De Jong. Genetic Algorithms are NOT Function Optimizers. In L. Darrell Whitley, editor, *Proceedings of the Second Workshop on Foundations of Genetic Algorithms (FOGA'92)*, pages 5–17. Morgan Kaufmann Publishers Inc.: San Francisco, CA, USA, 1992. URL `http://www.mli.gmu.edu/papers/91-95/92-12.pdf`.
39. David Edward Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc.: Boston, MA, USA, 1989. ISBN 0-201-15767-5.
40. Kenneth Alan De Jong. *Evolutionary Computation: A Unified Approach*, volume 4 of *Complex Adaptive Systems, Bradford Books*. MIT Press: Cambridge, MA, USA, February 2006. ISBN 0262041944 and 8120330021.

41. Thomas Bäck, David B. Fogel, and Zbigniew Michalewicz, editors. *Handbook of Evolutionary Computation*. Computational Intelligence Library. Oxford University Press, Inc.: New York, NY, USA, Institute of Physics Publishing Ltd. (IOP): Dirac House, Temple Back, Bristol, UK, and CRC Press, Inc.: Boca Raton, FL, USA, January 1, 1997. ISBN 0-7503-0392-1 and 0-7503-0895-8.
42. HB1998HHGEC. Hitch-Hiker's Guide to Evolutionary Computation: A List of Frequently Asked Questions (FAQ) (HHGT), March 29, 2000. URL `http://www.aip.de/~ast/EvolCompFAQ/`. USENET: comp.ai.genetic.
43. L. Darrell Whitley. A Genetic Algorithm Tutorial. *Statistics and Computing*, 4(2):65–85, June 1994. 10.1007/BF00175354. URL `http://samizdat.mines.edu/ga_tutorial/ga_tutorial.ps`.
44. John Henry Holland. Genetic Algorithms – Computer programs that "evolve" in ways that resemble natural selection can solve complex problems even their creators do not fully understand. *Scientific American*, 267(1):44–50, July 1992. URL `http://members.fortunecity.com/templarseries/algo.html`.
45. Riccardo Poli, William B. Langdon, and Nicholas Freitag McPhee. *A Field Guide to Genetic Programming*. Lulu Enterprises UK Ltd: London, UK, March 2008. ISBN 1-4092-0073-6. URL `http://www.gp-field-guide.org.uk/`. With contributions by John R. Koza.
46. Roberto R. F. Mendes, Fabricio de B. Voznika, Alex Alves Freitas, and Julio C. Nievola. Discovering Fuzzy Classification Rules with Genetic Programming and Co-Evolution. In Luc de Raedt and Arno Siebes, editors, *5th European Conference on Principles of Data Mining and Knowledge Discovery (PKDD'01)*, volume 2168 of *Lecture Notes in Artificial Intelligence (LNAI, SL7), Lecture Notes in Computer Science (LNCS)*, pages 314–325. Springer-Verlag GmbH: Berlin, Germany, 2001. 10.1007/3-540-44794-6_26. URL `http://www.cs.bham.ac.uk/~wbl/biblio/gp-html/Mendes_2001_PKDD.html`.
47. Pu Wang, Thomas Weise, and Raymond Chiong. Novel Evolutionary Algorithms for Supervised Classification Problems: An Experimental Study. *Evolutionary Intelligence*, 4(1):3–16, January 12, 2011. 10.1007/s12065-010-0047-7. URL `http://www.it-weise.de/documents/files/WWC2011NEAFSCPAES.pdf`.
48. John R. Koza. Concept Formation and Decision Tree Induction using the Genetic Programming Paradigm. In Hans-Paul Schwefel and Reinhard Männer, editors, *Proceedings of the 1st Workshop on Parallel Problem Solving from Nature (PPSN I)*, volume 496/1991 of *Lecture Notes in Computer Science (LNCS)*, pages 124–128. Springer-Verlag GmbH: Berlin, Germany, 1990. 10.1007/BFb0029742.
49. John R. Koza, David Andre, Forrest H. Bennett III, and Martin A. Keane. Use of Automatically Defined Functions and Architecture-Altering Operations in Automated Circuit Synthesis Using Genetic Programming. In John R. Koza, David Edward Goldberg, David B. Fogel, and Rick L. Riolo, editors, *Proceedings of the First Annual Conference of Genetic Programming (GP'96)*, Complex Adaptive Systems, Bradford Books, pages 132–149. MIT Press: Cambridge, MA, USA, 1996.
