An Accelerated Introduction to Memetic Algorithms

Pablo Moscato and Carlos Cotta

Abstract Memetic algorithms (MAs) are optimization techniques based on the orchestrated interplay between global and local search components and have the exploitation of specific problem knowledge as one of their guiding principles. In its most classical form, a MA is typically composed of aun underlying population-based engine onto which a local search component is integrated. These aspects are described in this chapter in some detail, paying particular attention to design and integration issues. After this description of the basic architecture of MAs, we move to different algorithmic extensions that give rise to more sophisticated memetic approaches. After providing a meta-review of the numerous practical applications of MAs, we close this chapter with an overview of current perspectives of memetic algorithms.

1 Introduction and historical notes

The generic denomination of *'Memetic Algorithms'* (MAs) [133] is used to encompass a broad class of metaheuristics, understanding the latter as high-level templates that orchestrate the functioning on low-level rules and heuristics. The method, which is based on a population of agents, had practical success in a variety of problem domains, in particular for the heuristic resolution of **NP**-hard optimization problems.

Unlike traditional evolutionary computation (EC) methods, MAs are intrinsically concerned with exploiting *all available knowledge* about the problem under study. The incorporation of problem domain knowledge is not an optional mechanism,

Pablo Moscato
The University of Newcastle, University Drive, Callaghan NSW 2308, Australia, e-mail: Pablo.Moscato@newcastle.edu.au

Carlos Cotta
Escuela Técnica Superior de Ingeniería Informática, Universidad de Málaga, Campus de Teatinos, 29071 Málaga, Spain, e-mail: ccottap@lcc.uma.es

but a fundamental feature that characterizes MAs. This functioning philosophy is perfectly illustrated by the term "memetic". Coined by R. Dawkins [40], the word *'meme'* denotes an analogous to the gene in the context of cultural evolution [115]. In Dawkins' words:

> *"Examples of memes are tunes, ideas, catch-phrases, clothes fashions, ways of making pots or of building arches. Just as genes propagate themselves in the gene pool by leaping from body to body via sperms or eggs, so memes propagate themselves in the meme pool by leaping from brain to brain via a process which, in the broad sense, can be called imitation."*

This characterization of a meme suggests that in cultural evolution processes, information is not simply transmitted unaltered between individuals. Rather, it is processed and enhanced by the communicating parts. This enhancement is accomplished in MAs by incorporating heuristics, approximation algorithms, local search techniques, specialized recombination operators, truncated exact methods, etc. In essence, most MAs can be interpreted as a search strategy in which a population of optimizing agents cooperate and compete [144]. The success of MAs can probably be explained as being a direct consequence of the *synergy* of the different search approaches they incorporate.

The most crucial and distinctive feature of MAs, the inclusion of problem knowledge, is also supported by strong theoretical results. As Hart and Belew [67] initially stated and Wolpert and Macready [191] later popularized in the so-called *No-Free-Lunch Theorem*, a search algorithm strictly performs in accordance with the amount and quality of the problem knowledge they incorporate. More precisely, the theorem establishes that the performance of any search algorithm is indistinguishable on average from any other one when all possible problems are considered, a scenario that captures the lack of knowledge on the target problem (this very broad assumption can be challenged [44]; this said, similar results can be found for more restricted scenarios [77, 166]). The quest for universal solvers is thus futile [36]: using and exploiting problem knowledge is a requirement for attaining efficient problem solvers [116]. Given that the term *hybridization* is often used to denote the process of incorporating problem knowledge (due to the fact that it is accomplished by combining or *hybridizing* concepts from diffferent resolution algorithms [39]), it is not surprising that MAs are sometimes called 'Hybrid Evolutionary Algorithms' (hybrid EAs) as well.

One of the first algorithms to which the MA label was assigned dates back to 1988 [144], and was regarded by many as a hybrid of *traditional* Genetic Algorithms (GAs) and *Simulated Annealing* (SA). Part of the initial motivation was to find a way out of the limitations of both techniques on a well-studied combinatorial optimization problem the MIN EUCLIDEAN TRAVELING SALESMAN problem (MIN ETSP) – the reader interested in the historical circumstances of the initial developments in this field is directed to a personal and very detailed account in [119]. According to the authors, the original inspiration came from *computer game tournaments* [71] used to study *"the evolution of cooperation"* [4, 130]. That approach had several features which anticipated many current algorithms in practice today. The competitive phase of the algorithm was based on the new allocation of

search points in the configuration space, a process involving a *"battle"* for survival followed by the so-called *"cloning"*, which has a strong similarity with *'go with the winners'* algorithms [1, 151]. Thus, the cooperative phase followed by local search may be better named *"go-with-the-local-winners"* since the optimizing *agents* were arranged in a topology of a two dimensional toroidal lattice. After initial computer experiments, an insight was derived on the particular relevance of the *"spatial"* organization, when coupled with an appropriate set of rules, for the overall performance of population search processes. A few months later, Moscato and Norman discovered that they shared similar views with other researchers [60, 126] and other authors proposing *"island models"* for GAs. Spacialization is now being recognized as the "catalyzer" responsible for a variety of phenomena [129, 130]. This is an important research issue, currently only understood in a rather heuristic way. However, some proper undecidability results have been obtained for related problems [62] giving some hope to a more formal treatment.

Less than a year later, in 1989, Moscato and Norman identified several authors who were also pioneering the introduction of heuristics to improve the solutions before recombining them [59, 127] (see other references and the discussion in [115]). Particularly coming from the GA field, several authors were introducing *problem-domain knowledge* in a variety of ways. In [115] the denomination of *'memetic algorithms'* was introduced for the first time. It was also suggested that *cultural evolution* can be a better working metaphor for these metaheuristics to avoid *"biologically constrained"* thinking that was restricting progress at that time.

Thirty years later, albeit unfortunately under different names, MAs have become an important optimization approach, with several successes in a variety of classical **NP**-hard optimization problems. We aim to provide an updated and self-contained introduction to MAs, focusing on their technical innards and formal features, but without loosing the perspective of their practical applications and open research issues.

2 Memetic Algorithms

Before proceeding to the description of MAs, it is necessary to provide some basic concepts and definitions. Several notions introduced in the first subsection are strongly related to the field of computational complexity. Nevertheless, we approach them in a slightly different way, more oriented to the subsequent development we shall do later on in the chapter. These basic concepts will give rise to the notions of local search and population-based search, upon which MAs are founded. This latter class of search settles the scenario for *recombination*, a crucial mechanism in the functioning of MAs that will be studied to some depth. Finally, a basic algorithmic template and some guidelines for designing MAs will be presented.

2.1 Basic Concepts

An *algorithm* is a detailed step-by-step procedure for solving a *computational problem*. A computational problem P denotes a class of algorithmically-doable tasks, and it has an input domain set of *instances* denoted I_P. For each instance $x \in I_P$, there is an associated set $sol_P(x)$ which denotes the *feasible* solutions for problem P given instance x. The set $sol_P(x)$ is also known as the set of *acceptable* or *valid* solutions.

We are expected to deliver an algorithm that solves problem P; this means that our algorithm, given instance $x \in I_P$, must return at least one element y from a set of *answers* $ans_P(x)$ (also called *given solutions*) that satisfies the requirements of the problem. This is the first design issue to face. To be precise, depending on the kind of answers expected, computational problems can be classified into different categories; for example:

- finding *all* solutions in $sol_P(x)$, i.e., *enumeration* problems.
- counting *how many* solutions exist in $sol_P(x)$, i.e. *counting* problems.
- determining whether the set $sol_P(x)$ *is empty or not*, i.e., *decision* problems.
- finding a solution in $sol_P(x)$ maximizing or minimizing a given function, i.e., *optimization* problems.

In this chapter, we will focus on the last possibility, that is, a problem will be considered *solved* by finding a feasible solution $y \in sol_P(x)$ which is optimal or by giving an indication that no such feasible solution exists. It is thus convenient in many situations to define a Boolean *feasibility* function $feasible_P(x,y)$ in order to identify whether a given solution $y \in ans_P(x)$ is acceptable for an instance $x \in I_P$ of a computational problem P, i.e., checking if $y \in sol_P(x)$.

An algorithm is said to *solve* problem P if it can fulfill this condition for any given instance $x \in I_P$. This definition is certainly too broad, so a more restrictive characterization for our problems of interest is necessary. This characterization is provided by restricting ourselves to the so-called *combinatorial optimization* problems. These constitute a special subclass of computational problems in which for each instance $x \in I_P$:

- the cardinality of $sol_P(x)$ is finite.
- each solution $y \in sol_P(x)$ has a *goodness integer value* $m_P(y,x)$, obtained by means of an associated *objective function* m_P.
- a partial order $\prec_P$ is defined over the set of goodness values returned by the objective function, thus allowing to determine which of two goodness values is preferable.

An instance $x \in I_P$ of a combinatorial optimization problem P is solved by finding the best solution $y^* \in sol_P(x)$, i.e., finding a solution y^* such that no other solution $y \prec_P y^*$ exists if $sol_P(x)$ is not empty. It is very common to have $\prec_P$ defining a total order. In this case, the best solution is the one that maximizes (or minimizes) the objective function.

As an example of a combinatorial optimization problem consider the 0-1 MULTIPLE KNAPSACK PROBLEM (0-1 MKP). Each instance x of this problem is defined by a vector of profits $V = \{v_0, \cdots, v_{n-1}\}$, a vector of capacities $C = \{c_0, \cdots, c_{m-1}\}$, and a matrix of capacity constraints coefficients $M = \{m_{ij} : 0 \leqslant i < m,\ 0 \leqslant j < n\}$. Intuitively, the problem consists of selecting a set of objects so as to maximize the profit of this set without violating the capacity constraints. If the objects are indexed with the elements of the set $\mathbb{N}_n = \{0, 1, \cdots, n-1\}$, the answer set $ans_P(x)$ for an instance x is simply the power set of $\mathbb{N}_n$, that is, each subset of $\mathbb{N}_n$ is a possible answer. Furthermore, the set of feasible answers $sol_P(x)$ is composed of those subsets whose incidence vector B verifies $M \cdot B \leqslant C$. Finally, the objective function is defined as $m_P(y,x) = \sum_{i \in y} v_i$, i.e., the sum of profits for all selected objects, the goal being to maximize this value.

Notice that a *decisional* version can be associated with a combinatorial optimization problem. To formulate the decision problem, an integer goodness value K is considered, and instead of trying to find the best solution of instance x, we ask whether x has a solution whose goodness is equal or better than K. In the above example, we could ask whether a feasible solution y exists such that its associated profit is equal or better than K.

2.2 Search Landscapes

As mentioned above, having defined the concept of combinatorial optimization problem, the goal is to find at least one of the optimal solutions for a given instance. For this purpose, a search algorithm must be used. Before discussing search algorithms, three entities must be discussed. These are the *search space*, the *neighborhood relation*, and the *guiding function*. It is important to consider that, for any given computational problem, these three entities can be instantiated in several ways, giving rise to different optimization tasks.

Let us start by defining the concept of search space for a combinatorial problem P. To do so, we consider a set $\mathscr{S}_P(x)$, whose elements must satisfy the following requirements:

- Each element $s \in \mathscr{S}_P(x)$ represents at least one answer in $ans_P(x)$.
- For decision problems: at least one element of $sol_P(x)$ that stands for a 'Yes' answer must be represented by one element in $\mathscr{S}_P(x)$.
- For optimization problems: at least one *optimal* element y^* of $sol_P(x)$ is represented by one element in $\mathscr{S}_P(x)$.

Each element of $\mathscr{S}_P(x)$ is called a *configuration*. It is related to an answer in $ans_P(x)$ by a *growth function* $g : \mathscr{S}_P(x) \to ans_P(x)$. Note that the first requirement refers to $ans_P(x)$ and not to $sol_P(x)$, i.e., some configurations in the search space may correspond to infeasible solutions. Thus, the search algorithm may need to be prepared to deal with this fact. If these requirements have been achieved, we say that we have a *valid representation* or *valid formulation* of the problem. For simplicity, we will

just write $\mathscr{S}$ to refer to $\mathscr{S}_P(x)$ when x and P are clear from the context. People using biologically-inspired metaphors like to call $\mathscr{S}_P(x)$ the *genotype space* and $ans_P(x)$ the *phenotype space*, so we appropriately refer to g as the *growth function*.

To illustrate this notion of search space, consider again the case of the 0-1 MKP. Since solutions in $ans_P(x)$ are subsets of $\mathbb{N}_n$, we can define the search space as the set of n-dimensional binary vectors. Each vector will represent the incidence vector of a certain subset, i.e., the growth function g is defined as $g(s) = g(b_0 b_1 \cdots b_{n-1}) = \{i \mid b_i = 1\}$. As mentioned above, many binary vectors may correspond to infeasible sets of objects. Another possibility is defining the search space as the set of permutations of elements in $\mathbb{N}_n$ [61]. In this case, the growth function may consist of applying a greedy construction algorithm, considering objects in the order provided by the permutation. Unlike the binary search space previously mentioned, all configurations represent feasible solutions in this case.

The role of the search space is to provide a "ground" where the search algorithm will act. Important properties of the search space that affect the dynamics of the search algorithm are related to the accessibility relationships between the configurations. These relationships are dependent of a *neighborhood function* $\mathscr{N} : \mathscr{S} \to 2^{\mathscr{S}}$. This function assigns to each element $s \in \mathscr{S}$ a set $\mathscr{N}(s) \subseteq \mathscr{S}$ of neighboring configurations of s. The set $\mathscr{N}(s)$ is called the *neighborhood* of s and each member $s' \in \mathscr{N}(s)$ is called a *neighbor* of s.

