Intelligent Computing
A SCIENCE PARTNER JOURNAL

REVIEW ARTICLE

# Designing New Metaheuristics: Manual Versus Automatic Approaches

**Christian L. Camacho-Villalón*, Thomas Stützle, and Marco Dorigo**

Institut de Recherches Interdisciplinaires et de Développements en Intelligence Artificielle (IRIDIA), Université Libre de Bruxelles, 1050 Bruxelles, Belgium.

*Address correspondence to: ccamacho@ulb.ac.be

**Citation:** Camacho-Villalón CL, Stützle T, Dorigo M. Designing New Metaheuristics: Manual Versus Automatic Approaches. *Intell. Comput.* 2023;2:Article 0048. https://doi.org/10.34133/icomputing.0048

Submitted 22 May 2023
Accepted 5 July 2023
Published 4 December 2023



A metaheuristic is a collection of algorithmic concepts that can be used to define heuristic methods applicable to a wide set of optimization problems for which exact/analytical approaches are either limited or impractical. In other words, a metaheuristic can be considered a general algorithmic framework that can be easily adapted to different optimization problems. In this article, we discuss the two main approaches used to create new metaheuristics: manual design, which is based on the designer's "intuition" and often involves looking for inspiration in other fields of knowledge, and automatic design, which seeks to reduce human involvement in the design process by harnessing recent advances in automatic algorithm configuration methods. In this context, we discuss the trend of manually designed "novel" metaphor-based metaheuristics inspired by natural, artificial, and even supernatural behaviors. In recent years, this trend has been strongly criticized due to the uselessness of new metaphors in devising truly novel algorithms and the confusion such metaheuristics have created in the literature. We then present automatic design as a powerful alternative to manual design that has the potential to render the "novel" metaphor-based metaheuristics trend obsolete. Finally, we examine several fundamental aspects of the field of metaheuristics and offer suggestions for improving them.

## Introduction

Optimization is a vast research field with hundreds of years of history. It deals with a wide variety of optimization problems and solution methods. Although the early days of optimization were characterized by the development of algorithms that could find optimal solutions, it eventually became clear that many optimization problems cannot be efficiently solved to optimality. Well-known examples of such problems are multimodal non-differentiable functions in the continuous optimization domain [1,2] and NP-hard problems in the discrete optimization domain [3–5]. With the advent of increasingly powerful computers, heuristic algorithms have rapidly become the mainstream approach to difficult optimization problems, replacing the use of exact algorithms in many cases. In other words, the focus of research has shifted from the design and development of algorithms that find the best solution to the design and development of algorithms that can rapidly provide solutions that are good, although not provably optimal. Since the seminal work of Glover [6], the most commonly used term to refer to this type of algorithm has been metaheuristic.

There have been many attempts to provide a definition of the term "metaheuristic" that is both precise and encompasses all the diverse metaheuristics that have been proposed in the literature. The definition provided by the Metaheuristics Network [7], which is the one we adopt in this article, is as follows:

"A metaheuristic is a set of algorithmic concepts that can be used to define heuristic methods applicable to a wide set of different problems. In other words, a metaheuristic can be seen as a general algorithmic framework which can be applied to different optimization problems with relatively few modifications to make it adapted to a specific problem."

Some of the most popular and best-performing metaheuristics include evolutionary computation [8–12], tabu search [13,14], simulated annealing [15,16], ant colony optimization [17–19], particle swarm optimization [20–22], and iterated local search [23,24].

Seeking to improve the efficiency of metaheuristic implementations, researchers have redefined the components of various metaheuristics and explored new ways of implementing them. However, the resulting high number of possible components to use increases considerably the complexity of the design process. Thus, it became less efficient to implement metaheuristics manually, that is, by handcrafting the components one by one.

To address this problem, and motivated by the early success of metaheuristics inspired by natural processes (e.g., evolutionary computation, simulated annealing, ant colony optimization, and particle swarm optimization), some members of the metaheuristics community have been proposing "novel" metaheuristics based on a disparate set of metaphors. However, not only this approach is just another instance of the inefficient manual design approach, but it has also caused a number of undesirable consequences for the entire field. The main negative consequence is that, most of the time, the only novelty in a proposed "novel" metaheuristic is the use of new and confusing terminology. Analyses of these "novel" metaheuristics show that they can often be exactly mapped to already published ones by changing the terminology used to describe them [25–34].

Another approach has been to propose automatic algorithm design methods in which human involvement is less important [35] and that have no need of novel metaphors. The development of these automatic methods and their application in creating efficient metaheuristic implementations are currently central topics in the field of metaheuristics. In this paper, we examine the automatic design approach in detail and describe how it is being used to create a new generation of high-performance metaheuristic implementations. Following this examination of the automatic design approach, we provide some reflections on fundamental aspects of the field of metaheuristics that we believe will help advance the field.

The remainder of this paper is organized as follows. The "Metaheuristics" section provides a general introduction to metaheuristics by explaining their main characteristics and classifications. The "Manual Design of Metaheuristics" section elaborates on the disadvantages of the manual algorithm design approach and discusses the trend of "novel" metaphor-based algorithms and its negative impact on the field. The "Automatic Design of Metaheuristics" section describes the automatic algorithm design paradigm and its use in developing high-performance metaheuristic implementations from automatically configurable frameworks. The "Discussion" section presents our perspective on improving three foundational aspects of the field: the focus of the research, the way metaheuristic implementations are benchmarked, and the creation of new metaheuristics. Finally, the "Conclusions" section provides a summary of the topics discussed in the paper.

## Metaheuristics

Broadly speaking, metaheuristics are optimization techniques that extract information from the search space and use it to direct the search toward areas where high-quality solutions can be found. Most metaheuristics have the following characteristics: They are iterative—that is, solutions are constructed/perturbed based on starting points or complete initial solutions by an optimization process that consists of a number of steps that repeat for multiple iterations; they use randomization—that is, they make use of random variables in one or more of their components; and they have a user-defined termination criterion—e.g., reaching a maximum computation time or obtaining a solution of minimum desired quality.

Metaheuristics can be classified in different ways. For example, they can be constructive or perturbative, depending on the manner in which they create new candidate solutions. They can be memory-based or without memory, depending on whether they memorize solutions (or solution components). They can also be metaphor-based or non-metaphor-based, depending on whether they are inspired by a metaphor. However, one of the most common ways to distinguish them is by the number of solutions handled in each iteration, that is, by distinguishing between those that are single-solution and those that are population-based.

In single-solution metaheuristics, the optimization process is based on a single solution that is iteratively improved by means of small changes. Examples of this type of metaheuristic include tabu search [13], simulated annealing [16,36], and iterated local search [23]. By contrast, population-based metaheuristics maintain multiple solutions in parallel and combine them to create new solutions. Most swarm intelligence [22,37–39] and evolutionary algorithms [8–12,40] belong to this class.

Population-based and single-solution metaheuristics have different, often complementary, optimization capabilities. For example, while population-based metaheuristics are typically better at exploring the search space and quickly identifying some of the most promising regions, single-solution metaheuristics tend to be more effective at improving existing solutions. Thus, it is common to combine these two types of metaheuristics to produce metaheuristic implementations that are better equipped to perform robust optimization. Two methods of doing so are component-based hybrids, in which a population-based metaheuristic includes a single-solution metaheuristic as an additional component in its procedure, and algorithm-sequential hybrids, in which two metaheuristics are executed one after the other, with the output of one metaheuristic becoming the input of the next in the sequence [41,42].

Although most metaheuristics share the same high-level characteristics (i.e., they are iterative, are randomized, and have a user-defined termination criterion, as mentioned above), they can differ in numerous aspects, including the specific mechanisms used to sample the search space, the way they organize the search process, and the mechanisms they use to control the exploration–exploitation trade-off. In several cases, these aspects are implemented in the metaheuristics by taking inspiration from natural, social, or human-made processes. The metaheuristics that have been created using the "inspiration-based" approach are commonly referred to as metaphor-based metaheuristics [43].

Indeed, the approach of looking for inspiration from other fields of knowledge was important for designing some of the best-performing metaheuristics. As early as the 1970s, the use of naturally occurring optimization processes—such as evolution by natural selection, which inspired evolutionary computation [8,9]—to formulate new optimization algorithms became appealing to researchers in the areas of computer science and engineering. However, it was in the 1980s and early 1990s that the approach of looking for inspiration in other fields of knowledge began to be vigorously explored in the field of optimization and became a major driver of its development. Notably, during these decades, thermodynamic principles were used to develop simulated annealing [16,36], the foraging behavior of some ant species was used to develop ant colony optimization [44–46], and the dynamics and social interactions of bird flocks were used to develop particle swarm optimization [20–22].