50. Ingo Rechenberg. *Cybernetic Solution Path of an Experimental Problem*. Royal Aircraft Establishment: Farnborough, Hampshire, UK, August 1965. Library Translation 1122.
51. Ingo Rechenberg. *Evolutionsstrategie '94*, volume 1 of *Werkstatt Bionik und Evolutionstechnik*. Frommann-Holzboog Verlag: Bad Cannstadt, Stuttgart, Baden-Württemberg, Germany, 1994. ISBN 3-7728-1642-8.
52. Hans-Paul Schwefel. Kybernetische Evolution als Strategie der exprimentellen Forschung in der Strömungstechnik. Master's thesis, Technische Universität Berlin: Berlin, Germany, 1965.
53. Hans-Paul Schwefel. Experimentelle Optimierung einer Zweiphasendüse Teil I. Technical Report 35, AEG Research Institute: Berlin, Germany, 1968. Project MHD–Staustrahlrohr 11.034/68.

54. Hans-Paul Schwefel. *Evolutionsstrategie und numerische Optimierung*. PhD thesis, Technische Universität Berlin, Institut für Meß- und Regelungstechnik, Institut für Biologie und Anthropologie: Berlin, Germany, 1975.
55. Thomas Bäck, Frank Hoffmeister, and Hans-Paul Schwefel. A Survey of Evolution Strategies. In Richard K. Belew and Lashon Bernard Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms (ICGA'91)*, pages 2–9. Morgan Kaufmann Publishers Inc.: San Francisco, CA, USA, 1991. URL `http://130.203.133.121:8080/viewdoc/summary?doi=10.1.1.42.3375`.
56. Hans-Georg Beyer and Hans-Paul Schwefel. Evolution Strategies – A Comprehensive Introduction. *Natural Computing: An International Journal*, 1(1):3–52, March 2002. 10.1023/A:1015059928466. URL `http://www.cs.bham.ac.uk/~pxt/NIL/es.pdf`.
57. Hans-Paul Schwefel. *Evolution and Optimum Seeking*. Sixth-Generation Computer Technology Series. John Wiley & Sons Ltd.: New York, NY, USA, 1995. ISBN 0-471-57148-2.
58. Hans-Georg Beyer. *The Theory of Evolution Strategies*. Natural Computing Series. Springer New York: New York, NY, USA, May 27, 2001. ISBN 3-540-67297-4.
59. Nikolaus Hansen, Andreas Ostermeier, and Andreas Gawelczyk. On the Adaptation of Arbitrary Normal Mutation Distributions in Evolution Strategies: The Generating Set Adaptation. In Larry J. Eshelman, editor, *Proceedings of the Sixth International Conference on Genetic Algorithms (ICGA'95)*, pages 57–64. Morgan Kaufmann Publishers Inc.: San Francisco, CA, USA, 1995.
60. Silja Meyer-Nieberg and Hans-Georg Beyer. Self-Adaptation in Evolutionary Algorithms. In Fernando G. Lobo, Cláudio F. Lima, and Zbigniew Michalewicz, editors, *Parameter Setting in Evolutionary Algorithms*, volume 54 of *Studies in Computational Intelligence*, chapter 3, pages 47–75. Springer-Verlag: Berlin/Heidelberg, 2007. 10.1007/978-3-540-69432-8_3.
61. Oliver Kramer. *Self-Adaptive Heuristics for Evolutionary Computation*, volume 147 of *Studies in Computational Intelligence*. Springer-Verlag: Berlin/Heidelberg, July 2008. ISBN 3-540-69280-0. 10.1007/978-3-540-69281-2.
62. Nikolaus Hansen and Andreas Ostermeier. Adapting Arbitrary Normal Mutation Distributions in Evolution Strategies: The Covariance Matrix Adaptation. In Keisoku Jidō and Seigyo Gakkai, editors, *Proceedings of IEEE International Conference on Evolutionary Computation (CEC'96)*, pages 312–317. IEEE Computer Society Press: Los Alamitos, CA, USA, 1996. 10.1109/ICEC.1996.542381. URL `http://www.bionik.tu-berlin.de/ftp-papers/CMAES.ps.Z`.