It must be noted that the neighborhood depends on the instance, so the notation $\mathscr{N}(s)$ is a simplified form of $\mathscr{N}_P(s,x)$ since it is clear from the context. The elements of $\mathscr{N}(s)$ need not be listed explicitly. In fact, it is very usual to define them *implicitly* by referring to a set of possible *moves*, which define *transitions* between configurations. Moves are usually defined as *"local"* modifications of some part of s, where "locality" refers to the fact that the move is done on a single solution to obtain another single solution. This "locality", is one of the key ingredients of *local search*, and actually it has also given the name to the whole search paradigm.

As examples of concrete neighborhood definitions, consider the two representations of solutions for the 0-1 MKP presented above. In the first case (binary representation), moves can be defined as changing the values of a number of bits. If just one bit is modified at a time, the resulting neighborhood structure is the n-dimensional binary hypercube. In the second case (permutation representation), moves can be defined as the interchange of two positions in the permutation. Thus, two configurations are neighboring if, and only if, they differ in exactly two positions.

This definition of locality presented above is not necessarily related to "closeness" under some kind of distance relationship between configurations (except in the tautological situation in which the distance between two configurations s and s' is defined as the number of moves needed to reach s' from s). As a matter of fact, it is possible to give common examples of very complex neighborhood definitions unrelated to intuitive distance measures.

An important feature that must be considered when selecting the class of moves to be used in the search algorithm is its *"ergodicity"*, that is the ability, given any $s \in \mathscr{S}$ to find a sequence of moves that can reach *all other* configurations $s' \in \mathscr{S}$.

In many situations this property is self-evident and no explicit demonstration is required. It is important since even if we have a valid representation (recall the definition above), it is necessary to guarantee *a priori* that at least one optimal solution is reachable from any given initial solution. Again, consider the binary representation of solutions for a 0-1 MKP instance. If moves are defined as single bit-flips, it is easily seen that any configuration s' can be reached from another configuration s in exactly h moves, where h is the Hamming distance between these configurations. This is not always the case though.

The last entity that must be defined is the *guiding function*. To do so, we require a set $\mathscr{F}$ whose elements are termed *fitness* values (typically $\mathscr{F} \equiv \mathbb{R}$), and a partial order $\prec_{\mathscr{F}}$ on $\mathscr{F}$ (typically, but not always, $\prec_{\mathscr{F}} \equiv <$). The guiding function is defined as a function $F_g : \mathscr{S} \to \mathscr{F}$ that associates to each configuration $s \in \mathscr{S}$ a value $F_g(s)$ that assesses the quality of the solution. The behavior of the search algorithm will be "controlled" by these fitness values.

Notice that for optimization problems there is an obvious direct connection between the guiding function F_g and the objective function m_P (and hence between partial orders $\prec_P$ and $\prec_{\mathscr{F}}$). As a matter of fact, it is very common to enforce this relationship to the point that both terms are usually considered equivalent. However, this equivalence is not necessary and, in many situations, not even desirable. For decision problems, since a solution is a 'Yes' or 'No' answer, associated guiding functions usually take the form of *distance to satisfiability*.

A typical example is the BOOLEAN SATISFIABILITY PROBLEM, i.e., determining whether a Boolean expression in conjunctive normal form is satisfiable. In this case, solutions are assignments of Boolean values to variables, and the objective function m_P is a binary function returning 1 if the solution satisfies the Boolean expression, and 0 otherwise. This objective function could be used as the guiding function. However, a much more typical choice is to use the number of satisfied clauses in the current configuration as guiding function, i.e., $F_g(s) = \sum_i f_i(s)$, the sum over clause indexes i of $f_i(s)$, defined as $f_i(s) = 0$ for a yet unsatisfied clause i, and $f_i(s) = 1$ if the clause i is satisfied. Hence, the goal is to maximize this number. Notice that the guiding function in this case is the objective function of the associated **NP**-hard optimization problem called MAX SAT.

The above differentiation between objective function and guiding function is also very important in the context of constrained optimization problems, i.e., problems for which, in general, $sol_P(x)$ is chosen to be a proper subset of $ans_P(x)$. Since the growth function establishes a mapping from $\mathscr{S}$ to $ans_P(x)$, the search algorithm may need to process both feasible solutions (whose goodness values are well-defined) and infeasible solutions (whose goodness values are ill-defined in general). In many implementations of MAs for these problems, a guiding function is defined as a weighted sum of the value of the objective function and the distance to feasibility (which accounts for the constraints). Typically, a higher weight is assigned to the constraints, so as to give preference to feasibility over optimality. Several other remedies to this problem abound, including resorting to multi-objective techniques.

The combination of a certain problem instance and the three entities defined above induces a so-called *fitness landscape* [86]. Essentially, a fitness landscape

can be defined as a weighted digraph, in which the vertices are configurations of the search space $\mathscr{S}$, and the arcs connect neighboring configurations. The weights are the differences between the guiding function values of the two endpoint configurations. The search can thus be seen as the process of "navigating" the fitness landscape using the information provided by the guiding function. This is a very powerful metaphor; it allows interpretations in terms of well-known topographical objects such as *peaks*, *valleys*, *mesas*, etc., which is of great utility to visualize the search progress, and to grasp factors affecting the performance of the process. In particular, the important notion of *local* optimum is associated with this definition of fitness landscape. To be precise, a local optimum is a vertex of the fitness landscape whose guiding function value is better than the values of all its neighbors. Notice that different moves define different neighborhoods and hence different fitness landscapes, even when the same problem instance is considered. For this reason, the notion of local optimum is not intrinsic to a problem instance as it is, sometimes, erroneously considered.

The notion of fitness landscape is not only useful for conceptual or visualization purposes. It also serves as a very useful instrument in order to analyze the properties of the search space as regarded by a certain search algorithm (via the moves used by the latter). Thus, analytical tools such as random-walk correlation or fitness distance correlation can be used to assess the difficulty perceived by the optimizer, and other statistical tools can be utilized to guide the design/parameterization of the search algorithm – see [110].

2.3 Local vs. Population-Based Search

The definitions presented in the previous subsection naturally lead to the notion of *local search algorithm*. A local search algorithm starts from a configuration $s_0 \in \mathscr{S}$, generated at random or constructed by some other algorithm. Subsequently, it iterates using at each step a transition based on the neighborhood of the current configuration. Transitions leading to preferable (according to the partial order $\prec_{\mathscr{F}}$) configurations are accepted, i.e., the newly generated configuration turns to be the current configuration in the next step. Otherwise, the current configuration is kept. This process is repeated until a certain termination criterion is met. Typical criteria are the realization of a pre-specified number of iterations, not having found any improvement in the last m iterations, or even more complex mechanisms based on estimating the probability of being at a local optimum [24]. Due to these characteristics, the approach is metaphorically called *"hill climbing"*. The whole process is sketched in Algorithm 1.

The selection of the particular type of moves (which are also known as *mutations* in the context of GAs) to use does certainly depend on the specific characteristics of the problem and the representation chosen. There is no general advice for this, since it is a matter of the available computer time for the whole process as well as other algorithmic decisions that include ease of coding, etc. In some cases some moves

Algorithm 1: A Local Search Algorithm

```
1  Procedure Local-Search-Engine (current);
2  begin
3      repeat
4          new ← GenerateNeighbor(current);
5          if F_g(new) ≺_ℱ F_g(current) then
6              current ← new;
7          end if
8      until TerminationCriterion();
9      return current;
10 end
```

are conspicuous, for example it can be the change of the value of one single variable or the swap of the values of two different variables. Sometimes the "step" may also be composed of a chain of transitions. For instance, in relation with MAs, Radcliffe and Surry introduced the concept of *Binomial Minimal Mutation*, where the number of mutations to perform is selected according to a certain binomial distribution [160]. In the context of fitness landscapes, this is equivalent to a redefinition of the neighborhood relation, considering two configurations as neighbors when there exists a chain of transitions connecting them.

Local search algorithms are thus characterized by keeping a single configuration at a time. The immediate generalization of this behavior is the simultaneous maintenance of k, $(k \geqslant 2)$ configurations. The term *population-based* search algorithms has been coined to denote search techniques behaving this way.

The availability of several configurations at a time allows the use of new powerful mechanisms for traversing the fitness landscape in addition to the standard mutation operator. The most popular of these mechanisms, the recombination operator, will be studied in more depth in the next section. In any case, notice that the general functioning of population-based search techniques is very similar to the pseudocode depicted in Algorithm 1. As a matter of fact, a population-based algorithm can be seen as a procedure in which we sequentially visit vertices of a hypergraph. Each vertex of the hypergraph represents a set of configurations in $\mathscr{S}_P(x)$, i.e., a population. The next vertex to be visited, i.e., the new population, can be established according to the composition of the neighborhoods of the different transition mechanisms used in the population algorithm. Despite the analogy with local search, it is widely accepted in the scientific literature to apply the denomination 'local' just to one-configuration-at-a-time search algorithms. For this reason, the term 'local' will be used with this interpretation in the remainder of the chapter.

2.4 Recombination

As mentioned in the previous section, local search is based on the application of a mutation operator to a single configuration. Despite the apparent simplicity of this mechanism, "mutation-based" local search has revealed itself a very powerful mechanism for obtaining good quality solutions for **NP**−hard problems. For this reason, some researchers have tried to provide a more theoretically-solid background to this class of search. In this line, it is worth mentioning the definition of the *Polynomial Local Search* class (PLS) by Johnson et al. [85]. Basically, this complexity class comprises a problem and an associated search landscape such that for any given point in the search landscape we can decide in polynomial time if it is a local optimum or not, and in the latter case find an improved solution in the neighborhood. Unfortunately, this does not mean that we can find a local optimum in polynomial time (in fact, it may generally take an exponential number of steps to do so). This fact has justified the quest for additional search mechanisms to be used as standalone operators or as complements to standard mutation.

In this line, recall that population-based search allowed the definition of generalized move operators termed *recombination* operators. In essence, recombination can be defined as a process in which a set $\mathscr{S}_{par}$ of n configurations (informally referred to as "parents") is taken into consideration to create a set $\mathscr{S}_{desc} \subseteq sol_P(x)$ of m new configurations (informally termed "descendants"). The creation of these descendants involves the identification and combination of features extracted from the parents.

At this point, it is possible to consider properties of interest that can be exhibited by recombination operators [160]. The first property, *respect*, represents the exploitation side of recombination. A recombination operator is said to be *respectful*, regarding a particular type of features of the configurations, if, and only if, it generates descendants carrying all basic features (where the term 'basic' refers to their being used to represent solutions, hence constituting a representation basis in an algebraic sense) common to all parents. Notice that, if all parent configurations are identical, a respectful recombination operator is forced to return the same configuration as a descendant. This property is termed *purity*, and can be achieved even when the recombination operator is not generally respectful.

On the other hand, *assortment* represents the exploratory side of recombination. A recombination operator is said to be *properly assorting* if, and only if, it can generate descendants carrying any combination of compatible features taken from the parents. The assortment is said to be *weak* if this cannot be accomplished in a single recombination event, and further applications of the recombination operator on the offspring are required.

Finally, *transmission* is a very important property that captures the intuitive role of recombination. An operator is said to be transmitting if every feature exhibited by the offspring is present in at least one of the parents. Thus, a transmitting recombination operator combines the information present in the parents but does not introduce new information. This latter task is usually left to the mutation opera-

tor. For this reason, a non-transmitting recombination operator is said to introduce *implicit mutation*.

The three properties above suffice to describe the abstract input/output behavior of a recombination operator regarding some particular features. It provides a characterization of the possible descendants that can be produced by the operator. Nevertheless, there exist other aspects of the functioning of recombination that must be studied. In particular, it is interesting to consider how the construction of $\mathscr{S}_{desc}$ is approached.

First of all, a recombination operator is said to be *blind* if it has no other input than $\mathscr{S}_{par}$, i.e., it does not use any information from the problem instance. This definition is certainly very restrictive, and hence is sometimes relaxed to allow the recombination operator to use information regarding the problem constraints (so as to construct feasible descendants), and possibly the fitness values of configurations $y \in \mathscr{S}_{par}$ (so as to bias the generation of descendants toward the best parents). A typical example of a blind recombination operator is the classical *Uniform crossover* [181]. This operator is defined on search spaces $\mathscr{S} \equiv \Sigma^n$, i.e., strings of n symbols taken from an alphabet Σ. The construction of the descendant is done by randomly selecting at each position one of the symbols appearing in that position in any of the parents. This random selection can be totally uniform or can be biased according to the fitness values of the parents as mentioned before. Furthermore, the selection can be done so as to enforce feasibility (e.g., consider the binary representation of solutions in the 0-1 MKP). Notice that, in this case, the resulting operator is neither respectful nor transmitting in general.

The use of blind recombination operators has been usually justified on the grounds of not introducing excessive bias in the search algorithm, thus preventing extremely fast convergence to suboptimal solutions. This is questionable though. First, notice that the behavior of the algorithm is in fact biased by the choice of representation and the mechanics of the particular operators. Second, there exist widely known mechanisms (e.g., spatial isolation) to hinder these problems. Finally, it can be better to quickly obtain a suboptimal solution and restart the algorithm than using blind operators for a long time in pursuit of an asymptotically optimal behavior (not even guaranteed in most cases).