Owing to their success, these metaphor-based metaheuristics are among the most extensively studied and well-understood techniques in the optimization literature. The reason for their success is not their source of inspiration; rather, it is that they introduced novel and useful algorithmic concepts that could be conveniently used for optimization. However, some members of the metaheuristics research community nevertheless continue to propose "novel" metaphor-based metaheuristics whose only novelty is in the chosen metaphor and terminology.

## Manual Design of Metaheuristics

### General issues with metaheuristic manual design

Historically, new metaheuristics, and metaheuristic implementations with improved performance, were created by manual design—that is, algorithm designers manually devised or modified designs according to their knowledge (empirical, theoretical, or intuitive) and expertise [35,47].

Although manual design has been useful for implementing high-performance metaheuristics, it is becoming less efficient. Indeed, the design of a high-performance implementation of a metaheuristic requires both choosing among large sets of possible metaheuristic components and fine-tuning the values of their parameters, and it is unrealistic to expect developers who rely on their own knowledge and expertise to perform these tasks efficiently. Human algorithm designers are biased by their previous experience and limited in the number of designs they can attempt; thus, the design process is time-consuming and error-prone. In practice, the main disadvantage of manual design is that it limits the flexibility with which new metaheuristic designs can be explored and the generality of the developed metaheuristics. In addition, the manual design process is directed by subjective choices, which may make it difficult for the design process to be understood a posteriori.

This flexibility limitation is caused by the fact that metaheuristics have been traditionally conceived as monolithic blocks; that is, they have a predefined rigid structure. This rigid structure constrains the options for modifying the behavior of the metaheuristic: The designer can either adjust the parameter values of the metaheuristic or redefine its components [48]. However, the existence of high-performance hybrid metaheuristic implementations [41,42,49,50] highlights the importance of metaheuristic designs that go beyond the bounds of the original monolithic block. However, creating hybrid designs manually is difficult.

The generality limitation is caused by the fact that, in most cases, the initial development of a metaheuristic is performed with some application in mind; the performance of the developed metaheuristic therefore tends to be very good for the specific problem or problem class for which the metaheuristic was developed, but less good for other problems or problem classes. In general, the performance of a metaheuristic tends to drop considerably when it is applied to a problem that has different characteristics from those for which it was originally developed or when the optimization scenario has specific constraints that were not considered in the initial metaheuristic design, such as being allowed to run only for a limited time or being unable to use problem-related information (e.g., black-box scenarios or ill-defined real-life problems) [51,52]. In such cases, a new implementation of the metaheuristic must be designed for the new problem in order to obtain good results.

However, the process of manually adjusting a metaheuristic so as to propose a new metaheuristic implementation is laborious and has several potential pitfalls. First, as mentioned above, the designer of the new metaheuristic implementation may start by trying to enhance the behavior of the metaheuristic by using different parameter values or by modifying one or more of the metaheuristic components in the implementation that, according to his/her knowledge and expertise, may hinder the performance of the metaheuristic. If this process, which depends solely on the ability of the human designer, does not produce the desired outcome, the only option may be to design and implement a new metaheuristic from scratch. The latter task, however, is even more challenging than the former because the algorithm designer must now create a better design, once again guided only by his/her knowledge and previous experience.

Finally, manual design makes the design of a metaheuristic a subjective process in which the rationale behind some design decisions remains hidden in the mind of the human designer, and what is learned by the designer is therefore not shared with the rest of the research community. In practice, manual design involves finding a good algorithm design in a large set of options through trial and error. To reduce the number of options, algorithm designers eliminate designs that, based on their knowledge, they believe would not work for the problem at hand. Designs that are experimentally tested but do not produce good results are often discarded rather than reported in technical papers. Knowing more about which designs have been deemed unsuitable for testing and which designs have already proven unsuccessful would help guide the creation of new designs.

## The "novel" metaphor-based metaheuristic problem

For better or for worse, manual design has been guided by optimization processes observed in natural systems. Although taking inspiration from nature was successful in the early days of metaphor-based metaheuristics, when innovative and well-performing metaheuristics such as evolutionary computation, ant colony optimization, simulated annealing, and particle swarm optimization were proposed, this is no longer the case. Instead, it has become commonplace to find papers proposing "novel" metaheuristics, in which the use of the metaphors does not provide a clear mapping between a natural behavior and the implemented optimization process.

In the last few decades, hundreds of metaphors from the most diverse set of natural, artificial, and even supernatural behaviors have been used to develop "novel" metaphor-based metaheuristics. In most cases, these metaheuristics are based on simplistic mathematical models that vaguely match the behaviors that inspired them and are presented using elaborate metaphoric descriptions that make them difficult to understand (some examples are provided below). Recently, some of the most widespread "novel" metaphor-based metaheuristics have been subjected to rigorous analyses, which have provided compelling evidence that, rather than being novel, they are either copies or minor variations of well-established metaheuristics and that their only novelty is in the use of new metaphors and terminology [25–27,29–34].

In particular, three main problems have been identified in papers proposing "novel" metaphor-based metaheuristics [53]. First, they introduce useless metaphors that lack a scientific basis—e.g., zombies, reincarnation, and intelligent water drops—and use new terminology that makes it difficult to understand the ideas being proposed. Second, they lack meaningful novelty, as typically the ideas proposed are already known. Third, they use poor experimental validation and comparison practices, such as comparing "novel" metaheuristics run on recent computers against old algorithms run on old computers, and they use benchmark testbeds that contain biases that can be exploited by the "novel" metaheuristics, thus favoring their performance. An example of such unfair comparison was provided in [54], in which it was experimentally demonstrated that the "novel" slime mold, butterfly, and Harris hawks metaheuristics, among others, make use of center-bias operators that increase their efficiency when the testbed includes problems that have the optimal solution located in the center of the search space.

The negative consequences of the "novel" metaphor-based metaheuristics trend extend well beyond the existence of a few algorithms that were inspired by far-fetched behaviors and presented in papers with methodological issues. Hundreds of "novel" metaphor-based metaheuristics have been published in the literature [55] as a result of this trend, which is grounded in unscientific practices. The trend persists largely because

manual design is still the primary method of creating metaheuristics. In the remainder of this subsection, we explain why "novel" metaphor-based metaheuristics are problematic, list some of their negative consequences, and outline the efforts that have been made to address the problem.

### The metaphor rush

Since the mid-2000s, the field of optimization has witnessed a rush to find "interesting" behaviors, mostly examples of natural and social phenomena, that can be used to devise "novel" metaheuristics. In the numerous papers proposing this kind of metaheuristic, the authors use the following sequence of steps: (a) They start by claiming to have found a new behavior that has applications in optimization, (b) they present why, in their opinion, the behavior is interesting, and provide an extensive list of other "novel" metaheuristics based on "interesting" behaviors, (c) they describe their proposed metaheuristic using the terminology of the behavior instead of the terminology that is typically used in optimization, and (d) they compare the "novel" metaheuristic with other optimization techniques, often ones that are old and whose performance is much worse than the state of the art.

Like other authors, such as [56], we avoid citing publications that propose faulty metaphor-based metaheuristics. The reader is invited to visit the Evolutionary Computation Bestiary [55] for a list of the metaheuristics discussed here or to search for specific metaheuristics on Google Scholar using their names (which are given in italics in the text). As a concrete example, we consider the grey wolf optimizer. According to its authors, this metaheuristic is inspired by "the way grey wolves organize for hunting," following a "strict social hierarchy." To describe the metaheuristic, the authors introduce a new terminology in which candidate solutions are referred to as "wolves," the three best solutions in the "pack" (i.e., the set of candidate solutions) are referred to as the "alpha," "beta," and "delta" wolves, and the optimum of the problem is the "prey" the wolves are hunting. As shown in [30,32], the grey wolf optimizer metaheuristic is based on the idea of computing, at each iteration, the centroid of a hypertriangle whose vertices are the positions of the three best solutions (i.e., "alpha," "beta," and "delta"). Then, the computed centroid is used to bias the movement of the remaining solutions (i.e., the "pack"). The grey wolf optimizer metaheuristic was evaluated on a set of 29 continuous functions, all with low dimensionality and/or with the optimum at the center of the search space, as well as on some classic engineering design problems. When it was compared with PSO, differential evolution (DE) [57], and covariance matrix adaptation evolution strategies (CMA-ESs) [54], it was found to have similar performance. This similarity in performance is not surprising, considering that the "novel" grey wolf optimizer metaheuristic was later shown to be a variant of PSO [30,32].