63. Nikolaus Hansen and Andreas Ostermeier. Convergence Properties of Evolution Strategies with the Derandomized Covariance Matrix Adaption: The $(\mu/\mu_I, \lambda)$-CMA-ES. In Hans-Jürgen Zimmermann, editor, *Proceedings of the 5th European Congress on Intelligent Techniques and Soft Computing (EUFIT'97)*, volume 1, pages 650–654. ELITE Foundation: Aachen, North Rhine-Westphalia, Germany and Wissenschaftsverlag Mainz: Germany, 1997.
64. Nikolaus Hansen and Andreas Ostermeier. Completely Derandomized Self-Adaptation in Evolution Strategies. *Evolutionary Computation*, 9(2):159–195, 2001. URL `http://www.bionik.tu-berlin.de/user/niko/cmaartic.pdf`.
65. Grahame A. Jastrebski and Dirk V. Arnold. Improving Evolution Strategies through Active Covariance Matrix Adaptation. In Gary G. Yen, Simon M. Lucas, Gary B. Fogel, Graham Kendall, Ralf Salomon, Byoung-Tak Zhang, Carlos Artemio Coello Coello, and Thomas Philip Runarsson, editors, *Proceedings of the IEEE Congress on Evolutionary Computation (CEC'06), 2006 IEEE World Congress on Computation Intelligence (WCCI'06)*, pages 9719–9726. IEEE Computer Society: Piscataway, NJ, USA, IEEE Computer Society: Piscataway, NJ, USA, 2006. 10.1109/CEC.2006.1688662.
66. Lawrence Jerome Fogel. *On the Organization of Intellect*. PhD thesis, University of California (UCLA): Los Angeles, CA, USA, 1964.

67. Ágoston E. Eiben and James E. Smith. *Introduction to Evolutionary Computing*. Natural Computing Series. Springer New York: New York, NY, USA, 1st edition, November 2003. ISBN 3540401849.
68. Xin Yao, Yong Liu, and Guangming Lin. Evolutionary Programming Made Faster. *IEEE Transactions on Evolutionary Computation (IEEE-EC)*, 3(2):82–102, July 1999. 10.1109/4235.771163. URL `http://eprints.kfupm.edu.sa/38413/`.
69. William Eugene Hart, Natalio Krasnogor, and James E. Smith. Memetic Evolutionary Algorithms. In William Eugene Hart, Natalio Krasnogor, and James E. Smith, editors, *Recent Advances in Memetic Algorithms*, volume 166/2005 of *Studies in Fuzziness and Soft Computing*, chapter 1, pages 3–27. Springer-Verlag GmbH: Berlin, Germany, 2005.
70. David H. Wolpert and William G. Macready. No Free Lunch Theorems for Optimization. *IEEE Transactions on Evolutionary Computation (IEEE-EC)*, 1(1):67–82, April 1997. 10.1109/4235.585893.
71. Christian Igel and Marc Toussaint. On Classes of Functions for which No Free Lunch Results Hold. *Information Processing Letters*, 86(6):317–321, June 30, 2003. 10.1016/S0020-0190(03)00222-9.
72. Pablo Moscato. On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts: Towards Memetic Algorithms. Caltech Concurrent Computation Program C3P 826, California Institute of Technology (Caltech), Caltech Concurrent Computation Program (C3P): Pasadena, CA, USA, 1989.
73. Michael G. Norman and Pablo Moscato. A Competitive and Cooperative Approach to Complex Combinatorial Search. Caltech Concurrent Computation Program 790, California Institute of Technology (Caltech), Caltech Concurrent Computation Program (C3P): Pasadena, CA, USA, 1989.
74. Michael G. Norman and Pablo Moscato. A Competitive and Cooperative Approach to Complex Combinatorial Search. In *Proceedings of the 20th Informatics and Operations Research Meeting (20th Jornadas Argentinas e Informática e Investigación Operativa) (JAIIO'91)*, pages 3.15–3.29, 1991. Also published as Technical Report Caltech Concurrent Computation Program, Report. 790, California Institute of Technology, Pasadena, California, USA, 1989.
75. Richard Dawkins. *The Selfish Gene*. Oxford University Press, Inc.: New York, NY, USA, 1st/2nd edition, 1976. ISBN 0-192-86092-5.