Recombination operators that use problem knowledge are commonly termed *heuristic* or *hybrid*. In these operators, problem information is utilized to guide the process of constructing the descendants. This can be done in a plethora of ways for each problem, so it is difficult to provide a taxonomy of heuristic recombination operators. Nevertheless, there exist two main aspects into which problem knowledge can be injected: the selection of the parental features that will be transmitted to the descendant, and the selection of non-parental features that will be added to it. A heuristic recombination operator can focus in one of these aspects, or in both of them simultaneously.

As an example of a heuristic recombination operator focusing on the first aspect, Dynastically Optimal Recombination (DOR) [34] can be mentioned. This operator explores the *dynastic potential* (i.e., the set of possible children) of the configurations being recombined, so as to find the best member of this set (notice that, since

configurations in the dynastic potential are entirely composed of features taken from any of the parents, this is a transmitting operator). This exploration is done using a subordinate complete algorithm, and its goal is thus to find the best combination of parental features giving rise to a feasible child. This can be accomplished using techniques such as branch and bound (BnB) or dynamic programming (see, e.g. [56]). This operator is monotonic in the sense that any child generated is at least as good as the best parent.

With regard to heuristic operators concentrating on the selection of non-parental features, one can cite the *patching-by-forma-completion* operators proposed by Radcliffe and Surry [159]. These operators are based on generating an incomplete child using a non-heuristic procedure (e.g., the RAR_ω operator [158]), and then completing the child either using a local hill climbing procedure restricted to non-specified features (*locally optimal forma completion*) or a global search procedure that finds the globally best solution carrying the specified features (*globally optimal forma completion*). Notice the similarity of this latter approach with DOR.

Finally, there exist some operators trying to exploit knowledge in both of the above aspects. A distinguished example is the *Edge Assembly Crossover* (EAX) [128]. EAX is a specialized operator for the TSP (both for symmetric and asymmetric instances) in which the construction of the child comprises two-phases: the first one involves the generation of an incomplete child via the so-called E-sets (subtours composed of alternating edges from each parent); subsequently, these subtours are merged into a single feasible subtour using a greedy repair algorithm. The authors of this operator reported impressive results in terms of accuracy and speed. It has some similarities with the recombination operator proposed in [117]. We can also mention the use of path relinking [58], a method based on creating a trajectory in the search space between the solutions being "recombined" and picking the best solution along that path.

A final comment must be made in relation to the computational complexity of recombination. It is clear that combining the features of several solutions is in general computationally more expensive than modifying a single solution (i.e., a mutation). Furthermore, the recombination operation will be usually invoked a large number of times. For this reason, it is convenient (and in many situations mandatory) to keep it at a low computational cost. A reasonable guideline is to consider an $O(N \log N)$ upper bound for its complexity, where N is the size of the input (the set S_{par} *and* the problem instance x). Such limit is easily affordable for blind recombination operators, which are called *crossover*, a reasonable name to convey their low complexity (yet not always used in this context). However, this limit can be relatively astringent in the case of heuristic recombination, mainly when epistasis (non-additive inter-feature influence on the fitness value) is involved. This admits several solutions depending upon the particular heuristic used. For example, DOR has exponential worst case behavior, but it can be made affordable by picking larger pieces of information from each parent (the larger the size of these pieces of information, the lower the number of them needed to complete the child) [33]. In any case, heuristic recombination operators provide better solutions than blind recombination operators, and hence they need not be invoked the same number of times.

Algorithm 2: A Population-Based Search Algorithm

```
1  Procedure Population-Based-Search-Engine;
2  begin
3      Initialize pop using GenerateInitialPopulation();
4      repeat
5          newpop ← GenerateNewPopulation(pop);
6          pop ← UpdatePopulation (pop, newpop);
7          if pop has converged then
8              pop ← RestartPopulation(pop);
9          end if
10     until TerminationCriterion();
11 end
```

Algorithm 3: Injecting high-quality solutions in the initial population.

```
1  Procedure GenerateInitialPopulation;
2  begin
3      Initialize pop using EmptyPopulation();
4      for j ← 1 to popsize do
5          i ← GenerateRandomConfiguration();
6          i ← Local-Search-Engine (i);
7          InsertInPopulation individual i to pop;
8      end for
9      return pop;
10 end
```

2.5 A Memetic Algorithm Template

In light of the above considerations, it is possible to provide a general template for a memetic algorithm. As mentioned in Subsection 2.3, this template is very similar to that of a local search procedure acting on a set of $|pop| \geqslant 2$ configurations. This is shown in Algorithm 2.

This template requires some explanation. First of all, the GenerateInitialPopulation procedure is responsible for creating the initial set of $|pop|$ configurations. This can be done by simply generating $|pop|$ random configurations or by using a more sophisticated seeding mechanism (for instance, some constructive heuristic), by means of which high-quality configurations are injected in the initial population [180]. Another possibility is to use the Local-Search-Engine presented in Subsection 2.3 (as shown in Algorithm 3) or any other randomized constructive algorithm for that matter. For example, a Greedy Randomized Adaptive Search Procedure (GRASP) [162, 163] mechanism was used in [54], and Beam Search [190] was used in [55].

As for the TerminationCriterion function, it can be defined very similarly to Local Search, i.e., setting a limit on the total number of iterations, reaching a maximum number of iterations without improvement or performing a certain number of population restarts, etc.

Algorithm 4: The pipelined GenerateNewPopulation procedure.

```
1  Procedure GenerateNewPopulation (pop);
2  begin
3      buffer^0 ← pop;
4      for j ← 1 to n_op do
5          Initialize buffer^j using EmptyPopulation();
6      end for
7      for j ← 1 to n_op do
8          S^j_par ← ExtractFromBuffer (buffer^{j-1}, arity^j_in);
9          S^j_desc ← ApplyOperator (op^j, S^j_par);
10         for z ← 1 to arity^j_out do
11             InsertInPopulation individual S^j_desc[z] to buffer^j;
12         end for
13     end for
14     return buffer^{n_op};
15 end
```

The GenerateNewPopulation procedure is at the core of memetic algorithms. Essentially, this procedure can be seen as a pipelined process comprising n_{op} stages. Each of these stages consists of applying a *variation* (or *reproductive*) operator op^j by taking $arity^j_{in}$ configurations from the previous stage to produce $arity^j_{out}$ new configurations. This pipeline is restricted to have $arity^1_{in} = popsize$. The whole process is sketched in Algorithm 4.

This template for the GenerateNewPopulation procedure is typically instantiated in GAs by letting $n_{op} = 3$, using a selection, a recombination, and a mutation operator. Traditionally, mutation is applied after recombination, i.e., on each child generated by the recombination operator. However, if a heuristic recombination operator is being used, it may be more convenient to apply mutation *before* recombination. Since the purpose of mutation is simply to introduce new features in the configuration pool, in advance is possible in this case. Furthermore, the *smart* feature combination performed by the heuristic operator would not be disturbed this way.

This situation is slightly different in MAs. In this case, it is very common to let $n_{op} = 4$, inserting a Local-Search-Engine right after applying op^2 and op^3 (respectively recombination and mutation). Due to the local optimization performed after mutation, their combined effect (i.e., mutation + local search) cannot be regarded as a simple disruption of a computationally-demanding recombination. Note also that the interplay between mutation and local search requires the former to be different than the neighborhood structure used in the latter; otherwise mutations can be readily reverted by local search, and their usefulness would be negligible.

The UpdatePopulation procedure is used to reconstruct the current population using the old population pop and the newly generated population $newpop$. Borrowing the terminology from the evolution strategy [161, 168] community, there exist two main possibilities to carry on this reconstruction: the *plus* strategy and the *comma* strategy. In the former, the current population is constructed taken the best $popsize$ configurations from $pop \cup newpop$. For the latter, the best $popsize$ configurations

are taken just from *newpop*. In this case, it is required to have $|newpop| > popsize$, so as to put some selective pressure on the process (the bigger the $|newpop|/popsize$ ratio, the stronger the pressure). Otherwise, the search would reduce to a random wandering through $\mathscr{S}$.

There are a number of studies regarding appropriate choices for the UpdatePopulation procedure (see e.g., [6]). As a general guideline, the comma strategy is usually regarded as less prone to stagnation, with the ratio $|newpop|/popsize \simeq 6$ being a common choice [7]. Nevertheless, this option can be somewhat computationally expensive if the guiding function is complex and time-consuming. Another common alternative is to use a plus strategy with a low value of $|newpop|$, analogous to the so-called *steady-state* replacement strategy in GAs [188]. This option usually provides a faster convergence to high-quality solutions. However, care has to be taken with premature convergence to suboptimal regions of the search space, i.e., all configurations in the population being very similar to each other, hence hindering the exploration of other regions of $\mathscr{S}$.

The above consideration about premature convergence leads to the last component of the template shown in Algorithm 2, the restarting procedure. First of all, it must be decided whether the population has degraded or has not. To do so, it is possible to use some measure of information diversity in the population such as Shannon's entropy [38]. If this measure falls below a predefined threshold, the population is considered to be in a degenerate state. This threshold depends upon the representation (number of values per variable, constraints, etc.) and hence must be determined in an *ad-hoc* fashion. A different possibility is using a probabilistic approach to determine with a desired confidence that the population has converged. For example, in [76] a Bayesian approach is presented for this purpose.

Once the population is considered to be at a degenerate state, the restart procedure is invoked. Again, this can be implemented in a number of ways. A very typical strategy is to keep a fraction of the current population (this fraction can be as small as one solution, the current best), and substituting the remaining configurations with newly generated (from scratch) solutions, as shown in Algorithm 5.

The procedure shown in Algorithm 5 is also known as the *random-immigrant* strategy [20]. Another possibility is using the previous search history [179] or activate a *strong* or *heavy* mutation operator in order to drive the population away from its current location in the search space. Both options have their advantages and disadvantages. For example, when using the random-immigrant strategy, one has to take some caution to prevent the preserved configurations to take over the population (this can be achieved by putting a low selective pressure, at least in the first iterations after a restart). As to the heavy mutation strategy, one has to achieve a tradeoff between an excessively strong mutation that would destroy any information contained in the current population, and a not so strong mutation that would cause the population to converge again in a few iterations.

Algorithm 5: The RestartPopulation procedure.

```
1  Procedure RestartPopulation (pop);
2  begin
3      Initialize newpop using EmptyPopulation();
4      #preserved ← popsize · %preserve;
5      for j ← 1 to #preserved do
6          i ← ExtractBestFromPopulation(pop);
7          InsertInPopulation individual i to newpop;
8      end for
9      for j ← #preserved + 1 to popsize do
10         i ← GenerateRandomConfiguration();
11         i ← Local-Search-Engine (i);
12         InsertInPopulation individual i to newpop;
13     end for
14     return newpop;
15 end
```

2.6 Designing an Effective Memetic Algorithm

The general template of MAs depicted in the previous section must be instantiated with precise components in order to be used for solving a specific problem. This instantiation has to be done carefully so as to obtain an effective optimization tool. We will address some design issues in this section.

A first obvious remark is that there exist no general approach for the design of effective MAs. This observation is based on different proofs depending on the precise definition of *effective* in the previous statement. Such proofs may involve classical complexity results and conjectures if 'effective' is understood as 'polynomial-time', or the NFL Theorem if we consider a more general set of performance measures, and even Computability Theory if we relax the definition to arbitrary decision problems. For these reasons, we can only define several *design heuristics* that will likely result in good-performing MAs, but without explicit guarantees for this.

This said, MAs are commonly implemented as evolutionary algorithms endowed with an independent search component, sometimes provided by a local search mechanism (recall previous section), and as such can benefit from the theoretical corpus available for EAs. This is particularly applicable to some basic aspects such as the representation of solutions in terms of meaningful information units [37, 159]. Focusing now on more specific aspects of MAs, the first consideration that must be clearly taken into account is the interplay among the local search component and the remaining operators, mostly with respect to the characteristics of the search landscape. A good example of this issue can be found in the work of Merz and Freisleben on the TSP [48]. They consider the use of a highly intensive local search procedure –the Lin-Kernighan heuristic [103]– and note that the average distance between local optima is similar to the average distance between a local optimum and the global optimum. For this reason, they introduce a distance-preserving crossover (DPX) operator that generate offspring whose distance from the parents is the same

as the distance between the parents themselves. Such an operator is likely to be less effective if a not-so-powerful local improvement method, e.g., 2-opt, was used, inducing a different distribution of local optima.

Another important choice refers to the learning model used. The most common option is to use a Lamarckian model, whereby an improved solution is sought via local search and the corresponding genotypic changes are retained in the solution. However, there also exists the possibility of using a Baldwinian model, in which the improved solution is only used for the purposes of fitness computation, but the the actual solution is not changed at all. This might be useful in order to avoid local optima while converging to the global optimum [57, 88, 194]; see also [189] for a classical analysis of these two strategies in optimization.

In addition to the particular choice (or choices) of local search operator, there remains the issue of determining an adequate parameterization for the procedure, namely, how much effort must be spent on each local search, how often the local search must be applied, and –were it not applied to every new solution generated– how to select the solutions that will undergo local improvement. Regarding the first two items, there exists theoretical evidence [98, 176] that an inadequate parameter setting can turn the algorithmic solution from easily solvable to non-polynomially solvable. Besides, there are obvious practical limitations in situations where the local search and/or the fitness function is computationally expensive. This fact admits different solutions. On the one hand, the use of surrogates (i.e., fast approximate models of the true function) to accelerate evolution is an increasingly popular option in such highly demanding problems [63, 101, 186, 187, 196]. On the other hand, partial lamarckism [23, 73, 150], where not every individual is subject to local search, is commonly used as well. The precise value for the local search application probability (or multiple values when more than one local search procedure is available) largely depends on the problem under consideration [81], and its determination is in many cases an art. For this reason, adaptive and self-adaptive mechanisms have been defined in order to let the algorithm learn what the most appropriate setting is (see Section 3.4). The interested reader is referred to [177, 178] for a more in-depth analysis of the balance between the local and global (i.e., population-based) components of the memetic algorithm.