Another example of a popular "novel" metaphor-based metaheuristic is cuckoo search, which is described using the metaphor of the "cuckoo's parasitic behavior." In this metaheuristic, initial solutions are referred to as "cuckoos," while solutions that have been perturbed are referred to as "eggs." The cuckoo search metaheuristic is based on the idea that some "cuckoos" lay "eggs" in the "nests" of other birds (which are random points in the search space) and only some of these "eggs" will still exist in the next iteration. The authors of cuckoo search compared their metaheuristic against a genetic algorithm and the first version of PSO. The comparison was performed using 13 continuous functions that all had an optimum solution at the center of the search space. Because the authors did not provide any information about the dimensionality of the functions, it was initially difficult to evaluate the quality of their results. However, cuckoo search turned out to be an evolutionary strategy that uses the recombination mechanism of DE [31] and therefore does not advance the state of the art.

### Consequences of the metaphor rush

Approximately 500 papers proposing "novel" metaphor-based metaheuristics have been published [33,58]. The authors of many of these papers claimed to be proposing a novel technique inspired by some sort of "intelligent" behavior or even that they opened up a new avenue of research. See for example "PostDoc: The Human Optimization," by Satish Gajawada. Notwithstanding increasing awareness of the problems that these "novel" metaheuristics are causing in the field, some members of the metaheuristics community continue to actively propose more metaheuristics of this kind.

One of the main problems caused by the publication of papers proposing "novel" metaheuristics has been the fragmentation of the literature into dozens of barely distinguishable niches [53,56]. A direct consequence of this fragmentation is a confusing literature, in which the same ideas and concepts are repeatedly reintroduced using different terminologies derived from the use of new metaphors. This, in turn, makes the comparison of metaheuristics increasingly challenging. For example, it is difficult to compare the optimization capabilities of "grey wolves hunting" with those of "cuckoos laying eggs." Moreover, when one analyzes the mathematical models proposed for these metaheuristics, they turn out to be either copies or minor variations of optimization techniques published many years previously.

The publication of hundreds of "novel" metaphor-based metaheuristics has created the impression that every simplistic mathematical model based on an "interesting" behavior deserves to be added to the metaheuristics literature. Although it is the responsibility of scientific venues to ensure that their output is of scientific value, articles with methodological flaws and a lack of scientific rationale are being peer-reviewed and published despite their inability to contribute meaningfully to the field. Consider, for example, the following excerpt from the intelligent water drops paper: "In nature, we often see water drops moving in rivers, lakes and seas … We also know that the water drops have no visible eyes to be able to find the destination (lake or river)." See [28,29] for a rigorous analysis of this metaheuristic.

Finally, "novel" metaphor-based heuristics damage the reputation of the field, as documented by Sörensen [43] and parodied in "A Spectral Approach to Ghost Detection," by Maturana and Fouhey. In particular, they damage the fields of swarm intelligence and evolutionary computation, which are used to justify the use of metaphors. The truth is that, although metaphors can guide the design of successful metaheuristics, they do so rarely.

### Efforts to mitigate the metaphor rush

Efforts to stop the spread of "novel" metaphor-based metaheuristics mainly consist of raising awareness of how they negatively affect the field. One of the earliest efforts was "A Rigorous Analysis of the Harmony Search Algorithm" by Dennis Weyland [25], which showed, by means of a component-by-component comparison, that harmony search is an evolutionary algorithm. The paper also identified systemic problems that allowed harmony

search to become popular despite its lack of novelty. Another notable example is "Metaheuristics—The Metaphor Exposed" by Sörensen [43], the first paper clearly attempting to call attention to the "metaphor problem" in the field of metaheuristics.

Although these papers barely resonated outside the community that was already aware of the problem, they encouraged other researchers to act and propose possible solutions. Most of these efforts can be categorized as follows: (a) critical analysis of metaphor-based metaheuristics; (b) modeling frameworks, taxonomies, and metaphor-free descriptions; and (c) editorial policies.

In category (a), we find efforts to clarify whether there is any real novelty in "novel" metaheuristics and to obtain insights into the reasons the authors used a particular metaphor. Component-based analyses, similar to that of Weyland [25], have been conducted for the following "novel" metaheuristics: biogeography-based optimization [59], black hole optimization [27], intelligent water drops [28,29], the grey wolf optimizer, the moth-flame optimization algorithm, whale optimization, the firefly algorithm, the bat algorithm, the antlion optimizer [30,32], and cuckoo search [31]. The conclusions of these analyses are clear: There is no novelty in any of these "novel" metaheuristics. In a critical study with a slightly different focus, Melvin et al. [60] demonstrated that the gravitational search algorithm fails as a metaphor because it is based on a mathematical model that is inconsistent with Newtonian gravity. The failure of the gravitational search algorithm shows that using new metaphors without a sound motivation to do so may result in ineffective metaheuristics.

Category (b) includes taxonomies and modeling frameworks that group metaheuristics based on patterns in their design [26,33,34,61–64]. The goal of these efforts is to provide the metaheuristics community with a tool that can identify the components that make up a "novel" metaheuristic and, consequently, reveal whether it is actually novel. The main challenge is to incorporate a sufficiently large number of components. While the first steps in this direction have been taken, this is a huge endeavor.

Also in category (b), we find papers that examine the way metaphor-based metaheuristics work and their relationship to other metaheuristics. Examples include papers by Lones [65,66], which describe some of the "novel" metaphor-based metaheuristics using metaphor-free terminology. In addition, an increasing number of papers aim to quantify the problem by compiling lists of metaphor-based metaheuristics and/or analyzing their performance [54,55,58,67,68].

Finally, category (c) contains editorial policies that explicitly forbid the submission of papers proposing metaphor-based metaheuristics unless the authors can provide compelling evidence that the use of the metaphor contributes to the advancement of the state of the art. Some journals that have established this type of policy are *4OR* [69], *Journal of Heuristics* [70], *Swarm Intelligence* [71], *ACM Transactions on Evolutionary Learning and Optimization* [72], and *Engineering Applications of Artificial Intelligence* [73]. Establishing editorial policies is undoubtedly one of the most effective mechanisms for stopping the publication of metaphor-based metaheuristics; however, this approach remains the exception rather than the rule.

## Automatic Design of Metaheuristics

As the need to solve increasingly complex problems more efficiently has grown, so has the need for better and more efficient problem-solving methods. This has motivated researchers to search for alternative design approaches that are not subject to the disadvantages of manual design. One of the main goals of this research has been to reduce the heavy reliance on human algorithm designers that makes the design process biased, time-consuming, and error-prone. Automatic algorithm design methods are a powerful alternative to manual design. These methods eliminate the need for human involvement by exploiting recent advances in automatic algorithm configuration methods.

The automatic design of metaheuristic implementations is a relatively new paradigm in which the creation of a metaheuristic implementation is handled as an optimization problem that consists of finding a combination of metaheuristic components and parameter settings that will perform well when applied to the optimization problem considered. To achieve this, automatic design methods for metaheuristic implementations rely on two main components: a design space—that is, the set of all possible metaheuristic designs that can be obtained by combining metaheuristic components and parameters settings, and an automatic configuration tool (ACT)—that is, a tool that allows the exploration of the design space of the metaheuristic. In recent years, several metaheuristic software frameworks (MSFs) have been proposed that facilitate the automatic design of high-performance metaheuristic implementations.

In the metaheuristics literature, methods that target the design of metaheuristic implementations as an optimization problem are sometimes referred to as hyper-heuristics. A modern definition of the term hyper-heuristic is as follows: "a search method or learning mechanism for selecting or generating heuristics to solve computational search problems" [74]. However, the initial research on hyper-heuristics was not focused on the automatic design of metaheuristic implementations but rather on the selection of a suitable implementation from a portfolio of preexisting metaheuristic implementations, the so-called "heuristics for choosing heuristics" for combinatorial optimization problems. Currently, the automatic design of metaheuristics is approached by hyper-heuristics in the same way as automatic design methods, that is, by defining a metaheuristic design space and using an optimization algorithm to explore it and find a suitable design. In fact, in the vast majority of cases, the only difference between automatic design methods and hyper-heuristics is that hyper-heuristics explore the design space using genetic programming [75,76].