76. Yew-Soon Ong, Meng Hot Lim, Ning Zhu, and Kok-Wai Wong. Classification of Adaptive Memetic Algorithms: A Comparative Study. *IEEE Transactions on Systems, Man, and Cybernetics – Part B: Cybernetics*, 36(1):141–152, February 2006. 10.1109/TSMCB.2005.856143.
77. Peter Cowling, Graham Kendall, and Eric Soubeiga. A Hyperheuristic Approach to Scheduling a Sales Summit. In Edmund K. Burke and Wilhelm Erben, editors, *Selected Papers from the Third International Conference on Practice and Theory of Automated Timetabling (PATAT'00)*, volume 2079/2001 of *Lecture Notes in Computer Science (LNCS)*, pages 176–190. Springer-Verlag GmbH: Berlin, Germany, 2000. 10.1007/3-540-44629-X_11. URL `http://www.asap.cs.nott.ac.uk/publications/pdf/exs_patat01.pdf`.
78. Graham Kendall, Peter Cowling, and Eric Soubeiga. Choice Function and Random HyperHeuristics. In Kay Chen Tan, Meng Hot Lim, Xin Yao, and Lipo Wang, editors, *Recend Advances in Simulated Evolution and Learning – Proceedings of the Fourth Asia-Pacific Conference on Simulated Evolution And Learning (SEAL'02)*, volume 2 of *Advances in Natural Computation*, pages 667–671. World Scientific Publishing Co.: Singapore, 2002. URL `http://www.cs.nott.ac.uk/~gxk/papers/seal01.pdf`.
79. Edmund K. Burke, Graham Kendall, and Eric Soubeiga. A Tabu Search Hyperheuristic for Timetabling and Rostering. *Journal of Heuristics*, 9 (6):451–470, December 2003. 10.1023/B:HEUR.0000012446.94732.b6. URL `http://www.asap.cs.nott.ac.uk/publications/pdf/Journal_eric.pdf`.

80. Anna V. Kononova, Derek B. Ingham, and Mohamed Pourkashanian. Simple Scheduled Memetic Algorithm for Inverse Problems in Higher Dimensions: Application to Chemical Kinetics. In Zbigniew Michalewicz and Robert G. Reynolds, editors, *Proceedings of the IEEE Congress on Evolutionary Computation (CEC'08), Computational Intelligence: Research Frontiers – IEEE World Congress on Computational Intelligence – Plenary/Invited Lectures (WCCI)*, volume 5050/2008 of *Theoretical Computer Science and General Issues (SL 1), Lecture Notes in Computer Science (LNCS)*, pages 3905–3912. IEEE Computer Society: Piscataway, NJ, USA, 2008. 10.1109/CEC.2008.4631328.
81. James E. Smith. Coevolving Memetic Algorithms: A Review and Progress Report. *IEEE Transactions on Systems, Man, and Cybernetics – Part B: Cybernetics*, 37(1):6–17, 2007. 10.1109/TSMCB.2006.883273. URL `http://sci2s.ugr.es/docencia/algoritmica/04067089.pdf`.
82. E. L. Yu and Ponnuthurai Nagaratnam Suganthan. Ensemble of Niching Algorithms. *Information Sciences – Informatics and Computer Science Intelligent Systems Applications: An International Journal*, 180(15):2815–2833, August 1, 2010. 10.1016/j.ins.2010.04.008.
83. Natalio Krasnogor and James E. Smith. A Tutorial for Competent Memetic Algorithms: Model, Taxonomy, and Design Issues. *IEEE Transactions on Evolutionary Computation (IEEE-EC)*, 9(5):474–488, October 2005. 10.1109/TEVC.2005.850260. URL `http://www.cs.nott.ac.uk/~nxk/PAPERS/IEEE-TEC-lastVersion.pdf`.
84. Yew-Soon Ong and Andy J. Keane. Meta-Lamarckian Learning in Memetic Algorithms. *IEEE Transactions on Evolutionary Computation (IEEE-EC)*, 8(2):99–110, 2004. 10.1109/TEVC.2003.819944. URL `http://eprints.soton.ac.uk/22794/1/ong_04.pdf`.
85. Peter Korošec, Jurij Šilc, and Bogdan Filipič. The Differential Ant-Stigmergy Algorithm. *Information Sciences – Informatics and Computer Science Intelligent Systems Applications: An International Journal*, 2011.