As to the selection of individuals that will undergo local search, the most common options are random-selection, and fitness-based selection, where only the best individuals are subject to local improvement. Nguyen *et al.* [137] also consider a 'stratified' approach, in which the population is sorted and divided into k levels (k being the number of local search applications), and one individual per level is randomly selected. Their experimentation on some continuous functions indicates that this strategy and improve-the-best (i.e., applying local search to the best individuals) provide better results than random selection. Such strategies can be readily deployed on a structured MA as defined by Moscato *et al.* [10, 15, 47, 109, 124], where good solutions flow upwards within a tree-structured population, and layers are explicitly available. Other population management strategies are possible as well, see [14, 154, 155, 174].

3 Algorithmic Extensions of Memetic Algorithms

The algorithmic template and design guidelines described in the previous section can characterize most basic incarnations of MAs, namely population-based algorithms endowed with static local search for single-objective optimization. However, more sophisticated approaches can be conceived, and are certainly required in certain applications. This section is aimed at providing an overview of more advanced algorithmic extensions used in the MA realm.

3.1 Multiobjective Memetic Algorithms

Multiobjective problems are frequent in real-world applications. Rather than having a single objective to be optimized, the solver is faced with multiple, partially conflicting objectives. As a result, there is no *a priori* single optimal solution but rather a collection of optimal solutions, providing different trade-offs among the objectives considered. In this scenario, the notion of Pareto-dominance is essential: given two solutions $s, s' \in sol_P(x)$, s is said to dominate s' if it is better than s' in at least one of the objectives, and it is no worse in the remaining ones. This clearly induces a partial order $\prec_P$, since given two solutions it may be the case that none of them dominates the other. This collection of optimal solutions is termed the optimal Pareto front, or the optimal non-dominated front.

Population-based search techniques, in particular evolutionary algorithms (EAs), are naturally fit to deal with multiobjective problems, due to the availability of a population of solutions which can approach the optimal Pareto front from different directions. There is an extensive literature on the deployment of EAs in multiobjective settings, and the reader is referred to [21, 22, 41, 197], among others, for more information on this topic. MAs can obviously benefit from this corpus of knowledge. However, MAs typically incorporate a local search mechanism, and it has to be adapted to the multiobjective setting as well. This can be done in different ways [92], which can be roughly classified into two major classes: scalarizing approaches, and Pareto-based approaches. Scalarizing approaches are based on the use of some aggregation mechanism to combine the multiple objectives into a single scalar value. This is usually done using a linear combination of the objective values, with weights that are either fixed (at random or otherwise) for the whole execution of the local search procedure [183], or adapted as the local search progresses [65]. With regard to Pareto-based approaches, the notion of Pareto-dominance is cosidered for deciding transitions among neighboring solutions, typically coupled with the use of some measure of crowding to spread the search, e.g, [90].

A full-fledged multiobjective MA (MOMA) is obtained by appropriately combining population-based and local search-based components for multiobjective optimization. Again, the strategy used in the local search mechanism can be used to classify most MOMAs. On one hand, we have aggregation approaches. Thus, two proposals due to Ishibuchi and Murata [79, 80] and Jaszkiewicz [82, 83] are based

on the use of random scalarization each time a local search is to be used. Alternatively, a single-objective local search could be used to optimize individual objectives [78]. Ad hoc mating strategies based on the particular weights chosen at each local search invocation (whereby the solutions to be recombined are picked according to these weights) are used as well. A related approach –including the on-line adjustment of scalarizing weights– is followed by Guo *et al.* [64, 65, 66]. On the other hand, we have Pareto-based approaches. In this line, a MA based on PAES (Pareto Archived Evolution Strategy) was defined by Knowles and Corne [93, 91]. More recently, a MOMA based on particle swarm optimization (PSO) has been defined by Liu *et al.* [100, 107]. In this algorithm, an archive of non-dominated solutions is maintained and randomly sampled to obtain reference points for particles. A different approach is used by Schuetze *et al.* [165] for numerical-optimization problems. The continuous nature of solution variables allows using their values for computing search directions. This fact is exploited in their local search procedure (HCS for Hill Climber with Sidestep) to direct the search toward specific regions (e.g., along the Pareto front) when required. We refer to [84] for a more in-depth discussion on multiobjective MAs.

3.2 Continuous Optimization

Continuous optimization problems are defined on a dense search space [184], typically by some subset of the n-fold Cartesian product $\mathbb{R}^n$. Many problems have decision variables of this continuous nature and hence continuous optimization is a realm of paramount importance. Throughout previous sections, MAs were admittedly described with a discrete background in mind. Indeed, discrete optimization problems put to test the skills of the algorithmic designer in the sense that the difficulty of solving a particular problem and the effectiveness of the solver depend on the precise instantiation of notions such as the neighborhood relation. This said, most of the ideas and concepts sketched before for discrete optimization are also applicable to continuous optimization. Of course, in this realm there is a natural notion of neighborhood of a point s given by open balls $B_d(s) = \{s' : ||s - s'|| < d\}$, i.e., those points located within distance d of x, for a suitable distance metric (typically, but not necessarily, the Euclidean distance – see [43]).

The different components of a classic MA, namely the population-based engine and the local search technique, must be adapted to deal with this new domain of solutions. Regarding the former, there is plenty of literature on how to adapt the variation operators to tackle continuous optimization [69, 70, 108, 192] and actually some evolutionary computation families lend themselves naturally to this kind of optimization [13, 175]. Typical options with regard to the recombination operator are the following (assuming for the sake of notation that parental solutions $s = \langle s_1, \ldots, s_n \rangle$ and $s' = \langle s'_1, \ldots, s'_n \rangle$ are being recombined to obtain $u = \langle u_1, \ldots, u_n \rangle$):

- use a discrete approach and create the offspring by using the precise values the decision variable have in the parental solutions, i.e., $u_i \in \{s_i, s'_i\}$.

- use some arithmetical operation to combine the values of homologous variables in the parental solutions, e.g., compute an average: $u_i = (s_i + s'_i)/2$.
- use some sampling procedure within some hyperrectangle, hyperellipse, or other suitable hypersurface defined by the parental solutions, e.g., $u_i \in [m_i, M_i]$ where $m_i = \min(s_i, s'_i) - \alpha d_i$, $M_i = \max(s_i, s'_i) + \alpha d_i$, $d_i = |s_i - s'_i|$, $\alpha \geqslant 0$.

The situation is more flexible when multiparent recombination is used. In this case, other possibilities exist in addition to the previous methods, such as utilizing some subset of the parental solutions to create a hypersurface and using some projection technique to create the offspring, much like it is done in the Nelder-Mead method [131]. For mutation, it is typically accomplished by some additive or multiplicative perturbation to variable values, obtained by means of some predefined distribution such as uniform, Gaussian or Cauchy, just to cite some of the most common examples. The extent of the perturbation is a parameter than can be subject to adaptation during the run (cf. Section 3.4)

Regarding the local search component, there are many techniques that can be used for this purpose, just by adapting the definition of neighborhood as mentioned before and using some kind of gradient ascent, possibly modulated with some mechanism to escape from local optima (as it is done in e.g., simulated annealing); see [145] for a more detailed discussion of these. One particular issue worth mentioning in connection with local search is the fact that, unlike many typical discrete optimization scenarios in which the objective function can be decomposed in order to isolate the effect caused by the modification of a certain decision variable (i.e., the fitness value of the modified solution is $f(u) = f(s) + \Delta(s,u)$ for some function $\Delta(s,u)$ which is computationally cheaper to compute than $f(u)$), continuous optimization problem usually exhibits many couplings and non-linearities that preclude or at least limit such approaches. This affects the cost of the local search component which in turn may influence the optimal balance between local and global search in the algorithm. Some authors [114] have proposed to store the state of the local search along with each solution it is applied to, so that further applications of the local improvement routine resume from this state. We refer to [31] for a more detailed discussion of design issues in MAs for continuous optimization.

3.3 Memetic Computing Approaches

Memes were introduced in Section 1 as units of imitation. In a computational context (and more precisely with regard to memetic algorithms), they acquire a new meaning though. In this sense, a first interpretation would be to use the notion of meme as a high-level non-genetic pattern of information, that is, the carrier particle of individual learning. From the standpoint of classical MAs, this role is implemented by local improvement procedures. Thus, the particular choice of a local search procedure (a simple heuristic rule, hill climbing, simulated annealing, etc.) plus the corresponding parameterization can be regarded as the implicit definition of

a fixed meme. However, earlier works already anticipated that these memes needed not be static, but change dynamically during the search. Quoting [118]:

> *"It may be possible that a future generation of MAs will work in at least two levels and two timescales. In the short-timescale, a set of agents would be searching in the search space associated to the problem while the long-time scale adapts the heuristics associated with the agents."*

The first steps in this direction were taken in [95, 169] by including an explicit representation of memes alongside solutions, and having them evolve. This has given rise to the notion of memetic computing, which can be defined as a broad discipline that focuses on the use of dynamic complex computational structures composed of interacting modules (the memes) which are harmoniously coordinated to solve particular problems [132]; see also [19, 147].

There are obvious connections here with the notion of adaptive hyperheuristics [16, 18, 35, 87], particularly in the context of Meta-Lamarckian learning [136, 146], in which a collection of memes are available and some mechanism is used to decide which one to apply and when (be it using information on the previous applications of each meme or gathering population statistics [135]). Some other possibilities can be used though. As mentioned above, memes can be explicitly represented (this can range from a simple parameterization of a generic template –i.e., the neighborhood definition of a local search procedure, the pivot rule, etc.– to the full definition of the local improver using mechanisms akin to genetic programming) and self-adapt during the execution of the algorithm, either as a part of solutions [96, 97, 140, 171] or in a separate population [172]. Furthermore, it is possible to aggregate simple memes into larger compounds or *memeplexes* [19] in order to attain synergistic cooperation and improved search efficiency.

3.4 Self-⋆ Memetic Algorithms

When some design guidelines were given in Section 2.6, the fact that these were heuristics that ultimately relied on the available problem knowledge was stressed. This is not a particular feature of MAs, but affects the field of metaheuristics as a whole. Indeed, one of the keystones in practical metaheuristic problem-solving is the necessity of customizing the solver for the problem at hand [32]. Therefore, it is not surprising that attempts to transfer a part of this tuning effort to the metaheuristic technique itself have been common. Such attempts can take place at different levels, or can affect different components of the algorithm. The first –and more intuitive one– is the parametric level involving the numerical values of parameters, such as the operator application rates. Examples of this can be found in early EAs, see for example [3, 12, 39, 167]. An overview of these approaches (actually broader in scope, covering more advanced topics than parameter adaptation) can be found in [170]. Focusing specifically on MAs, this kind of adaptation has been applied in [8, 69, 112, 113, 148].

The explicit procesing of memes described in the previous section is actually a further step in the direction of promoting the autonomous functioning of the algorithm. Indeed, from a very general point of view this connects to the idea of autonomic computing [72], that tries to transfer to the computing realm the idea of the autonomic nervous system carrying essential functions without conscious control. In this line, the umbrella term *self-⋆ properties* [5] is used to describe the capacity of self-management in complex computational systems [75]. Self-parameterization attempts mentioned previously fall within the scope of self-⋆ properties, and so does the explicit handling of memes described in previous section, which can be considered a case of self-generating search strategies. As a matter of fact, both approaches constitute examples of self-optimization [9], because they aim at improving the capabilities of the algorithm for carrying out its functions (which is in turn solving the objective problem).

Self-⋆ properties can encompass other advanced capabilities beyond self-optimization such as self-scaling or self-healing. The former refers to the ability of the system to react efficiently to changes in its scale parameters, be it related to changes in the scale of the problem being solved, in the scale of the computational resources available, or in other circumstance or combination of circumstances of the computation. Such capability may involve some form of self-configuration in order to accomplich the objective of the computation in the most effective way in light of the change of scale. An example can be found in the domain of island-based MAs [138] deployed in unstable distributed environments [28]: if the computational substrate is composed of processing nodes whose availability fluctuate, the algorithm may face uncontrollable reductions or increments of the computational resources (i.e., some islands may appear, other islands may disappear). As a reaction, the algorithm may attempt to resize the islands and balance them out, so that the population size is affected as little as possible [141]. The second property, namely self-healing, is also relevant in this context: it aims to maintain and restore system attributes that may have been affected by internal or external actions, i.e., self-healing externally infringed damage. In the volatile computational scenario depicted, such damage is caused by the loss of information and the disruptions in connectivity caused by the disappearance of islands [142]. To tackle these issues, the algorithm may use self-sampling (using a probabilistic model of the population –much like it is done in estimation of distribution algorithms [99, 152]– in order to enlarge it in a sensible way when required) and self-rewiring in order to create new connectivity links and prevent the network from becoming disconnected. It must also be noted as an aside that very traditional techniques commonly used when metaheuristic face contrained problems, namely using a repair function to restore the feasibility of solutions [111], can also fall within the scope of self-repairing approaches.