### Metaheuristic design space: Component-based view

The first step in defining a metaheuristic design space is to derive a component-based view of the considered metaheuristic. To do so, the algorithm designer first identifies ways in which the components of a metaheuristic can be implemented (e.g., by studying the different implementations of the metaheuristic that have been proposed in the literature) and then groups them based on their functionality. The components obtained in this manner define the metaheuristic design space and are combined using an ACT (as explained in the next section). The ACT considers these components, which can be numerical, categorical, and subordinate, as parameters to be optimized. Numerical parameters whose values are either real numbers or integers are classical parameters—e.g., the mutation rate in EC, the evaporation rate in ACO, or particle inertia in PSO. Categorical parameters are alternatives for the functionality of a particular component—e.g., the recombination operator in EC, the solution construction rule in ACO, or population topology in PSO. Finally, subordinate parameters are those that are only necessary for particular values of other

parameters—e.g., in ACO, if the MAX − MIN Ant System pheromone update rule is selected, then the subordinate parameters controlling the lower and upper bounds of the pheromone should also be selected. These parameters together form the parameter configuration space **C**, which is used by the configuration tool, as explained in the next section.

## Automatic configuration tools

ACTs were initially developed to automatically select parameter values in parameterized software to maximize the performance of the software [77,78]. However, more general-purpose ACTs that allow the selection of the algorithm components of the implementation have been proposed. The use of ACTs has increased over the last decade, not only because they generate high-performance algorithms that are tailored for a specific problem but also because of increases in the availability of inexpensive computing power, as they can be computationally expensive. The working mechanisms of ACTs are diverse, ranging from experimental design techniques to surrogate model-based approaches. The specific mechanisms implemented in an ACT determine how computationally intensive it is, the types of parameters it can handle, and the types of post-configuration analyses that can be conducted.

The general workflow followed by ACTs is depicted in Fig. 1. Given a parameter configuration space **C**, an iterative process is performed in which the metaheuristic $M$ being configured is executed with different parameter configurations **c** on the set of test instances **I** until a given computational budget $b$ is fully used. The approaches that have been investigated to develop ACTs to date can be categorized as follows.

### *Experimental design techniques*

These are based on the use of statistical techniques to evaluate aspects such as the statistical significance of performance differences; an example of these techniques is CALIBRA [79].

### *Heuristic search techniques*

These consist, as their name suggests, of the application of metaheuristics to handle configuration tasks. Examples include ParamILS [80], which implements an iterated local search in the parameter configuration space, and the work presented in [81], in which CMA-ES [82] is used for a configuration task of numerical parameters.

### *Surrogate model-based techniques*

These aim to predict the shape of the configuration landscape based on previous executions of the algorithm, with the goal of avoiding the waste of executions on unpromising regions. The best-known technique of this type is the sequential model-based algorithm configuration (SMAC) [83].

### *Iterated racing approaches*

These are based on the idea of performing sequential statistical testing using the Friedman test and its related post-tests to create a sampling model that can be refined by iteratively "racing" candidate configurations and discarding those that perform poorly. Various racing algorithms are implemented in the irace package [47].

Although ACTs differ mostly in the way they approach automatic configuration problems and in their generality, there are also practical differences that may be important for users. For example, when used out-of-the-box, iterated racing approaches, such as irace, can impose a higher computational overhead during the configuration process compared to surrogate model-based approaches, such as SMAC, thus making the latter more suitable for configuration scenarios with expensive objective functions. However, irace, SMAC, and other actively maintained ACTs now allow changes to be made to their sampling models to reduce computation time in expensive configuration scenarios or to perform a more intensive search if computation time is abundant. Another important difference between ACTs is the use of early termination mechanisms for poorly performing configurations, which is also called capping or adaptive capping [47,80]. These mechanisms help make more efficient use of the available computation time and are particularly useful for optimization problems involving time-related objective functions. Finally, a common feature of ACTs is that they provide data that can be used for conducting post-configuration analyses, such as parameter importance [84,85] and ablation analysis [86].

## Metaheuristic Software Frameworks

An MSF is a parameterized software tool that implements the design space of a metaheuristic. To automatically generate a metaheuristic implementation, an MSF is used in combination with an

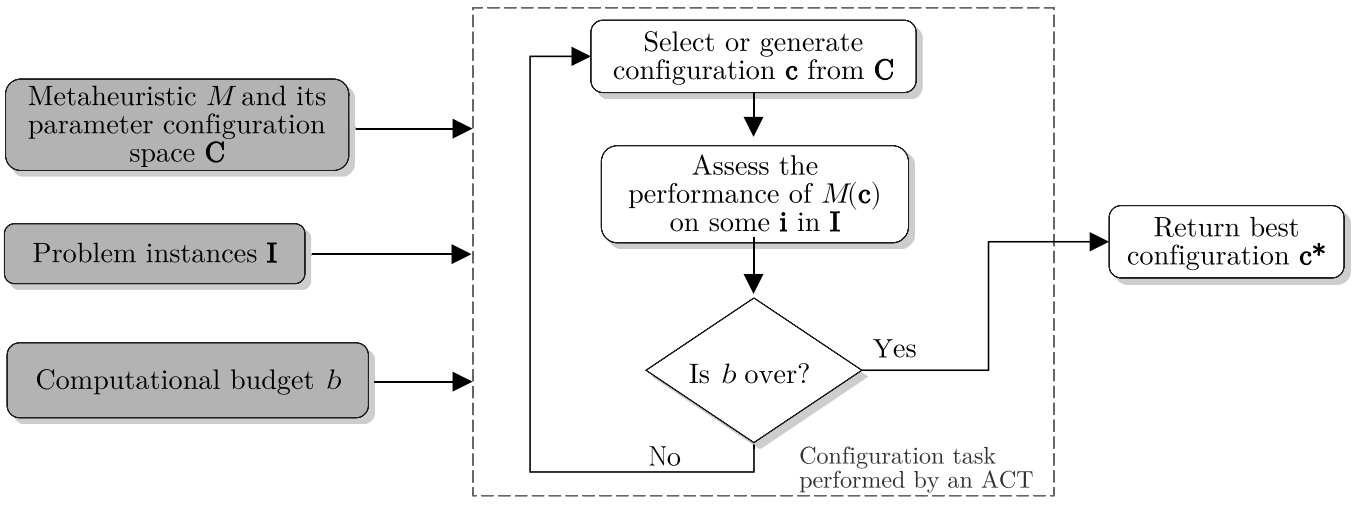


**Fig. 1.** General workflow followed by an automatic configuration tool (ACT) used to configure a metaheuristic.

ACT, which, as explained in the previous section, iteratively executes the MSF with different configurations. The ACT evaluates the performance of each MSF configuration (i.e., metaheuristic implementation) on a set of problem instances until a configuration for the MSF is found that satisfies the needs of the user.

It is important to differentiate between automatic configuration and automatic design. The former refers to fine-tuning the parameter values of an already defined metaheuristic design, whereas the latter refers to composing new metaheuristic designs by recombining their components in new ways in addition to fine-tuning their parameter values. Moreover, automatic configuration and automatic design have different goals. The goal of automatic configuration is to find a high-performance parameter setting for the considered metaheuristic without changing the components of its implementation. In contrast, the goal of automatic design is typically to explore combinations of components and parameter settings that have never been considered.

MSFs proposed in the early days, with few exceptions (such as ParadisEO [87,88], which we discuss below), only enabled the use of ACTs to perform automatic configuration tasks. If users were interested in performing automatic design tasks using these MSFs, they had to make major adaptations to the code of the MSF to extend its metaheuristic design space with new components and rules for combining them. In the worst-case scenario, a complete reimplementation of the MSF was necessary. Over time, the approach to designing MSFs has changed appreciably. In contrast to their earlier counterparts, most modern MSFs strive for a flexible, modular design that allows users to apply them to solve different types of problems and easily extend them with new metaheuristic components and rules for combining them.

The general approach to combining flexible, modular MSFs with ACTs to instantiate ad hoc metaheuristic implementations for specific problems or problem instance distributions is illustrated in Fig. 2. The goal of this approach is to enable the automatic solution of new problems by allowing the configuration tool to find an effective metaheuristic implementation. Therefore, human involvement is necessary only in cases in which one wishes to add new metaheuristic components to the MSF; moreover, this task is typically straightforward, owing to the modular design of the MSF.

The main challenge in creating MSFs is the definition of the rules that control the manner in which metaheuristic components can be combined. There are two main methods to achieve this: algorithm templates and grammar-based programming. Using an algorithm template (or top-down design) consists in creating a parameterized algorithm template in which metaheuristic components are represented as possible alternatives in a typically fixed algorithmic procedure. In contrast, in grammar-based programming (or bottom-up design), the correct combination of components is checked against a grammar, that is, a set of "production rules" that are applied repeatedly. The main difference between the two approaches is that algorithm templates can be much easier to define than grammars but provide limited flexibility in the implementation of metaheuristics (such as component recursion), whereas grammar-based programming can be conceptually more difficult but allows the creation of designs that are much more complex [89].