86. Quang Huy Nguyen, Yew-Soon Ong, Meng Hot Lim, and Natalio Krasnogor. Adaptive Cellular Memetic Algorithms. *Evolutionary Computation*, 17(2):231–256, 2009. 10.1162/evco.2009.17.2.231. URL `http://www.cs.nott.ac.uk/~nxk/PAPERS/CMA.PDF`.
87. Minh Nghia Le, Yew-Soon Ong, Yaochu Jin, and Bernhard Sendhoff. Lamarckian Memetic Algorithms: Local Optimum and Connectivity Structure Analysis. *Memetic Computing*, 1(3):175–190, November 2009. 10.1007/s12293-009-0016-9. URL `http://ntu-cg.ntu.edu.sg/ysong/journal/Lamarckian-MA-Analysis.pdf`.
88. Andrea Caponio, Giuseppe Leonardo Cascella, Ferrante Neri, Nadia Salvatore, and Mark Sumner. A Fast Adaptive Memetic Algorithm for Online and Offline Control Design of PMSM Drives. *IEEE Transactions on Systems, Man, and Cybernetics – Part B: Cybernetics*, 37(1):28–41, February 2007. 10.1109/TSMCB.2006.883271.
89. Ferrante Neri, Jari Toivanen, Giuseppe Leonardo Cascella, and Yew-Soon Ong. An Adaptive Multimeme Algorithm for Designing HIV Multidrug Therapies. *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)*, 4(2), April 2007. 10.1109/TCBB.2007.070202. URL `http://ntu-cg.ntu.edu.sg/ysong/journal/Adaptive-Multimeme-HIV.pdf`.
90. Ferrante Neri, Jari Toivanen, and Raino A. E. Mäkinen. An Adaptive Evolutionary Algorithm with Intelligent Mutation Local Searchers for Designing Multidrug Therapies for HIV. *Applied Intelligence – The International Journal of Artificial Intelligence, Neural Networks, and Complex Problem-Solving Technologies*, 27(3): 219–235, December 2007. 10.1007/s10489-007-0069-8.
91. Ferrante Neri, Ville Tirronen, Tommi Kärkkäinen, and Tuomo Rossi. Fitness Diversity based Adaptation in Multimeme Algorithms: A Comparative Study. In *Proceedings of the IEEE Congress on Evolutionary Computation*

(CEC'07), pages 2374–2381. IEEE Computer Society: Piscataway, NJ, USA, 2007. 10.1109/CEC.2007.4424768.

92. Ville Tirronen, Ferrante Neri, Tommi Kärkkäinen, Kirsi Majava, and Tuomo Rossi. An Enhanced Memetic Differential Evolution in Filter Design for Defect Detection in Paper Production. *Evolutionary Computation*, 16(4):529–555, 2008. 10.1162/evco.2008.16.4.529.
93. Andrea Caponio, Ferrante Neri, and Ville Tirronen. Super-fit Control Adaptation in Memetic Differential Evolution Frameworks. *Soft Computing – A Fusion of Foundations, Methodologies and Applications*, 13(8-9):811–831, July 2009. 10.1007/s00500-008-0357-1.
94. Jing Tang, Meng Hot Lim, and Yew-Soon Ong. Diversity-Adaptive Parallel Memetic Algorithm for Solving Large Scale Combinatorial Optimization Problems. *Soft Computing – A Fusion of Foundations, Methodologies and Applications*, 11(9):873–888, July 2007. 10.1007/s00500-006-0139-6.
95. Quan Yuan, Feng Qian, and Wenli Du. A Hybrid Genetic Algorithm with the Baldwin Effect. *Information Sciences – Informatics and Computer Science Intelligent Systems Applications: An International Journal*, 180(5):640–652, March 2010. 10.1016/j.ins.2009.11.015.
96. Maoguo Gong, Licheng Jiao, and Lining Zhang. Baldwinian Learning in Clonal Selection Algorithm for Optimization. *Information Sciences – Informatics and Computer Science Intelligent Systems Applications: An International Journal*, 180(8): 1218–1236, April 15, 2010. 10.1016/j.ins.2009.12.007.
97. Muddassar Farooq. *Bee-Inspired Protocol Engineering – From Nature to Networks*, volume 15 of *Natural Computing Series*. Springer New York: New York, NY, USA, 2009. ISBN 3-540-85953-5. 10.1007/978-3-540-85954-3.