3.5 Memetic Algorithms and Complete Techniques

The combination of exact techniques with metaheuristics is an increasingly popular approach. Focusing on local search techniques, Dumitrescu and Stüztle [45] have provided a classification of methods in which exact algorithms are used to strengthen local search, i.e., to explore large neighborhoods, to solve exactly some subproblems, to provide bounds and problem relaxations to guide the search, etc. Some of these combinations can also be found in the literature on population-based methods. For example, exact techniques –such as BnB [34] or dynamic programming [52] among others– have been used to perform recombination (recall Section 2.4), and approaches in which exact techniques solved some subproblems provided by EAs date back to 1995 [25]; see also [46] for a large list of references regarding local search/exact hybrids.

Puchinger and Raidl [156] have provided a classification of this kind of hybrid techniques in which algorithmic combinations are either collaborative (sequential or intertwined execution of the combined algorithms) or integrative (one technique works inside the other one, as a subordinate). Some of the exact/metaheuristic hybrid approaches defined before are clearly integrative –i.e., using an exact technique to explore neighborhoods. Further examples are the use of BnB in the decoding process [157] of a genetic algorithm (i.e., exact method within a metaheuristic technique), or the use of evolutionary techniques for the strategic guidance of BnB [94] (metaheuristic approach within an exact method).

With regard to collaborative combinations, a sequential approach in which the execution of a MA is followed by a branch-and-cut method can be found in [89]. Intertwined approaches are also popular. For example, Denzinger and Offerman [42] combine genetic algorithms and BnB within a parallel multi-agent system. These two algorithms also cooperate in [25, 50], the exact technique providing partial promising solutions, and the metaheuristic returning improved bound. A related approach involving beam search and full-fledged MAs can be found in [51, 53, 55]; see also [27] for a broader overview of this kind of combinations.

It must be noted that most hybrid algorithms defined so far that involve exact techniques and metaheuristics are not complete, in the sense that they do not guarantee an optimal solution (an exception is the proposal of French *et al.* [49], combining an integer-programming BnB approach with GAs for MAX-SAT). Thus, the term 'complete MA' may be not fully appropriate. Nevertheless, many of these hybrids can be readily adapted for completeness purposes, although obviously time and/or space requirements will grow faster-than-polynomial in general.

4 Applications of Memetic Algorithms

Applications are the "raison d'être" of memetic algorithms. Their functioning philosophy, namely incorporating and exploiting knowledge of the problem being solved, presumes they are designed with a target problem in mind. This section will

provide an overview of the numerous applications of MAs. The reader may actually be convinced of the breadth of these applications by noting the existence of a number of domain-specific reviews of MAs. As a matter of fact, we have organized this section as a meta-review of applications, providing pointers to these compilations rather than to individual specific applications. This is done in Table 1.

Table 1 Application surveys of memetic algorithms. General overviews are also referenced with respect to the subdomains in which they are internally structured.

Domain	**References**
General overviews	[30, 68, 120, 121, 122, 123, 132]
Bioinformatics	[11, 122]
Combinatorial optimization	[120, 121, 122, 123]
Electronics and telecommunications	[29, 30, 120, 122, 123]
Engineering	[17, 30]
Machine learning and knowledge discovery	[120, 122, 123]
Molecular optimization	[120, 123]
Planning, scheduling, and timetabling	[26, 122, 123, 125]

Any of the reviews mentioned are far from exhaustive since new applications are being developed continuously. However, they are intended to illustrate the practical impact of these optimization techniques, pointing out some selected compilations from these well-known application areas. For further information about MA applications, we suggest querying bibliographical databases or web browsers for the keywords *'memetic algorithms'* and *'hybrid genetic algorithms'*.

5 Conclusions

We believe that the future looks good for MAs. This belief is based on the following. First of all, MAs are showing a great record of efficient implementations, providing very good results for practical problems, as the reader may have noted in Section 4. We also have reasons to believe that we are close to some major leaps forward in our theoretical understanding of these techniques, including for example the worst-case and average-case computational complexity of recombination procedures. On the other hand, the ubiquitous nature of distributed systems is likely to boost the deployment of MAs on large-scale, computationally demanding optimization problems.

We also see as a healthy sign the systematic development of other particular optimization strategies. If any of the simpler metaheuristics (SA, TS, VNS, GRASP, etc.) performs the same as a more complex method (GAs, MAs, Ant Colonies, etc.), an "elegance design" principle should prevail and we must either resort to the simpler method, or to the one that has less free parameters, or to the one that is easier to implement. Such a fact should challenge us to adapt complex methodologies to beat simpler heuristics, or to check if that is possible at all. An unhealthy sign of

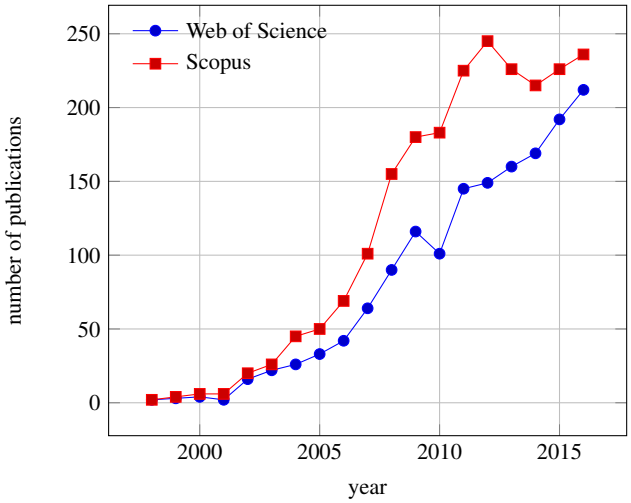

Fig. 1 Number of publications obtained by querying the Web of Science and Scopus with the term "memetic algorithm" (1998-2016)

current research, however, are the attempts to encapsulate metaheuristics in watertight compartments. Fortunately, such attempts are becoming increasingly less frequent. Indeed, combinations of MAs with other metaheuristics such as differential evolution [134, 143, 164, 182], estimation of distribution algorithms [2, 139, 185], particle swarm optimization [74, 100, 104, 105, 106, 107, 149, 153, 173, 195], or ant-colony optimization [102] are not unusual nowadays. Furthermore, there is a clear ascending trend in the number of publications related to MAs, as shown in Fig. 1. Thus, as stated before, the future looks promising for MAs.

Acknowledgements This chapter is an update of [122], refurbished with new references and the inclusion of sections on timely topics which were not fully addressed in the previous editions. Pablo Moscato acknowledges funding of his research by the Australian Research Council grants Future Fellowship FT120100060 and Discovery Project DP140104183. He also acknowledges previous support by FAPESP, Brazil (1996-2001). Carlos Cotta acknowledges the support of Spanish Ministry of Economy and Competitiveness and European Regional Development Fund (FEDER) under project EphemeCH (TIN2014-56494-C4-1-P).

References

1. Aldous, D., Vazirani, U.: "Go with the winners" algorithms. In: Proceedings of the 35th Annual Symposium on Foundations of Computer Science, pp. 492–501. IEEE Press, Los Alamitos, CA (1994)
2. Amaya, J.E., Cotta, C., Fernández, A.J.: Cross entropy-based memetic algorithms: An application study over the tool switching problem. Int. J. Computational Intelligence Systems

6(3), 559–584 (2013)
3. Angeline, P.: Morphogenic evolutionary computations: Introduction, issues and example. In: J.M. et al. (ed.) Fourth Annual Conference on Evolutionary Programming, pp. 387–402. MIT Press, Cambridge, Massachusetts (1995)
4. Axelrod, R., Hamilton, W.: The evolution of cooperation. Science **211**(4489), 1390–1396 (1981)
5. Babaoğlu, Ö., Jelasity, M., Montresor, A., Fetzer, C., Leonardi, S., van Moorsel, A., van Steen, M. (eds.): Self-star Properties in Complex Information Systems, *Lecture Notes in Computer Science*, vol. 3460. Springer-Verlag, Berlin Heidelberg (2005)
6. Bäck, T.: Evolutionary Algorithms in Theory and Practice. Oxford University Press, New York (1996)
7. Bäck, T., Hoffmeister, F.: Adaptive search by evolutionary algorithms. In: W. Ebeling, M. Peschel, W. Weidlich (eds.) Models of Self-organization in Complex Systems, no. 64 in Mathematical Research, pp. 17–21. Akademie-Verlag (1991)
8. Bambha, N., Bhattacharyya, S., Teich, J., Zitzler, E.: Systematic integration of parameterized local search into evolutionary algorithms. IEEE Transactions on Evolutionary Computation **8**(2), 137–155 (2004)
9. Berns, A., Ghosh, S.: Dissecting self-$\star$ properties. In: Third IEEE International Conference on Self-Adaptive and Self-Organizing Systems - SASO 2009, pp. 10–19. IEEE Press, San Francisco, CA (2009)
10. Berretta, R., Cotta, C., Moscato, P.: Enhancing the performance of memetic algorithms by using a matching-based recombination algorithm: Results on the number partitioning problem. In: M. Resende, J. Pinho de Sousa (eds.) Metaheuristics: Computer-Decision Making, pp. 65–90. Kluwer Academic Publishers, Boston MA (2003)
11. Berretta, R., Cotta, C., Moscato, P.: Memetic algorithms in bioinformatics. In: F. Neri, C. Cotta, P. Moscato (eds.) Handbook of Memetic Algorithms, *Studies in Computational Intelligence*, vol. 379, pp. 261–271. Springer, Berlin Heidelberg (2012)
12. Beyer, H.: Toward a theory of evolution strategies: Self-adaptation. Evolutionary Computation **3**(3), 311–348 (1995)
13. Beyer, H.G., Schwefel, H.P.: Evolution strategies – A comprehensive introduction. Natural Computing **1**(1), 3–52 (2002)
14. Boudia, M., Prins, C., Reghioui, M.: An effective memetic algorithm with population management for the split delivery vehicle routing problem. In: T. Bartz-Beielstein, et al. (eds.) Hybrid Metaheuristics 2007, *Lecture Notes in Computer Science*, vol. 4771, pp. 16–30. Springer-Verlag (2007)
15. Buriol, L., França, P., Moscato, P.: A new memetic algorithm for the asymmetric traveling salesman problem. Journal of Heuristics **10**(5), 483–506 (2004)
16. Burke, E., Kendall, G., Soubeiga, E.: A tabu search hyperheuristic for timetabling and rostering. Journal of Heristics **9**(6), 451–470 (2003)
17. Caponio, A., Neri, F.: Memetic algorithms in engineering and design. In: F. Neri, C. Cotta, P. Moscato (eds.) Handbook of Memetic Algorithms, *Studies in Computational Intelligence*, vol. 379, pp. 241–260. Springer, Berlin Heidelberg (2012)
18. Chakhlevitch, K., Cowling, P.: Hyperheuristics: Recent developments. In: C. Cotta, M. Sevaux, K. Sörensen (eds.) Adaptive and Multilevel Metaheuristics, *Studies in Computational Intelligence*, vol. 136, pp. 3–29. Springer-Verlag, Berlin Heidelberg (2008)
19. Chen, X., Ong, Y.S.: A conceptual modeling of meme complexes in stochastic search. IEEE Transactions on Systems, Man and Cybernetics, Part C: Applications and Reviews **42**(5), 612–625 (2012)
20. Cobb, H., Grefenstette, J.: Genetic algorithms for tracking changing environments. In: S. Forrest (ed.) Proceedings of the Fifth International Conference on Genetic Algorithms, pp. 529–530. Morgan Kaufmann, San Mateo, CA (1993)
21. Coello Coello, C., Lamont, G.: Applications of Multi-Objective Evolutionary Algorithms. World Scientific, New York (2004)