As shown in Table, several MSFs proposed in the literature enable the automatic design of metaheuristics. The table shows the main types of metaheuristic components included in each software framework and the types of problems it can be used to address. Note that the last four MSFs in the table are of a more general nature because they include components from more than one metaheuristic, and that the list in the table is not exhaustive. Indeed, in a recently published paper [90], some current contributors and maintainers of ParadisEO identified 47 other MSFs that are available online. However, many are closed-source, unmaintained, and/or not aimed at designing new metaheuristic implementations (i.e., they only enable the use of automatic configuration and therefore only optimize the values of numerical parameters).

In the following, we present a more detailed discussion of the last four MSFs listed in Table, namely, ParadisEO, HeuristicLab, jMetal, and EMILI, which are some of the most comprehensive and actively maintained MSFs. After describing the general aspects of each MSF, we provide references to works that explore both technical and conceptual aspects of their use in the context of automatic design.

## ParadisEO

ParadisEO [87,88,90] is a well-known MSF whose initial development dates back to the early 2000s. This MSF includes four main modules that allow users to compose metaheuristic designs: *evolving objects* for population-based metaheuristics, *moving objects* for local search algorithms, *estimation of distribution objects*

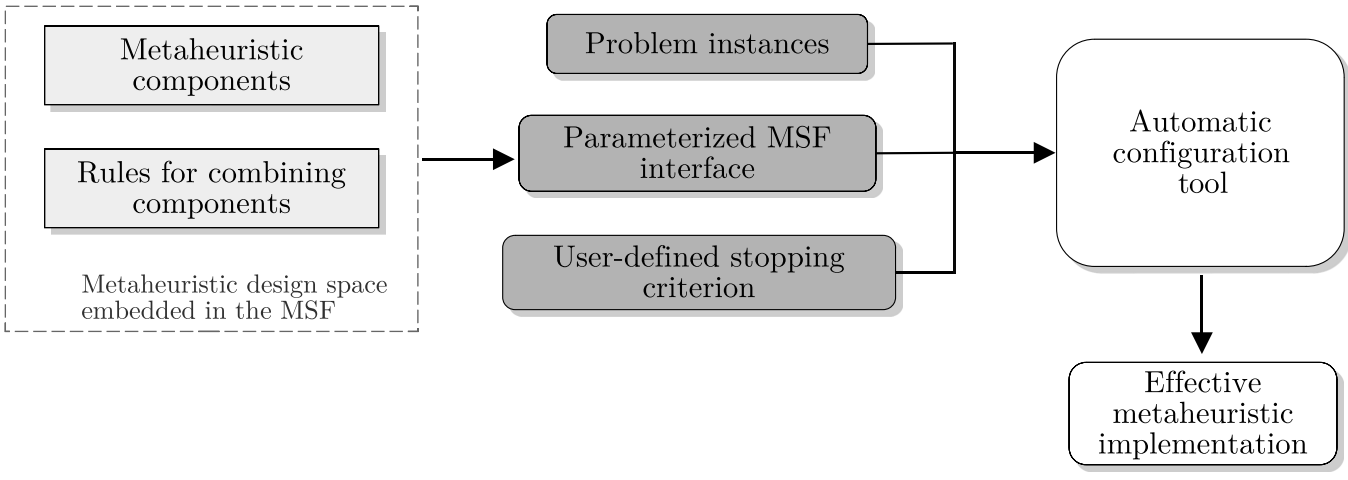


**Fig. 2.** General approach for combining modular MSFs and ACTs.

for estimation of distribution algorithms, and *multiobjective evolving objects* for multiobjective optimization. Some of the key features of ParadisEO are as follows: (a) It has a high runtime speed (as it is implemented in C++), (b) it integrates a state-of-the-art benchmarking and profiling tool called IOHprofiler [91] that simplifies the process of comparing and evaluating implementations against a benchmark, and (c) it has an active community of maintainers. ParadisEO has been applied to solve optimization problems for more than two decades. However, in its early days, it was manually configured and its use in the context of automatic design is something that has only recently been investigated. In [92], the authors studied 19 genetic algorithms for the W-model problem that were automatically generated using ParadisEO and irace. They found that the implementations automatically generated by irace were able to outperform all manually created baseline algorithms and that the fast computations that ParadisEO is able to provide allow large design spaces to be handled in short wall-clock times.

## HeuristicLab

HeuristicLab [93] is an optimization software system developed in the early 2000s that incorporates an MSF. In its current version (version 3.3, released in 2010), the MSF of HeuristicLab has modules for instantiating many different machine learning (ML) algorithms (e.g., neural networks, random forests, and support vector machines) and metaheuristic algorithms (e.g., genetic programming, evolutionary computation, particle swarm optimization, and simulated annealing). In addition to providing an MSF, HeuristicLab also has a number of useful features: (a) It uses a meta-model that allows the representation of arbitrary optimization algorithms, (b) it allows the manipulation and definition of metaheuristic designs via a graphical user interface, (c) it provides easy access to problems that can be used for benchmarking purposes, and (d) it provides interactive charts for the analysis of results. Note that while ParadisEO and EMILI (described below) are implemented in C++, HeuristicLab is implemented in C# and is therefore slower. Except for one paper addressing the algorithm selection problem [94], we could not find any work specifically targeting the automatic design of metaheuristics using HeuristicLab.

## jMetal

jMetal [95], which was developed in 2009, is an optimization software system implemented in Java that incorporates an MSF and has other useful features. jMetal focuses on multi-objective optimization; therefore, it allows the instantiation of many state-of-the-art metaheuristics specialized for multi-objective optimization, such as NSGA-II [96], GDE3 [97], and IBEA [98]. In its current version, jMetal also includes components from several single-objective algorithms, such as DE, particle swarm optimization, and CMA-ES. The main features of jMetal are as follows: (a) It provides a simple graphical user interface that allows the parameters of the metaheuristic implementation to be set; (b) it provides access to five popular testbeds that can be used for benchmarking purposes (e.g., ZDT [99], DTLZ [100], and WFG [101]); (c) it provides some of the most widely used quality indicators in multi-objective optimization, namely, hypervolume [102], spread [96], generational distance [103], inverted generational distance [103], and epsilon [104]; and (d) it offers support for performing experimental studies, including the automatic generation of LaTeX tables, statistical pairwise comparison using the Wilcoxon test, and R boxplots. Examples of the use of jMetal to automatically create metaheuristic implementations include [105] and [106].

## EMILI

EMILI [107], which was initially developed in 2015, is an MSF that implements metaheuristic- and problem-specific components

**Table.** List of representative MSFs for the automatic design of metaheuristic implementations

| Metaheuristic | Name of the MSF | Type of problem | Number of objectives | Year | Reference |
|---|---|---|---|---|---|
| Ant colony optimization | ACO-TSP-QAP | Discrete | Single objective | 2017 | [133] |
| Ant colony optimization | MOACO | Discrete | Multiobjective | 2012 | [134] |
| Ant colony optimization | UACOR | Continuous | Single objective | 2014 | [135] |
| Artificial bee colony | ABC-*X* | Continuous | Single objective | 2017 | [136] |
| Evolutionary computation | ModCMA–ES | Continuous | Single objective | 2021 | [137] |
| Evolutionary computation | DEAP | Discrete/continuous | Single/multiobjective | 2012 | [138] |
| Evolutionary computation | AutoMOEA | Discrete/continuous | Multiobjective | 2015 | [48] |
| Hybrid of particle swarm optimization and differential evolution | PSO–DE | Continuous | Single objective | 2020 | [139] |
| Particle swarm optimization | PSO-*X* | Continuous | Single objective | 2021 | [140] |
| Particle swarm optimization | MOPSO | Continuous | Multiobjective | 2022 | [106] |
| Randomized local search | SATenstein | Discrete | Single objective | 2009 | [141,142] |
| Multiple metaheuristics | ParadisEO | Discrete/continuous | Single/multiobjective | 2002 | [87,88,90] |
| Multiple metaheuristics | HeuristicLab | Discrete/continuous | Single/multiobjective | 2005 | [93] |
| Multiple metaheuristics | jMetal | Discrete/continuous | Single/multiobjective | 2010 | [95] |
| Multiple metaheuristics | EMILI | Discrete/continuous | Single objective | 2019 | [107] |

for stochastic local search algorithms. In its current version, EMILI is mostly used for single-solution metaheuristics (e.g., iterated local search, tabu search, and simulated annealing); however, its design makes it easily extensible to population-based metaheuristics. The distinguishing characteristic of EMILI is its architecture, which uses a grammatical representation to validate possible combinations of algorithm components. The components that make up the metaheuristic implementation and the order in which they will be executed are checked against a grammar and then encoded as a character string so that only valid combinations are produced. Then, EMILI translates the character string into a parametric form that can be executed by an ACT. Other important features of EMILI are as follows: (a) It implements a strict separation between algorithm- and problem-related components, and (b) it can consider algorithms as recursive metaheuristic components. So far, the two most relevant works using EMILI to automatically create metaheuristic designs and implementations are [107], which is focused on hybrid stochastic local search algorithms for permutation flowshop problems, and [108], which is focused on simulated annealing for the quadratic assignment and permutation flowshop problems.