98. Mike Campos, Eric W. Bonabeau, Guy Théraulaz, and Jean-Louis Deneubourg. Dynamic Scheduling and Division of Labor in Social Insects. *Adaptive Behavior*, 8(2):83–95, March 2000. 10.1177/105971230000800201. URL `http://www.icosystem.com/dynamic-scheduling-and-division-of-labor-in-social-insects/`.
99. Russel C. Eberhart and James Kennedy. A New Optimizer Using Particle Swarm Theory. In *Proceedings of the Sixth International Symposium on Micro Machine and Human Science (MHS'95)*, pages 39–43. IEEE Computer Society: Piscataway, NJ, USA, 1995. 10.1109/MHS.1995.494215. URL `http://webmining.spd.louisville.edu/Websites/COMB-OPT/FINAL-PAPERS/SwarmsPaper.pdf`.
100. James Kennedy and Russel C. Eberhart. Particle Swarm Optimization. In *Proceedings of the IEEE International Conference on Neural Networks (ICNN'95)*, volume 4, pages 1942–1948. IEEE Computer Society Press: Los Alamitos, CA, USA, 1995. 10.1109/ICNN.1995.488968. URL `http://www.engr.iupui.edu/~shi/Coference/psopap4.html`.
101. Russel C. Eberhart and Yuhui Shi. A Modified Particle Swarm Optimizer. In Patrick K. Simpson, editor, *The 1998 IEEE International Conference on Evolutionary Computation (CEC'98), 1998 IEEE World Congress on Computation Intelligence (WCCI'98)*, pages 69–73. IEEE Computer Society: Piscataway, NJ, USA, IEEE Computer Society: Piscataway, NJ, USA, 1998. 10.1109/ICEC.1998.699146.
102. Julia K. Parrish and William M. Hamner, editors. *Animal Groups in Three Dimensions: How Species Aggregate*. Cambridge University Press: Cambridge, UK, December 1997. ISBN 0521460247. 10.2277/0521460247.
103. Yuelin Gao and Yuhong Duan. An Adaptive Particle Swarm Optimization Algorithm with New Random Inertia Weight. In De-Shuang Huang, Laurent Heutte, and Marco Loog, editors, *Advanced Intelligent Computing Theories and Applications. With Aspects of Contemporary Intelligent Computing Techniques – Proceedings of the Third International Conference on Intelligent Computing (ICIC'07-2)*, volume 2 of *Communications in Computer and Information Science*, pages 342–350. Springer-Verlag GmbH: Berlin, Germany, 2007. 10.1007/978-3-540-74282-1_39.

104. Yuelin Gao and Zihui Ren. Adaptive Particle Swarm Optimization Algorithm With Genetic Mutation Operation. In Jingsheng Lei, JingTao Yao, and Qingfu Zhang, editors, *Proceedings of the Third International Conference on Advances in Natural Computation (ICNC'07)*, volume 2, pages 211–215. IEEE Computer Society Press: Los Alamitos, CA, USA, 2007. 10.1109/ICNC.2007.161.
105. Marco Dorigo, Vittorio Maniezzo, and Alberto Colorni. The Ant System: Optimization by a Colony of Cooperating Agents. *IEEE Transactions on Systems, Man, and Cybernetics – Part B: Cybernetics*, 26(1):29–41, February 1996. 10.1109/3477.484436. URL `ftp://iridia.ulb.ac.be/pub/mdorigo/journals/IJ.10-SMC96.pdf`.
106. Eric W. Bonabeau, Marco Dorigo, and Guy Théraulaz. *Swarm Intelligence: From Natural to Artificial Systems*. Oxford University Press, Inc.: New York, NY, USA, August 1999. ISBN 0195131592.
107. Eric W. Bonabeau, Marco Dorigo, and Guy Théraulaz. Inspiration for Optimization from Social Insect Behavior. *Nature*, 406:39–42, July 6, 2000. 10.1038/35017500. URL `http://biology.unm.edu/pibbs/classes/readings/socialinsectbehavior.pdf`.
108. Maurice Clerc and James Kennedy. The Particle Swarm – Explosion, Stability, and Convergence in a Multidimensional Complex Space. *IEEE Transactions on Evolutionary Computation (IEEE-EC)*, 6(1):58–73, February 2002. 10.1109/4235.985692. URL `http://clerc.maurice.free.fr/pso/TEC.zip`.