22. Coello Coello, C., Van Veldhuizen, D., Lamont, G.: Evolutionary Algorithms for Solving Multi-Objective Problems, *Genetic Algorithms and Evolutionary Computation*, vol. 5. Kluwer Academic Publishers (2002)
23. Cotta, C.: Memetic algorithms with partial lamarckism for the shortest common supersequence problem. In: J. Mira, J. Álvarez (eds.) Artificial Intelligence and Knowledge Engineering Applications: a Bioinspired Approach, *Lecture Notes in Computer Science*, vol. 3562, pp. 84–91. Springer-Verlag, Berlin Heidelberg (2005)
24. Cotta, C., Alba, E., Troya, J.: Stochastic reverse hillclimbing and iterated local search. In: Proceedings of the 1999 Congress on Evolutionary Computation, pp. 1558–1565. IEEE, Washington D.C. (1999)
25. Cotta, C., Aldana, J., Nebro, A., Troya, J.: Hybridizing genetic algorithms with branch and bound techniques for the resolution of the TSP. In: D. Pearson, N. Steele, R. Albrecht (eds.) Artificial Neural Nets and Genetic Algorithms 2, pp. 277–280. Springer-Verlag, Wien New York (1995)
26. Cotta, C., Fernández, A.: Memetic algorithms in planning, scheduling, and timetabling. In: K. Dahal, K. Tan, P. Cowling (eds.) Evolutionary Scheduling, *Studies in Computational Intelligence*, vol. 49, pp. 1–30. Springer-Verlag (2007)
27. Cotta, C., Fernández Leiva, A.J., Gallardo, J.E.: Memetic algorithms and complete techniques. In: F. Neri, C. Cotta, P. Moscato (eds.) Handbook of Memetic Algorithms, *Studies in Computational Intelligence*, vol. 379, pp. 189–200. Springer, Berlin Heidelberg (2012)
28. Cotta, C., Fernández-Leiva, A.J., Fernández de Vega, F., Chávez, F., Merelo, J.J., Castillo, P.A., Bello, G., Camacho, D.: Ephemeral computing and bioinspired optimization - challenges and opportunities. In: 7th International Joint Conference on Evolutionary Computation Theory and Applications, pp. 319–324. Lisboa, Portugal (2015)
29. Cotta, C., Gallardo, J., Mathieson, L., Moscato, P.: A contemporary introduction to memetic algorithms. In: Wiley Encyclopedia of Electrical and Electronic Engineering, pp. 1–15. Wiley (2016). DOI 10.1002/047134608X.W8330
30. Cotta, C., Mathieson, L., Moscato, P.: Memetic algorithms. In: M. Resende, R. Marti, P. Pardalos (eds.) Handbook of Heuristics. Springer-Verlag (2015)
31. Cotta, C., Neri, F.: Memetic algorithms in continuous optimization. In: F. Neri, C. Cotta, P. Moscato (eds.) Handbook of Memetic Algorithms, *Studies in Computational Intelligence*, vol. 379, pp. 121–134. Springer (2012)
32. Cotta, C., Sevaux, M., Sörensen, K.: Adaptive and Multilevel Metaheuristics, *Studies in Computational Intelligence*, vol. 136. Springer-Verlag, Berlin Heidelberg (2008)
33. Cotta, C., Troya, J.: On the influence of the representation granularity in heuristic forma recombination. In: J. Carroll, E. Damiani, H. Haddad, D. Oppenheim (eds.) ACM Symposium on Applied Computing 2000, pp. 433–439. ACM Press (2000)
34. Cotta, C., Troya, J.: Embedding branch and bound within evolutionary algorithms. Applied Intelligence **18**(2), 137–153 (2003)
35. Cowling, P., Kendall, G., Soubeiga, E.: A hyperheuristic aproach to schedule a sales submit. In: E. Burke, W. Erben (eds.) Third International Conference on Practice and Theory of Automated Timetabling III - PATAT 2000, *Lecture Notes in Computer Science*, vol. 2079, pp. 176–190. Springer-Verlag, Berlin Heidelberg (2000)
36. Culberson, J.: On the futility of blind search: An algorithmic view of "no free lunch". Evolutionary Computation **6**(2), 109–128 (1998)
37. Davidor, Y.: Epistasis Variance: Suitability of a Representation to Genetic Algorithms. Complex Systems **4**(4), 369–383 (1990)
38. Davidor, Y., Ben-Kiki, O.: The interplay among the genetic algorithm operators: Information theory tools used in a holistic way. In: R. Männer, B. Manderick (eds.) Parallel Problem Solving From Nature II, pp. 75–84. Elsevier Science Publishers B.V., Amsterdam (1992)
39. Davis, L.: Handbook of Genetic Algorithms. Van Nostrand Reinhold Computer Library, New York (1991)
40. Dawkins, R.: The Selfish Gene. Clarendon Press, Oxford (1976)
41. Deb, K.: Multi-Objective Optimization Using Evolutionary Algorithms. John Wiley & Sons, Chichester, UK (2001)

42. Denzinger, J., Offermann, T.: On cooperation between evolutionary algorithms and other search paradigms. In: 6th International Conference on Evolutionaey Computation, pp. 2317–2324. IEEE Press (1999)
43. Deza, M., Deza, E.: Encyclopedia of Distances. Springer Verlag, Berlin Heidelberg (2009)
44. Droste, S., Jansen, T., Wegener, I.: Perhaps not a free lunch but at least a free appetizer. In: W. Banzhaf, et al. (eds.) Genetic and Evolutionary Computation - GECCO 1999, vol. 1, pp. 833–839. Morgan Kaufmann Publishers, Orlando, FL (1999)
45. Dumitrescu, I., Stützle, T.: Combinations of local search and exact algorithms. In: G.R. Raidl, et al. (eds.) Applications of Evolutionary Computing: EvoWorkshops 2003, *LNCS*, vol. 2611, pp. 212–224. Springer (2003)
46. Fernandes, S., Lourenço, H.: Hybrids combining local search heurisitcs with exact algorithms. In: F. Almeida, et al. (eds.) V Congreso Español sobre Metaheurísticas, Algoritmos Evolutivos y Bioinspirados, pp. 269–274. Las Palmas, Spain (2007)
47. França, P.M., Gupta, J.N., Mendes, A.S., Moscato, P., Veltink, K.J.: Evolutionary algorithms for scheduling a flowshop manufacturing cell with sequence dependent family setups. Computers and Industrial Engineering **48**(3), 491 – 506 (2005)
48. Freisleben, B., Merz, P.: A genetic local search algorithm for solving symmetric and asymmetric traveling salesman problems. In: Proceedings of the 1996 IEEE International Conference on Evolutionary Computation, Nagoya, Japan, pp. 616–621. IEEE Press (1996)
49. French, A., Robinson, A., Wilson, J.: Using a hybrid genetic-algorithm/branch and bound approach to solve feasibility and optimization integer programming problems. Journal of Heuristics **7**(6), 551–564 (2001)
50. Gallardo, J., Cotta, C., Fernández, A.: Solving the multidimensional knapsack problem using an evolutionary algorithm hybridized with branch and bound. In: J. Mira, J. Álvarez (eds.) Artificial Intelligence and Knowledge Engineering Applications: a Bioinspired Approach, *Lecture Notes in Computer Science*, vol. 3562, pp. 21–30. Springer-Verlag, Berlin Heidelberg (2005)
51. Gallardo, J., Cotta, C., Fernández, A.: A multi-level memetic/exact hybrid algorithm for the still life problem. In: T. Runarsson, et al. (eds.) Parallel Problem Solving from Nature IX, *Lecture Notes inComputer Science*, vol. 4193, pp. 212–221. Springer-Verlag, Berlin Heidelberg (2006)
52. Gallardo, J., Cotta, C., Fernández, A.: A memetic algorithm with bucket elimination for the still life problem. In: J. Gottlieb, G. Raidl (eds.) Evolutionary Computation in Combinatorial Optimization, *Lecture Notes in Computer Science*, vol. 3906, pp. 73–84. Springer-Verlag, Budapest (2006)
53. Gallardo, J., Cotta, C., Fernández, A.: Reconstructing phylogenies with memetic algorithms and branch-and-bound. In: S. Bandyopadhyay, U. Maulik, J.T.L. Wang (eds.) Analysis of Biological Data: A Soft Computing Approach, pp. 59–84. World Scientific (2007)
54. Gallardo, J.E., Cotta, C.: A GRASP-based memetic algorithm with path relinking for the far from most string problem. Engineering Applications of Artificial Intelligence **41**, 183–194 (2015)
55. Gallardo, J.E., Cotta, C., Fernández, A.J.: On the hybridization of memetic algorithms with branch-and-bound techniques. IEEE Transactions on Systems, Man and Cybernetics, part B **37**(1), 77–83 (2007)
56. Gallardo, J.E., Cotta, C., Fernández, A.J.: Solving weighted constraint satisfaction problems with memetic/exact hybrid algorithms. Journal of Artificial Intelligence Research **35**, 533–555 (2009)
57. Gen, M., Cheng, R.: Genetic Algorithms and Engineering Optimization. John Wiley & Sons, Inc (2000)
58. Glover, F., Laguna, M., Mart, R.: Fundamentals of scatter search and path relinking. Control and Cybernetics **29**(3), 653–684 (2000)
59. Gorges-Schleuter, M.: ASPARAGOS: An asynchronous parallel genetic optimization strategy. In: J.D. Schaffer (ed.) Proceedings of the 3rd International Conference on Genetic Algorithms, pp. 422–427. Morgan Kaufmann Publishers (1989)

60. Gorges-Schleuter, M.: Explicit Parallelism of Genetic Algorithms through Population Structures. In: H.P. Schwefel, R. Männer (eds.) Parallel Problem Solving from Nature, pp. 150–159. Springer-Verlag (1991)
61. Gottlieb, J.: Permutation-based evolutionary algorithms for multidimensional knapsack problems. In: J. Carroll, E. Damiani, H. Haddad, D. Oppenheim (eds.) ACM Symposium on Applied Computing 2000, pp. 408–414. ACM Press (2000)
62. Grim, P.: The undecidability of the spatialized prisoner's dilemma. Theory and Decision **42**(1), 53–80 (1997)
63. Guimarães, F., Campelo, F., Igarashi, H., Lowther, D., Ramírez, J.: Optimization of cost functions using evolutionary algorithms with local learning and local search. IEEE Transactions on Magnetics **43**(4), 1641–1644 (2007)
64. Guo, X., Wu, Z., Yang, G.: A hybrid adaptive multi-objective memetic algorithm for 0/1 knapsack problem. In: AI 2005: Advances in Artificial Intelligence, *Lecture Notes in Artificial Intelligence*, vol. 3809, pp. 176–185. Springer-Verlag, Berlin Heidelberg (2005)
65. Guo, X., Yang, G., Wu, Z.: A hybrid self-adjusted memetic algorithm for multi-objective optimization. In: 4th Mexican International Conference on Artificial Intelligence, *Lecture Notes in Computer Science*, vol. 3789, pp. 663–672. Springer-Verlag, Berlin Heidelberg (2005)
66. Guo, X., Yang, G., Wu, Z., Huang, Z.: A hybrid fine-timed multi-objective memetic algorithm. IEICE Transactions on Fundamentals of Electronics Communication and Computer Sciences **E89A**(3), 790–797 (2006)
67. Hart, W., Belew, R.: Optimizing an arbitrary function is hard for the genetic algorithm. In: R. Belew, L. Booker (eds.) Proceedings of the Fourth International Conference on Genetic Algorithms, pp. 190–195. Morgan Kaufmann, San Mateo CA (1991)
68. Hart, W., Krasnogor, N., Smith, J.: Recent advances in memetic algorithms, *Studies in Fuzziness and Soft Computing*, vol. 166. Springer-Verlag (2005)
69. Herrera, F., Lozano, M., Sánchez, A.: A taxonomy for the crossover operator for real-coded genetic algorithms: An experimental study. International Journal of Intelligent Systems **18**, 309–338 (2003)
70. Herrera, F., Lozano, M., Verdegay, J.: Tackling real-coded genetic algorithms: Operators and tools for behavioural analysis. Artificial Intelligence Review **12**(4), 265–319 (1998)
71. Hofstadter, D.: Computer tournaments of the prisoners-dilemma suggest how cooperation evolves. Scientific American **248**(5), 16–23 (1983)
72. Horn, P.: Autonomic computing: IBM's perspective on the state of information technology. Tech. rep., IBM Research (2001). URL `http://people.scs.carleton.ca/~soma/biosec/readings/autonomic_computing.pdf`. Accessed 18/09/2017
73. Houck, C., Joines, J., Kay, M., Wilson, J.: Empirical investigation of the benefits of partial lamarckianism. Evolutionary Computation **5**(1), 31–60 (1997)
74. Hu, Z., Bao, Y., Xiong, T.: Comprehensive learning particle swarm optimization based memetic algorithm for model selection in short-term load forecasting using support vector regression. Applied Soft Computing **25**, 15–25 (2014)
75. Huebscher, M., McCann, J.: A survey of autonomic computing -degrees, models and applications. ACM Computing Surveys **40**(3) (2008). Article 7
76. Hulin, M.: An optimal stop criterion for genetic algorithms: A bayesian approach. In: T. Bäck (ed.) Proceedings of the Seventh International Conference on Genetic Algorithms, pp. 135–143. Morgan Kaufmann, San Mateo, CA (1997)
77. Igel, C., Toussaint, M.: On classes of functions for which no free lunch results hold. Information Processing Letters **86**(6), 317–321 (2003)
78. Ishibuchi, H., Hitotsuyanagi, Y., Tsukamoto, N., Nojima, Y.: Use of heuristic local search for single-objective optimization in multiobjective memetic algorithms. In: G. Rudolph, et al. (eds.) Parallel Problem Solving from Nature X, *Lecture Notes in Computer Science*, vol. 5199, pp. 743–752. Springer-Verlag, Berlin Heidelberg (2008)
79. Ishibuchi, H., Murata, T.: Multi-objective genetic local search algorithm. In: T. Fukuda, T. Furuhashi (eds.) 1996 International Conference on Evolutionary Computation, pp. 119–124. IEEE Press, Nagoya, Japan (1996)