## Discussion

In their 2017 contribution to the Handbook of Metaheuristics, "A history of metaheuristics" [109], Kenneth Sörensen, Marc Sevaux, and Fred Glover predicted that the next transition in the development of metaheuristics would be toward a scientific period. Although it seems counterintuitive to predict a scientific period in a scientific field, they did so because of past research that is extremely unscientific. Most of this unscientific research is related to the "novel" metaphor-based metaheuristics discussed in "Manual Design of Metaheuristics." This trend persists because a large community is actively "researching" these kinds of algorithms. Moreover, a number of papers have been published, which present "novel" metaheuristics in a positive light, ignoring well-founded criticisms and/or taking such criticisms out of context. See, e.g., "An exhaustive review of the metaheuristic algorithms for search and optimization: Taxonomy, applications, and open challenges," by Rajwar, Deep, and Das.

We strongly believe that once the metaheuristics community examines the trend of "novel" metaphor-based metaheuristics and related research scientifically, papers proposing "novel" metaheuristics will be withdrawn from journals and conferences and the trend will vanish, leaving only a cautionary tale. A good reason to call for a more scientific view is to attempt to unify an increasing body of research that is at risk of becoming fragmented. Adopting a scientific view as the baseline for a field that is still expanding is the best way to prevent the (re)appearance of detrimental trends in which personal beliefs can override rational thinking.

One of the most recent attempts to steer the metaheuristics community in a more scientific direction has been the "Metaheuristics in the Large" community project [110]. In this project, several prominent researchers presented their long-term vision for the field, which consists of three main conceptual underpinnings: (a) extensible and reusable framework templates—i.e., modern MSFs, as described above; (b) white-box problem descriptions—i.e., the use of analytic information to guide metaheuristic selection/construction in an informed manner; and (c) remotely accessible frameworks, components, and problems—i.e., the creation of service-oriented architectures that enable the widespread reuse of data and programs.

Another attempt to refocus the field is a recent open letter titled "Metaphor based metaheuristics, a call for action: The elephant in the room" [53], which brought together approximately 100 researchers who want to stop the publication of "novel" metaphor-based metaheuristics by adopting concrete actions, such as calling for scientific journals to establish clear editorial policies concerning how to manage articles presenting this type of metaheuristics. This open letter has been one of the most compelling efforts conducted so far to increase awareness of the problem of "novel" metaphor-based metaheuristics.

Although these documents have already helped steer the field in a healthier direction, much remains to be done. In particular, it seems that more effort is required to bring the metaheuristics community together to address issues that have remained unresolved for years. We propose three methods for removing unscientific approaches from the field of metaheuristics: (a) increasing the amount of research that is either experimentally or theoretically driven, (b) improving the way metaheuristics are benchmarked, and (c) changing the current mainstream approach to creating metaheuristics.

### Rethinking the focus of the research

Metaheuristics research is an applied science that deals with the design and application of optimization algorithms that work well regardless of the complexity of the considered problems. As such, the field has always had a strong bias toward application-oriented research and has extensively used "competitive testing" to make claims about algorithm performance [111]. Competitive testing measures and compares the performance of the compared algorithms for a given set of problem instances and for a given performance measure [112] but does not explain differences in performance. Therefore, to advance the field and increase our knowledge of why some techniques work well on some problems and not on others, it is key to reduce the asymmetry between the amount of research that is application-oriented and that which is experimentally or theory-driven.

Studying the interplay between the computational models underlying a metaheuristic and its performance on different problem classes is important not only from a research perspective but also from a practical perspective. Experimental and theoretical analyses allow us to, for example, (a) understand how extensible the ideas involved in a metaheuristic are and, therefore, to know how they can be used to address other problems; (b) guide the development of new metaheuristic components that can further improve performance or solve known issues; and (c) define guidelines for creating metaheuristic implementations, i.e., useful indications of the ways in which metaheuristic components can be combined so that new designs can be created and tested more easily.

In the last few years, this area seems to have seen improvement, as application-oriented papers that introduce new optimization algorithms motivated by empirical evidence or theoretical findings are now much more frequent. However, important challenges still need to be overcome regarding the way research is conducted in this field, starting with the methodological practices that we use to draw conclusions from experimental studies.

### Rethinking the way we benchmark metaheuristics

Carl Sagan popularized the phrase "extraordinary claims require extraordinary evidence" [113]. In the field of metaheuristics, there is great variety in the way evidence (i.e., data) is collected and used to draw conclusions about metaheuristic performance—a

process commonly referred to as benchmarking [112]. At one end of the spectrum, there are articles about "novel" metaphor-based algorithms that not only make extraordinary claims but also are textbook examples of poor scientific practice (see the discussion in "The 'novel' metaphor-based metaheuristic problem"). At the other end of the spectrum, there are researchers creating new tools to evaluate metaheuristic performance [114] and investigating new methodologies to compare metaheuristics using modern techniques such as deep statistical analysis [115].

In general, although state-of-the-art methodologies and tools for evaluating optimization algorithms are available in the literature [115–118], they are not always used to evaluate metaheuristic performance. For example, it is still common to observe structurally biased metaheuristics that are evaluated on biased test sets [54] and flawed experimental methodologies that present unfair comparisons, do not guarantee reproducibility, and neglect important performance metrics [111,119–123]. To address these issues, higher scientific standards would need to be implemented and systematically enforced by the outlets in which the literature on metaheuristics is published.

Recently, several researchers have proposed a set of guidelines and best practices to address poor benchmarking practices [112]. Among these guidelines and best practices are the following: (a) clearly specifying the goal of the benchmark study and designing it accordingly; (b) using benchmarks that are comprehensive in terms of the size, difficulty, and diversity of the problems; (c) using manual or automatic techniques to configure the parameters of the metaheuristic implementation; (d) using sound statistical methodologies to decide which experiments should be conducted, how many times each experiment should be repeated, which data should be gathered, and how it should be processed, analyzed, interpreted, and presented; and (e) avoiding generalizing the results without sufficient evidence or without defining clear bounds within which such generalization applies.

In addition to these guidelines and best practices, the authors identified several opportunities and open issues in the research on metaheuristic benchmarking. For example, the effort to create/update testbeds on a regular basis to reflect the complexity found in ever-changing realistic scenarios is underestimated. Moreover, there is a need to simultaneously measure different performance metrics (e.g., anytime behavior versus fixed budget, constraint violation costs, and robustness) to provide a better picture of the behavior of a metaheuristic when considering different objectives. Further, there is a need to implement sound data management practices that allow the storage, sharing, and reuse of data from benchmark studies.

It is impossible to overstate the importance of research focused on improving the way metaheuristics are compared and evaluated. However, the widespread implementation of sound benchmarking practices has proven relatively challenging for a field that is experimental in nature and has focused mostly on testing metaheuristics as if they were horses in a race for most of its history. Indeed, despite efforts devoted to improving experimental practices [111,119–122], good practices have not yet become widely adopted, particularly in venues where "novel" metaphor-based metaheuristics are regularly published.

### Rethinking the way we create metaheuristics

Based on the discussion in this article, we believe that the research community should move from manual to automatic design as the main method of creating metaheuristics. Doing so would require a focus on the creation of flexible, automatically configurable MSFs that can be extended with new metaheuristic components so that other researchers/practitioners can use them in different contexts. The long-term goal of this approach is to automate the process of creating metaheuristics so that when a new problem arises, an effective metaheuristic implementation can be automatically created in a timely and unbiased manner. In this article, we discussed the main aspects of the automatic design approach; readers interested in learning more about how to use it, both conceptually and in practice, can refer to [35,124,125], which complement the references given in the "Metaheuristic Software Frameworks" section.

In addition to automatic design, several research trends have emerged, which explore methods of integrating ML into metaheuristics [126–129] and of using data science tools to analyze their performance [86,130,131]. The literature has identified different levels of ML integration [127,129], such as problem-level integration—where ML can aid in modeling aspects of the optimization problem (e.g., the objective function and constraints) and in performing fitness landscape analysis, algorithm-level integration—where ML is used to select a suitable algorithm from an algorithm portfolio, and component-level integration—where ML is used to automate the task of selecting and fine-tuning the algorithm components that perform best for a particular problem. Data analytics tools [e.g., functional analysis of variance (ANOVA) [130], forward selection [131], and ablation [86]] have been used to obtain knowledge about algorithm performance from the data collected during the design process.