109. Rui Mendes, James Kennedy, and José Neves. Fully Informed Particle Swarm: Simpler, Maybe Better. *IEEE Transactions on Evolutionary Computation (IEEE-EC)*, 8(3):204–210, June 2004. 10.1109/TEVC.2004.826074. URL `http://www.cpdee.ufmg.br/~joao/CE/ArtigosPSO/FI_PSO.pdf`.
110. Marco Dorigo and Thomas Stützle. *Ant Colony Optimization*. Bradford Books. MIT Press: Cambridge, MA, USA, July 1, 2004. ISBN 0-262-04219-3.
111. Christian Blum. Ant Colony Optimization: Introduction and Recent Trends. *Physics of Life Reviews*, 2(4):353–373, December 2005. 10.1016/j.plrev.2005.10.001. URL `http://www.ie.metu.edu.tr/~ie505/CourseMaterial/Blum%20%282005%29.pdf`.
112. Mark Zlochin, Mauro Birattari, Nicolas Meuleau, and Marco Dorigo. Model-Based Search for Combinatorial Optimization: A Critical Survey. *Annals of Operations Research*, 132(1-4):373–395, November 2004. 10.1023/B:ANOR.0000039526.52305.af.
113. Pedro Larrañaga and José Antonio Lozano, editors. *Estimation of Distribution Algorithms – A New Tool for Evolutionary Computation*, volume 2 of *Genetic and Evolutionary Computation*. Springer US: Boston, MA, USA and Kluwer Academic Publishers: Norwell, MA, USA, 2001. ISBN 0-7923-7466-5.
114. Shumeet Baluja and Richard A. Caruana. Removing the Genetics from the Standard Genetic Algorithm. In Armand Prieditis and Stuart J. Russell, editors, *Proceedings of the Twelfth International Conference on Machine Learning (ICML'95)*, pages 38–46. Morgan Kaufmann Publishers Inc.: San Francisco, CA, USA, 1995. URL `http://www.cs.cornell.edu/~caruana/ml95.ps`.
115. Christian Blum and Marco Dorigo. The Hyper-Cube Framework for Ant Colony Optimization. *IEEE Transactions on Systems, Man, and Cybernetics – Part B: Cybernetics*, 34(2):1161–1172, April 2004. 10.1109/TSMCB.2003.821450. URL `http://code.ulb.ac.be/dbfiles/BluDor2004tsmcb.pdf`.
116. Thomas Weise, Stefan Niemczyk, Raymond Chiong, and Mingxu Wan. A Framework for Multi-Model EDAs with Model Recombination. In *Proceedings of the 4th European Event on Bio-Inspired Algorithms for Continuous Parameter Optimisation (EvoNUM'11), Applications of Evolutionary Computation – Proceedings of EvoApplications 2011: EvoCOMPLEX, EvoGAMES, EvoIASP, EvoINTELLIGENCE, EvoNUM, and EvoSTOC, Part 1 (EvoAPPLICATIONS'11)*, volume 6624 of *Theoretical Computer Science and General Issues (SL 1), Lecture Notes in Computer Science (LNCS)*, pages 304–313. Springer-Verlag GmbH: Berlin, Germany, 2011. 10.1007/978-3-642-20525-5_31. URL `http://www.it-weise.de/documents/files/WNCW2011AFFMMEWMR.pdf`.

This is a preview version of paper [1] (see page 31 for the reference). It is posted here for your personal use and not for redistribution. The final publication and definite version is available from Springer (who hold the copyright) at http://link.springer.com. See also http://dx.doi.org/10.1007/978-3-642-23424-8_1. (and the book Variants of Evolutionary Algorithms for Real-World Applications).

```
@incollection{BCCJMNW2011EO,
  title     = {Evolutionary Optimization},
  author    = {Christian Blum and Raymond Chiong and Maurice Clerc and Kenneth Alan {De Jong}
               and Zbigniew Michalewicz and Ferrante Neri and Thomas Weise},
  booktitle = {Variants of Evolutionary Algorithms for Real-World Applications},
  editor    = {Raymond Chiong and Thomas Weise and Zbigniew Michalewicz},
  chapter   = {1},
  pages     = {1--29},
  publisher = {Springer},
  address   = {Berlin Heidelberg},
  year      = {2011},
  isbn      = {978-3-642-23423-1},
  doi       = {10.1007/978-3-642-23424-8_1},
},
```