80. Ishibuchi, H., Murata, T.: Multi-objective genetic local search algorithm and its application to flowshop scheduling. IEEE Transactions on Systems, Man, and Cybernetics **28**(3), 392–403 (1998)
81. Ishibuchi, H., Yoshida, T., Murata, T.: Balance between genetic search and local search in memetic algorithms for multiobjective permutation flowshop scheduling. IEEE Transactions on Evolutionary Computation **7**(2), 204–223 (2003)
82. Jaszkiewicz, A.: Genetic local search for multiple objective combinatorial optimization. European Journal of Operational Research **137**(1), 50–71 (2002)
83. Jaszkiewicz, A.: A comparative study of multiple-objective metaheuristics on the bi-objective set covering problem and the Pareto memetic algorithm. Annals of Operations Research **131**(1-4), 135–158 (2004)
84. Jaszkiewicz, A., Ishibuchi, H., Zhang, Q.: Multiobjective memetic algorithms. In: F. Neri, C. Cotta, P. Moscato (eds.) Handbook of Memetic Algorithms, *Studies in Computational Intelligence*, vol. 379, pp. 201–217. Springer (2012)
85. Johnson, D., Papadimitriou, C., Yannakakis, M.: How easy is local search? Journal of Computers and System Sciences **37**(1), 79–100 (1988)
86. Jones, T.: Evolutionary algorithms, fitness landscapes and search. Ph.D. thesis, University of New Mexico (1995)
87. Kendall, G., Cowling, P., Sou, E.: Choice function and random hyperheuristics. In: L. Wang, et al. (eds.) Fourth Asia-Pacific Conference on Simulated Evolution and Learning, pp. 667–671 (2002)
88. Kheng, C.W., Chong, S.Y., Lim, M.: Centroid-based memetic algorithm - adaptive lamarckian and baldwinian learning. Int. J. Systems Science **43**(7), 1193–1216 (2012)
89. Klau, G., Ljubić, I., Moser, A., Mutzel, P., Neuner, P., Pferschy, U., Raidl, G., Weiskircher, R.: Combining a memetic algorithm with integer programming to solve the prize-collecting Steiner tree problem. GECCO 04: Genetic and Evolutionary Computation Conference **3102**(Part 1), 1304–1315 (2004)
90. Knowles, J., Corne, D.: Approximating the non-dominated front using the pareto archived evolution strategy. Evolutionary Computation **8**(2), 149–172 (2000)
91. Knowles, J., Corne, D.: A Comparison of Diverse Aproaches to Memetic Multiobjective Combinatorial Optimization. In: A.S. Wu (ed.) Proceedings of the 2000 Genetic and Evolutionary Computation Conference Workshop Program, pp. 103–108 (2000)
92. Knowles, J., Corne, D.: Memetic Algorithms for Multiobjective Optimization: Issues, Methods and Prospects. In: W. Hart, N. Krasnogor, J.E. Smith (eds.) Recent Advances in Memetic Algorithms, *Studies in Fuzziness and Soft Computing*, vol. 166, pp. 313–352. Springer-Verlag (2005)
93. Knowles, J., Corne, D.W.: M-PAES: A Memetic Algorithm for Multiobjective Optimization. In: Proceedings of the 2000 Congress on Evolutionary Computation (CEC00), pp. 325–332. IEEE Press, Piscataway, NJ (2000)
94. Kostikas, K., Fragakis, C.: Genetic programming applied to mixed integer programming. In: M. Keijzer, et al. (eds.) 7th European Conference on Genetic Programming, *Lecture Notes in Computer Science*, vol. 3003, pp. 113–124. Springer-Verlag, Berlin Heidelberg (2004)
95. Krasnogor, N.: Studies in the theory and design space of memetic algorithms. Ph.D. thesis, University of the West of England (2002)
96. Krasnogor, N.: Self generating metaheuristics in bioinformatics: The proteins structure comparison case. Genetic Programming and Evolvable Machines **5**(2), 181–201 (2004)
97. Krasnogor, N., Smith, J.: A tutorial for competent memetic algorithms: Model, taxonomy and desing issues. IEEE Transactions on Evolutionary Computation **9**(5), 474–488 (2005)
98. Krasnogor, N., Smith, J.: Memetic algorithms: The polynomial local search complexity theory perspective. Journal of Mathematical Modelling and Algorithms **7**(1), 3–24 (2008)
99. Larrañaga, P., Lozano, J. (eds.): Estimation of Distribution Algorithms, *Genetic Algorithms and Evolutionary Computation*, vol. 2. Springer Verlag, Berlin Heidelberg (2002)
100. Li, B.B., Wang, L., Liu, B.: An effective PSO-based hybrid algorithm for multiobjective permutation flow shop scheduling. IEEE Transactions on Systems, Man and Cybernetics Part B **38**(4), 818–831 (2008)

101. Lim, D., Ong, Y.S., Jin, Y., Sendhoff, B.: A study on metamodeling techniques, ensembles, and multi-surrogates in evolutionary computation. In: D. Thierens, et al. (eds.) GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation, vol. 2, pp. 1288–1295. ACM Press, London (2007)
102. Lim, K., Ong, Y.S., Lim, M., Chen, X., Agarwal, A.: Hybrid ant colony algorithms for path planning in sparse graphs. soft Computing **12**(10), 981–994 (2008)
103. Lin, S., Kernighan, B.: An Effective Heuristic Algorithm for the Traveling Salesman Problem. Operations Research **21**(2), 498–516 (1973)
104. Liu, B., Wang, L., Jin, Y.: An effective PSO-based memetic algorithm for flow shop scheduling. IEEE Transactions on Systems, Man, and Cybernetics, Part B **37**(1), 18–27 (2007)
105. Liu, B., Wang, L., Jin, Y., Huang, D.: Designing neural networks using PSO-based memetic algorithm. In: D. Liu, S. Fei, Z.G. Hou, H. Zhang, C. Sun (eds.) 4th International Symposium on Neural Networks, *Lecture Notes in Computer Science*, vol. 4493, pp. 219–224. Springer-Verlag (2007)
106. Liu, B., Wang, L., Jin, Y.H., Huang, D.X.: An effective PSO-based memetic algorithm for TSP. In: Intelligent Computing in Signal Processing and Pattern Recognition, *Lecture Notes in Control and Information Sciences*, vol. 345, pp. 1151–1156. Springer-Verlag (2006)
107. Liu, D., Tan, K.C., Goh, C.K., Ho, W.K.: A multiobjective memetic algorithm based on particle swarm optimization. IEEE Transactions on Systems, Man, and Cybernetics, Part B **37**(1), 42–50 (2007)
108. Lozano, M., Herrera, F., Krasnogor, N., Molina, D.: Real-coded memetic algorithms with crossover hill-climbing. Evolutionary Computation **12**(3), 273–302 (2004)
109. Mendes, A., Cotta, C., Garcia, V., França, P., Moscato, P.: Gene ordering in microarray data using parallel memetic algorithms. In: T. Skie, C.S. Yang (eds.) Proceedings of the 2005 International Conference on Parallel Processing Workshops, pp. 604–611. IEEE Press, Oslo, Norway (2005)
110. Merz, P.: Memetic algorithms and fitness landscapes in combinatorial optimization. In: F. Neri, C. Cotta, P. Moscato (eds.) Handbook of Memetic Algorithms, *Studies in Computational Intelligence*, vol. 379, pp. 95–119. Springer, Berlin Heidelberg (2012)
111. Michalewicz, Z.: Repair algorithms. In: T. Bäck, et al. (eds.) Handbook of Evolutionary Computation, pp. C5.4:1–5. Institute of Physics Publishing and Oxford University Press, Bristol, New York (1997)
112. Molina, D., Herrera, F., Lozano, M.: Adaptive local search parameters for real-coded memetic algorithms. In: D. Corne, et al. (eds.) Proceedings of the 2005 IEEE Congress on Evolutionary Computation, vol. 1, pp. 888–895. IEEE Press, Edinburgh, Scotland, UK (2005)
113. Molina, D., Lozano, M., Herrera, F.: Memetic algorithms for intense continuous local search methods. In: M. Blesa, et al. (eds.) Hybrid Metaheuristics 2008, *Lecture Notes in Computer Science*, vol. 5296, pp. 58–71. Springer-Verlag, Berlin Heidelberg (2008)
114. Molina, D., Lozano, M., Sánchez, A.M., Herrera, F.: Memetic algorithms based on local search chains for large scale continuous optimisation problems: Ma-ssw-chains. Soft Computing **15**(11), 2201–2220 (2011)
115. Moscato, P.: On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts: Towards Memetic Algorithms. Tech. Rep. Caltech Concurrent Computation Program, Report. 826, California Institute of Technology, Pasadena, California, USA (1989)
116. Moscato, P.: On evolution, search, optimization, genetic algorithms and martial arts. towards memetic algorithms. Tech. Rep. 826, California Institute of Technology, Pasadena, California, USA (1989)
117. Moscato, P.: An Introduction to Population Approaches for Optimization and Hierarchical Objective Functions: The Role of Tabu Search. Annals of Operations Research **41**(1-4), 85–121 (1993)
118. Moscato, P.: Memetic algorithms: A short introduction. In: D. Corne, M. Dorigo, F. Glover (eds.) New Ideas in Optimization, pp. 219–234. McGraw-Hill, Maidenhead, Berkshire, England, UK (1999)

119. Moscato, P.: Memetic algorithms: The untold story. In: F. Neri, C. Cotta, P. Moscato (eds.) Handbook of Memetic Algorithms, *Studies in Computational Intelligence*, vol. 379, pp. 275–309. Springer, Berlin Heidelberg (2012)
120. Moscato, P., Cotta, C.: A gentle introduction to memetic algorithms. In: F. Glover, G. Kochenberger (eds.) Handbook of Metaheuristics, pp. 105–144. Kluwer Academic Publishers, Boston MA (2003)
121. Moscato, P., Cotta, C.: Memetic algorithms. In: T. González (ed.) Handbook of Approximation Algorithms and Metaheuristics, chap. 22. Taylor & Francis (2006)
122. Moscato, P., Cotta, C.: A modern introduction to memetic algorithms. In: M. Gendreau, J. Potvin (eds.) Handbook of Metaheuristics, *International Series in Operations Research and Management Science*, vol. 146, 2nd edition, pp. 141–183. Springer-Verlag, Berlin Heidelberg (2010)
123. Moscato, P., Cotta, C., Mendes, A.: Memetic algorithms. In: G. Onwubolu, B. Babu (eds.) New Optimization Techniques in Engineering, pp. 53–85. Springer-Verlag, Berlin Heidelberg (2004)
124. Moscato, P., Mendes, A., Berretta, R.: Benchmarking a memetic algorithm for ordering microarray data. Biosystems **88**(1), 56 – 75 (2007)
125. Moscato, P., Mendes, A., Cotta, C.: Scheduling & produ. In: G. Onwubolu, B. Babu (eds.) New Optimization Techniques in Engineering, pp. 655–680. Springer-Verlag, Berlin Heidelberg (2004)
126. Mühlenbein, H.: Evolution in Time and Space – The Parallel Genetic Algorithm. In: G.J. Rawlins (ed.) Foundations of Genetic Algorithms, pp. 316–337. Morgan Kaufmann Publishers (1991)
127. Mühlenbein, H., Gorges-Schleuter, M., Krämer, O.: Evolution Algorithms in Combinatorial Optimization. Parallel Computing **7**(1), 65–88 (1988)
128. Nagata, Y., Kobayashi, S.: Edge assembly crossover: A high-power genetic algorithm for the traveling salesman problem. In: T. Bäck (ed.) Proceedings of the Seventh International Conference on Genetic Algorithms, pp. 450–457. Morgan Kaufmann, San Mateo, CA (1997)
129. Nakamaru, M., Matsuda, H., Iwasa, Y.: The evolution of social interaction in lattice models. Sociological Theory and Methods **12**(2), 149–162 (1998)
130. Nakamaru, M., Nogami, H., Iwasa, Y.: Score-dependent fertility model for the evolution of cooperation in a lattice. Journal of Theoretical Biology **194**(1), 101–124 (1998)
131. Nelder, J.A., Mead, R.: A simplex method for function minimization. The Computer Journal **7**(4), 308–313 (1965)
132. Neri, F., Cotta, C.: Memetic algorithms and memetic computing optimization: A literature review. Swarm and Evolutionary Computation **2**, 1–14 (2012)
133. Neri, F., Cotta, C., Moscato, P. (eds.): Handbook of Memetic Algorithms, *Studies in Computational Intelligence*, vol. 379. Springer-Verlag, Berlin Heidelberg (2012)
134. Neri, F., Tirronen, V.: On memetic differential evolution frameworks: A study of advantages and limitations in hybridization. In: J. Wang (ed.) 2008 IEEE World Congress on Computational Intelligence, pp. 2135–2142. IEEE Computational Intelligence Society, IEEE Press, Hong Kong (2008)
135. Neri, F., Tirronen, V., Kärkkäinen, T., Rossi, T.: Fitness diversity based adaptation in multimeme algorithms: A comparative study. In: IEEE Congress on Evolutionary Computation - CEC 2007, pp. 2374–2381. IEEE Press, Singapore (2007)
136. Nguyen, Q.C., Ong, Y.S., Kuo, J.L.: A hierarchical approach to study the thermal behavior of protonated water clusters H+(H2O)(n). Journal of Chemical Theory and Computation **5**(10), 2629–2639 (2009)
137. Nguyen, Q.H., Ong, Y.S., Krasnogor, N.: A study on the design issues of memetic algorithm. In: D. Srinivasan, L. Wang (eds.) 2007 IEEE Congress on Evolutionary Computation, pp. 2390–2397. IEEE Computational Intelligence Society, IEEE Press, Singapore (2007)
138. Nogueras, R., Cotta, C.: An analysis of migration strategies in island-based multimemetic algorithms. In: T. Bartz-Beielstein, et al. (eds.) Parallel Problem Solving from Nature - PPSN XIII, *Lecture Notes in Computer Science*, vol. 8672, pp. 731–740. Springer Verlag, Berlin Heidelberg (2014)