The productive synergy created between ML, data science, and metaheuristics has resulted not only in new ways to design and implement increasingly effective algorithms but also in new ways to study these techniques and understand why specific designs perform well, whereas others do not. Although in the literature is already available a vast diversity of metaheuristic techniques and mechanisms for creating new designs, many opportunities remain. For example, we consider it particularly promising to devote effort to (a) creating modeling frameworks that allow better characterization of metaheuristics [34]; (b) developing advanced methods of benchmarking metaheuristics, such as the creation of readily accessible statistical tools [114,115]; and (c) extending the existing ecosystem of MSFs and investigating ways to increase the re-usability of their components [132].

## Conclusions

Over the last few decades, metaheuristics have been the method of choice for finding approximate solutions to difficult optimization problems. However, although they have allowed important advances in the optimization field, the majority of metaheuristics are created in the same way as in the early days—that is, they are the result of a time-consuming, error-prone process in which a human designer manually creates the components to be used in the metaheuristic implementation. Although this method of creating metaheuristics has been successful in the past, it is now time to move toward a new method of creating metaheuristics that avoids the pitfalls of manual design. In this article, we examined an alternative approach, called automatic design, in detail.

The automatic design of metaheuristics is based on the use of a component-based view to define a metaheuristic design space and the application of ACTs to explore different designs until a design that satisfies the needs of the user is found.

Research on the automatic design of metaheuristics has already led to several MSFs that enable the use of ACTs to efficiently design high-performance implementations.

We also discussed the problematic trend of "novel" metaheuristics based on a wide variety of metaphors, which exists in part due to the prevalence of manual design as the primary method for creating metaheuristics. To illustrate why metaphor-based metaheuristics are problematic, we chose two examples of highly-cited "novel" metaheuristics from among many in the literature and showed how they turned out to lack any novelty and are therefore only a source of confusion and reiteration of known ideas.

Finally, we discussed three fundamental research directions that can contribute to further advances in the field of metaheuristics: (a) focus on experimentally or theoretically driven research rather than purely application-driven research and competitive testing, (b) use state-of-the-art benchmarking practices to evaluate metaheuristics, and (c) use modern tools and mechanisms to automatically create high-performance metaheuristic implementations.

In this paper, we focused particularly on the last of the three fundamental aspects, which argues for changing the way metaheuristic implementations are created. The area of automatic design is growing rapidly, and it seems only a matter of time before modern automatic design methods become widespread and replace manual design as the mainstream approach to designing new metaheuristics. We believe that this will help to put a definitive end to the trend of "novel" metaphor-based metaheuristics and will have a positive, long-lasting effect on the way we see, understand, and apply these optimization algorithms.

## Acknowledgments

We thank the two anonymous reviewers for taking the time and effort necessary to review the manuscript and provide valuable comments and suggestions to improve it.

**Funding:** C.L.C.-V., T.S., and M.D. acknowledge support from the Belgian F.R.S.-FNRS, of which they are, respectively, FNRS Aspirant and Research Directors.

**Author contributions:** All authors contributed equally to the writing of the manuscript.

**Competing interests:** The authors declare that they have no competing interests.

## Data Availability

Not applicable.

## References

1. Luenberger DG, Ye Y. *Linear and nonlinear programming*. Cham: Springer; 2016.
2. Andréasson N, Evgrafov A, Patriksson M. *An introduction to continuous optimization: Foundations and fundamental algorithms*. Mineola (New York): Courier Dover Publications; 2020.
3. Garey MR, Johnson DS. *Computers and intractability: A guide to the theory of NP-completeness*. San Francisco (CA): Freeman & Co, 1979.
4. Papadimitriou CH, Steiglitz K. *Combinatorial optimization – Algorithms and complexity*. Englewood Cliffs (NJ): Prentice Hall; 1982.
5. Tovey CA. Tutorial on computational complexity. *Interfaces*. 2002;32:30–61.
6. Glover F. Future paths for integer programming and links to artificial intelligence. *Comput Oper Res*. 1986;13:533–549.
7. Metaheuristics Network. Project Summary, http://www.metaheuristics.org/. Version visited last on 2023 March 26.
8. Fogel DB, Owens AJ, Walsh MJ. *Artificial intelligence through simulated evolution*. New York City (New York): John Wiley & Sons; 1966.
9. Holland JH. *Adaptation in natural and artificial systems*. Ann Arbor (Michigan): University of Michigan Press; 1975.
10. Rechenberg I. *Evolutionsstrategie: Optimierung technischer systeme nach prinzipien der biologischen evolution*. Stuttgart (Germany): Frommann-Holzboog; 1973.
11. Schwefel HP. *Numerische optimierung von computer–modellen mittels der evolutionsstrategie*. Basel (Switzerland): Birkhäuser; 1977.
12. Schwefel HP, *Numerical optimization of computer models*. Hoboken (New Jersey): John Wiley & Sons Inc.; 1981.
13. Glover F. Tabu search—Part I. *INFORMS J Comput*. 1989;1:190–206.
14. Glover F. Tabu search—Part II. *INFORMS J Comput*. 1990;2:4–32.
15. Kirkpatrick S. Optimization by simulated annealing: Quantitative studies. *J Stat Phys*. 1984;34:975–986.
16. Černý V. A thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *J Optim Theory Appl*. 1985;45:41–51.
17. Dorigo M. Ant algorithms solve difficult optimization problems. In: Kelemen J, editor. *Advances in Artificial Life: 6th European Conference*—ECAL 2001. Berlin (Germany): Springer; 2001. p. 11–22.
18. Dorigo M, Stützle T. *Ant colony optimization*. Cambridge (MA): MIT Press; 2004.
19. Dorigo M. Ant colony optimization. *Scholarpedia*. 2007;2(3):1461.
20. Kennedy J, Eberhart R. Particle swarm optimization. Paper presented at: Proceedings of ICNN'95-International Conference on Neural Networks. IEEE; 1995; Perth, WA, Australia. p. 1942–1948.
21. Eberhart R, Kennedy J. A new optimizer using particle swarm theory. Paper presented at: Proceedings of the Sixth International Symposium on Micro Machine and Human Science. 1995; Nagoya, Japan. p. 39–43.
22. Kennedy J, Eberhart RC, Shi Y. *Swarm intelligence*. San Francisco (CA): Morgan Kaufmann Publishers; 2001.
23. Ramalhinho Lourenço H, Martin O, Stützle T. Iterated local search. In: Glover F, Kochenberger G, editors. *Handbook of metaheuristics*. Norwell (MA): Kluwer Academic Publishers; 2002. p. 321–353.
24. Hoos HH, Stützle T. *Stochastic local search: Foundations and applications*. Amsterdam (The Netherlands): Elsevier; 2004.
25. Weyland D. A rigorous analysis of the harmony search algorithm: How the research community can be misled by a "novel" methodology. *Int J Appl Met Comput*. 2010;12:50–60.
26. Thymianis M, Tzanetos A. Is integration of mechanisms a way to enhance a nature-inspired algorithm? *Nat Comput*. 2022;1:1–21.
27. Piotrowski AP, Napiorkowski JJ, Rowinski PM. How novel is the "novel" black hole optimization approach? *Inf Sci*. 2014;267:191–200.