139. Nogueras, R., Cotta, C.: A study on multimemetic estimation of distribution algorithms. In: T. Bartz-Beielstein, et al. (eds.) Parallel Problem Solving from Nature - PPSN XIII, *Lecture Notes in Computer Science*, vol. 8672, pp. 322–331. Springer Verlag, Berlin Heidelberg (2014)
140. Nogueras, R., Cotta, C.: A study on meme propagation in multimemetic algorithms. Applied Mathematics and Computer Science **25**(3), 499–512 (2015)
141. Nogueras, R., Cotta, C.: Studying self-balancing strategies in island-based multimemetic algorithms. Journal of Computational and Applied Mathematics **293**, 180–191 (2016)
142. Nogueras, R., Cotta, C.: Self-healing strategies for memetic algorithms in unstable and ephemeral computational environments. Natural Computing **6**(2), 189–200 (2017)
143. Noman, N., Iba, H.: Accelerating differential evolution using an adaptive local search. IEEE Transactions on Evolutionary Computation **12**(1), 107–125 (2008)
144. Norman, M., Moscato, P.: A competitive and cooperative approach to complex combinatorial search. In: Proceedings of the 20th Informatics and Operations Research Meeting, pp. 3.15–3.29. Buenos Aires (1989)
145. de Oca, M.A.M., Cotta, C., Neri, F.: Local search. In: F. Neri, C. Cotta, P. Moscato (eds.) Handbook of Memetic Algorithms, *Studies in Computational Intelligence*, vol. 379, pp. 29–41. Springer (2012)
146. Ong, Y., Keane, A.: Meta-lamarckian learning in memetic algorithm. IEEE Transactions on Evolutionary Computation **8**(2), 99–110 (2004)
147. Ong, Y., Lim, M., Chen, X.: Memetic computation –past, present and future. IEEE Computational Intelligence Magazine **5**(2), 24–31 (2010)
148. Özcan, E., Drake, J.H., Altintas, C., Asta, S.: A self-adaptive multimeme memetic algorithm co-evolving utility scores to control genetic operators and their parameter settings. Applied Soft Computing **49**, 81–93 (2016)
149. Pan, Q.K., Wang, L., Qian, B.: A novel multi-objective particle swarm optimization algorithm for no-wait flow shop scheduling problems. Journal of Engineering Manufacture **222**(4), 519–539 (2008)
150. Paszkowicz, W.: Properties of a genetic algorithm extended by a random self-learning operator and asymmetric mutations: A convergence study for a task of powder-pattern indexing. Analytica Chimica Acta **566**(1), 81–98 (2006)
151. Peinado, M., Lengauer, T.: Parallel "go with the winners algorithms" in the LogP Model. In: Proceedings of the 11th International Parallel Processing Symposium, pp. 656–664. IEEE Computer Society Press, Los Alamitos, California (1997)
152. Pelikan, M., Hauschild, M., Lobo, F.: Estimation of distribution algorithms. In: J. Kacprzyk, W. Pedrycz (eds.) Handbook of Computational Intelligence, pp. 899–928. Springer Verlag, Berlin Heidelberg (2015)
153. Petalas, Y.G., Parsopoulos, K.E., Vrahatis, M.N.: Memetic particle swarm optimization. Annals of Operations Research **156**(1), 99–127 (2007)
154. Prins, C., Prodhon, C., Calvo, R.: A memetic algorithm with population management (MA | PM) for the capacitated location-routing problem. In: J. Gottlieb, G. Raidl (eds.) Evolutionary Computation in Combinatorial Optimization, *Lecture Notes in Computer Science*, vol. 3906, pp. 183–194. Springer-Verlag, Budapest (2006)
155. Prodhom, C., Prins, C.: A memetic algorithm with population management (MA|PM) for the periodic location-routing problem. In: M. Blesa, et al. (eds.) Hybrid Metaheuristics 2008, *Lecture Notes in Computer Science*, vol. 5296, pp. 43–57. Springer-Verlag, Berlin Heidelberg (2008)
156. Puchinger, J., Raidl, G.: Combining metaheuristics and exact algorithms in combinatorial optimization: A survey and classification. In: J. Mira, J. Álvarez (eds.) Artificial Intelligence and Knowledge Engineering Applications: a Bioinspired Approach, *Lecture Notes in Computer Science*, vol. 3562, pp. 41–53. Springer-Verlag (2005)
157. Puchinger, J., Raidl, G., Koller, G.: Solving a real-world glass cutting problem. In: J. Gottlieb, G. Raidl (eds.) 4th European Conference on Evolutionary Computation in Combinatorial Optimization, *Lecture Notes in Computer Science*, vol. 3004, pp. 165–176. Springer-Verlag, Berlin Heidelberg (2004)

158. Radcliffe, N.: The algebra of genetic algorithms. Annals of Mathematics and Artificial Intelligence **10**(4), 339–384 (1994)
159. Radcliffe, N., Surry, P.: Fitness Variance of Formae and Performance Prediction. In: L. Whitley, M. Vose (eds.) Proceedings of the 3rd Workshop on Foundations of Genetic Algorithms, pp. 51–72. Morgan Kaufmann, San Francisco (1994)
160. Radcliffe, N., Surry, P.: Formal Memetic Algorithms. In: T. Fogarty (ed.) Evolutionary Computing: AISB Workshop, *Lecture Notes in Computer Science*, vol. 865, pp. 1–16. Springer-Verlag, Berlin (1994)
161. Rechenberg, I.: Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution. Frommann-Holzboog Verlag, Stuttgart (1973)
162. Resende, M., Ribeiro, C.: Greedy randomized adaptive search procedures. In: F. Glover, G. Kochenberger (eds.) Handbook of Metaheuristics, pp. 219–249. Kluwer Academic Publishers, Boston MA (2003)
163. Resende, M.G.C., Ribeiro, C.C.: Optimization by GRASP: Greedy randomized adaptive search procedures. Springer, New York (2016)
164. Sabar, N.R., Abawajy, J.H., Yearwood, J.: Heterogeneous cooperative co-evolution memetic differential evolution algorithm for big data optimization problems. IEEE Trans. Evolutionary Computation **21**(2), 315–327 (2017)
165. Schuetze, O., Sanchez, G., Coello Coello, C.: A new memetic strategy for the numerical treatment of multi-objective optimization problems. In: M. Keijzer, et al. (eds.) GECCO '08: Proceedings of the 10th annual conference on Genetic and evolutionary computation, pp. 705–712. ACM Press, Atlanta, GA, USA (2008)
166. Schumacher, C., Vose, M., Whitley, L.: The no free lunch and description length. In: L. Spector, et al. (eds.) Genetic and Evolutionary Computation - GECCO 2001, pp. 565–570. Morgan Kaufmann Publishers, San Francisco, CA, USA (2001)
167. Schwefel, H.: Imitating evolution: Collective, two-level learning processes. In: Explaining Process and Change - Approaches to Evolutionary Economics, pp. 49–63. University of Michigan Press, Ann Arbor, Michigan (1992)
168. Schwefel, H.P.: Evolution strategies: A family of non-linear optimization techniques based on imitating some principles of natural evolution. Annals of Operations Research **1**(2), 165–167 (1984)
169. Smith, J.: The Co-Evolution of memetic algorithms for protein structure prediction. In: W. Hart, N. Krasnogor, J. Smith (eds.) Recent Advances in Memetic Algorithms, *Studies in Fuzziness and Soft Computing*, vol. 166, pp. 105–128. Springer Verlag, Berlin Heidelberg (2005)
170. Smith, J.: Self-adaptation in evolutionary algorithms for combinatorial optimization. In: C. Cotta, M. Sevaux, K. Sörensen (eds.) Adaptive and Multilevel Metaheuristics, *Studies in Computational Intelligence*, vol. 136, pp. 31–57. Springer-Verlag, Berlin Heidelberg (2008)
171. Smith, J.: Self-adaptative and coevolving memetic algorithms. In: F. Neri, C. Cotta, P. Moscato (eds.) Handbook of Memetic Algorithms, *Studies in Computational Intelligence*, vol. 379, pp. 167–188. Springer-Verlag, Berlin Heidelberg (2012)
172. Smith, J.E.: Coevolving memetic algorithms: A review and progress report. IEEE Transactions on Systems, Man and Cybernetics, Part B **37**(1), 6–17 (2007)
173. Soak, S.M., Lee, S.W., Mahalik, N., Ahn, B.H.: A new memetic algorithm using particle swarm optimization and genetic algorithm. In: Knowledge-based Intelligent Information and Engineering Systems, *Lecture Notes in Artificial Intelligence*, vol. 4251, pp. 122–129. Springer-Verlag (2006)
174. Sörensen, K., Sevaux, M.: MA | PM: memetic algorithms with population management. Computers & OR **33**(5), 1214–1225 (2006)
175. Storn, R., Price, K.: Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. Journal of Global Optimization **11**(4), 341–359 (1997)
176. Sudholt, D.: Memetic algorithms with variable-depth search to overcome local optima. In: M. Keijzer, et al. (eds.) GECCO '08: Proceedings of the 10th annual conference on Genetic and evolutionary computation, pp. 787–794. ACM Press, Atlanta, GA, USA (2008)

177. Sudholt, D.: The impact of parametrization in memetic evolutionary algorithms. Theoretical Computer Science **410**(26), 2511–2528 (2009)
178. Sudholt, D.: Parametrization and balancing local and global search. In: F. Neri, C. Cotta, P. Moscato (eds.) Handbook of Memetic Algorithms, *Studies in Computational Intelligence*, vol. 379, pp. 55–72. Springer, Berlin Heidelberg (2012)
179. Sun, J., Garibaldi, J.M., Krasnogor, N., Zhang, Q.: An intelligent multi-restart memetic algorithm for box constrained global optimisation. Evolutionary Computation **21**(1), 107–147 (2013)
180. Surry, P., Radcliffe, N.: Inoculation to initialise evolutionary search. In: T. Fogarty (ed.) Evolutionary Computing: AISB Workshop, no. 1143 in Lecture Notes in Computer Science, pp. 269–285. Springer-Verlag (1996)
181. Syswerda, G.: Uniform crossover in genetic algorithms. In: J. Schaffer (ed.) Proceedings of the 3rd International Conference on Genetic Algorithms, pp. 2–9. Morgan Kaufmann, San Mateo, CA (1989)
182. Tirronen, V., Neri, F., Kärkkäinen, T., Majava, K., Rossi, T.: A memetic differential evolution in filter design for defect detection in paper production. In: M. Giacobini, et al. (eds.) Applications of Evolutionary Computing, *Lecture Notes in Computer Science*, vol. 4448, pp. 320–329. Springer-Verlag (2007)
183. Ulungu, E., Teghem, J., Fortemps, P., Tuyttens, D.: MOSA method: A tool for solving multiobjective combinatorial optimization problems. Journal of Multi-Criteria Decision Analysis **8**(4), 221–236 (1999)
184. Voĭtsekhovskiĭ, M.: Continuous set. In: M. Hazewinkel (ed.) Encyclopaedia of Mathematics, vol. 1. Springer (1995)
185. Wang, S., Wang, L.: An estimation of distribution algorithm-based memetic algorithm for the distributed assembly permutation flow-shop scheduling problem. IEEE Trans. Systems, Man, and Cybernetics: Systems **46**(1), 139–149 (2016)
186. Wanner, E., Guimarães, F., Takahashi, R., Fleming, P.: Local search with quadratic approximations into memetic algorithms for optimization with multiple criteria. Evolutionary Computation **16**(2), 185–224 (2008)
187. Wanner, E., Guimarães, F., Takahashi, R., Lowther, D., Ramírez, J.: Multiobjective memetic algorithms with quadratic approximation-based local search for expensive optimization in electromagnetics. IEEE Transactions on Magnetics **44**(6), 1126–1129 (2008)
188. Whitley, D.: Using reproductive evaluation to improve genetic search and heuristic discovery. In: J. Grefenstette (ed.) Proceedings of the 2nd International Conference on Genetic Algorithms and their Applications, pp. 108–115. Lawrence Erlbaum Associates, Cambridge, MA (1987)
189. Whitley, D., Gordon, V.S., Mathias, K.: Lamarckian evolution, the baldwin effect and function optimization. In: Y. Davidor, H.P. Schwefel, R. Männer (eds.) Parallel Problem Solving from Nature — PPSN III, pp. 5–15. Springer, Berlin, Heidelberg (1994)
190. Wiston, P.: Artificial Intelligence. Addison-Wesley, Reading MA (1984)
191. Wolpert, D., Macready, W.: No free lunch theorems for optimization. IEEE Transactions on Evolutionary Computation **1**(1), 67–82 (1997)
192. Wright, A.H.: Genetic algorithms for real parameter qptimization. In: G.J.E. Rawlins (ed.) Proceedings of the First Workshop on Foundations of Genetic Algorithms, pp. 205–218. Morgan Kaufmann (1990)
193. Yannakakis, M.: Computational complexity. In: E. Aarts, J. Lenstra (eds.) Local Search in Combinatorial Optimization, pp. 19–55. Wiley, Chichester (1997)
194. Yuan, Q., Qian, F., Du, W.: A hybrid genetic algorithm with the baldwin effect. Information Sciences **180**(5), 640–652 (2010)
195. Zhen, Z., Wang, Z., Gu, Z., Liu, Y.: A novel memetic algorithm for global optimization based on PSO and SFLA. In: L. Kang, Y. Liu, S.Y. Zeng (eds.) 2nd International Symposium on Advances in Computation and Intelligence, *Lecture Notes in Computer Science*, vol. 4683, pp. 127–136. Springer-Verlag (2007)
196. Zhou, Z., Ong, Y.S., Lim, M.H., Lee, B.S.: Memetic algorithm using multi-surrogates for computationally expensive optimization problems. Soft Computing **11**(10), 957–971 (2007)

197. Zitzler, E., Laumanns, M., Bleuler, S.: A Tutorial on Evolutionary Multiobjective Optimization. In: X. Gandibleux, et al. (eds.) Metaheuristics for Multiobjective Optimisation, *Lecture Notes in Economics and Mathematical Systems*, vol. 535. Springer-Verlag (2004)