28. Camacho-Villalón CL, Dorigo M, Stützle T. Why the Intelligent Water Drops Cannot Be Considered as a Novel Algorithm. In: Dorigo M, Birattari M, Blum C, Christensen AL, Reina A, Trianni V, editors. *Swarm Intelligence, 11th International Conference, ANTS 2018*. Heidelberg (Germany): Springer; 2018. p. 302–314.
29. Camacho-Villalón CL, Dorigo M, Stützle T. The intelligent water drops algorithm: Why it cannot be considered a novel algorithm. *Swarm Intell*. 2019;13:173–192.
30. Camacho-Villalón CL, Stützle T, Dorigo M. Grey wolf, firefly, bat algorithms: three widespread algorithms that do not contain any novelty. In: Dorigo M, Stützle T, Blesa MJ, Blum C, Hamann H, Heinrich MK, Strobel V, editors. *Swarm Intelligence, 12th International Conference, ANTS 2020*. Heidelberg (Germany): Springer. 2020:121–33.
31. Camacho-Villalón CL, Dorigo M, Stützle T. An analysis of why cuckoo search does not bring any novel ideas to optimization. *Comput Oper Res*. 2022;142:Article 105747.
32. Camacho-Villalón CL, Dorigo M, Stützle T. Exposing the grey wolf, moth-flame, whale, firefly, bat, and antlion algorithms: Six misleading optimization techniques inspired by bestial metaphors. *Int Trans Oper Res*. 2022;30(2):13176.
33. Tzanetos A, Dounias G. Nature inspired optimization algorithms or simply variations of metaheuristics? *Artif Intell Rev*. 2021;54(3):1841–1862.
34. de Armas J, Lalla-Ruiz E, Tilahun SL, Voß S. Similarity in metaheuristics: A gentle step towards a comparison methodology. *Nat Comput*. 2022;21:265–87.
35. Stützle T, López-Ibáñez M. Automated design of metaheuristic algorithms. In: Gendreau M, Potvin JY, editors. *Handbook of metaheuristics*. New York City (New York): Springer; 2019. p. 541–579.
36. Kirkpatrick S, Gelatt CD, Vecchi MP. Optimization by simulated annealing. *Science*. 1983;220(4598):671–680.
37. Bonabeau E, Dorigo M, Theraulaz G. *Swarm intelligence: From natural to artificial systems*. New York: Oxford University Press; 1999.
38. Blum C, Merkle D. Swarm intelligence—Introduction and applications. In: Blum C, Merkle D, editors. *Natural computing series*. Berlin (Germany): Springer Verlag; 2008.
39. Dorigo M, Birattari M. Swarm intelligence. *Scholarpedia*. 2007;2(9):1462.
40. Goldberg DE. *Genetic algorithms in search, optimization and machine learning*. Boston (MA): Addison-Wesley; 1989.
41. Blum C, Roli A. Hybrid metaheuristics: An introduction. In: Blum C, Blesa MJ, Roli A, Sampels M, editors. *Hybrid metaheuristics: An emergent approach for optimization*. Berlin (Germany): Springer; 2008. p. 1–30.
42. Talbi EG. Hybrid metaheuristics. Heidelberg (Germany): Springer Verlag; 2013.
43. Sörensen K. Metaheuristics—The metaphor exposed. *Int Trans Oper Res*. 2015;22(1):3–18.
44. Dorigo M. Optimization, learning and natural algorithms [thesis]. Dipartimento di Elettronica, Politecnico di Milano, Italy; 1992.
45. Dorigo M, Maniezzo V, Colorni A. *The ant system: An autocatalytic optimizing process*. Technical report 91-016. Revised. Politecnico di Milano, Italy: Dipartimento di Elettronica; 1991.
46. Dorigo M, Maniezzo V, Colorni A. Ant system: Optimization by a Colony of cooperating agents. *IEEE Trans Syst Man Cyber Part B*. 1996;26:29–41.
47. López-Ibáñez M, Dubois-Lacoste J, Pérez Cáceres L, Stützle T, Birattari M. The irace package: Iterated racing for automatic algorithm configuration. *Oper Res Perspect*. 2016;3:43–58.
48. Bezerra LCT, López-Ibáñez M, Stützle T. Automatic component-wise design of multi-objective evolutionary algorithms. *IEEE Trans Evol Comput*. 2016;20(3):403–827.
49. Talbi EG. A taxonomy of hybrid metaheuristics. *J Heuristics*. 2002;8(5):541–564.
50. Maniezzo V, Boschetti MA, *Stützle T. Matheuristics—Algorithms and implementations*. EURO Advanced Tutorials on Operational Research. Cham: Springer; 2022.
51. Blackwell T, Branke J. Multiswarms, exclusion, and anti-convergence in dynamic environments. *IEEE Trans Evol Comput*. 2006;10(4):459–472.
52. Hansen N, Ros R, Mauny N, Schoenauer M, Auger A. Impacts of invariance in search: When CMA-ES and PSO face ill-conditioned and non-separable problems. *Appl Soft Comput*. 2011;11(8):5755–5769.
53. Aranha C, Camacho-Villalón CL, Campelo F. Metaphor-based metaheuristics, a call for action: The elephant in the room. *Swarm Intell*. 2022;16:1–6.
54. Kudela J. A critical problem in benchmarking and analysis of evolutionary computation methods. *Nat Mach Intell*. 2022;4(12):1238–1245.
55. Campelo F, Aranha C. Evolutionary Computation Bestiary. https://github.com/fcampelo/EC-Bestiary. Version visited last on 2021 March 26.
56. Campelo F, Aranha C. Sharks, zombies and volleyball: Lessons from the evolutionary computation bestiary. In: *LIFELIKE Computing Systems Workshop*. Aachen (Germany): CEUR Workshop Proceedings (CEUR-WS.org); 2021.
57. Storn R, Price K. Differential evolution—A simple and efficient heuristic for global Optimization over continuous spaces. *J Glob Optim*. 1997;11(4):341–359.
58. Ma Z, Wu G, Suganthan PN, Song A, Luo Q. Performance assessment and exhaustive listing of 500+ nature-inspired metaheuristic algorithms. *Swarm Evol Comput*. 2023;77:Article 101248.
59. Simon D, Rarick R, Ergezer M, Du D. Analytical and numerical comparisons of biogeography based optimization and genetic algorithms. *Inf Sci*. 2011;181(7):1224–1248.
60. Melvin G, Dodd TJ, Groß R. Why 'GSA: A gravitational search algorithm' is not genuinely based on the law of gravity. *Nat Comput*. 2012;11:719–720.
61. Fong S, Wang X, Xu Q, Wong R, Fiaidhi J, Mohammed S. Recent advances in metaheuristic algorithms: Does the Makara dragon exist? *J Supercomput*. 2016;72(10):3764–3786.
62. Molina D, Poyatos J, Ser JD, García S, Hussain A, Herrera F. Comprehensive taxonomies of nature-and bio-inspired optimization: Inspiration versus algorithmic behavior, critical analysis recommendations. *Cogn Comput*. 2020;12(1):897–939.
63. Cruz-Duarte JM, Ortiz-Bayliss JC, Amaya I, Shi Y, Terashima-Marín H, Pillay N. Towards a generalised metaheuristic model for continuous optimisation problems. *Mathematics*. 2020;8(11):2046.
64. Stegherr H, Heider M, Hähner J. Classifying metaheuristics: Towards a unified multi-level classification system. *Nat Comput*. 2020;21(5):1–17.
65. Lones MA. Metaheuristics in nature-inspired algorithms. In: Igel C, Arnold DV. *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2014*. New York (NY): ACM Press; 2014. p. 1419–1422.

66. Lones MA. Mitigating metaphors: A comprehensible guide to recent nature-inspired algorithms. *SN Comput Sci*. 2020;1:1–12.
67. Tzanetos A, Fister I Jr, Dounias G. A comprehensive database of nature-inspired Algorithms. *Data Brief*. 2020;31:Article 105792.
68. Kudela J. The evolutionary computation methods no one should use. 2023. arXiv:2301.01984.
69. 4OR—A Quarterly Journal of Operations Research Research papers; https://www.springer.com/journal/10288. Version visited last on 2023 March 19.
70. Journal of Heuristics. Policies on Heuristic Search Research. https://www.springer.com/journal/10732/updates/17199246. Version visited last on 2021 March 26.
71. Dorigo M. Swarm intelligence: A few things you need to know if you want to publish in this journal. https://www.springer.com/cda/content/document/cda_downloaddocument/ Additional_submission_instructions.pdf. Version visited last on 2021 March 26.
72. ACM Transactions on Evolutionary Learning and Optimization. Guidelines for Authors. https://dl.acm.org/journal/telo/author-guidelines. Version visited last on 2021 March 26.
73. Engineering Applications of Artificial Intelligence. Aims & Scope. https://www.sciencedirect.com/journal/engineering-applications-of-artificial-intelligence. Version visited last on 2023 March 3.
74. Burke EK, Gendreau M, Hyde MR, Kendall G, Ochoa G, Özcan E, Qu R. Hyper-heuristics: A survey of the state of the art. *J Oper Res Soc*. 2013;64(12):1695–1724.
75. Koza J. *Genetic programming: On the programming of computers by the means of natural selection*. Cambridge (MA): MIT Press; 1992.
76. Sabar NR, Ayob M, Kendall G, Qu R. Grammatical evolution hyper-heuristic for combinatorial optimization problems. *IEEE Trans Evol Comput*. 2013;17(6):840–861.
77. Nannen V, Eiben AE. A method for parameter calibration and relevance estimation in evolutionary algorithms. In: Keijzer M, Cattolico M. *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2006*. New York (NY): ACM Press; 2006. p. 183–190.
78. Audet C, Orban D. Finding optimal algorithmic parameters using derivative-free Optimization. *SIAM J Optim*. 2006;17(3):642–664.
79. Adenso-Dıéaz B, Laguna M. Fine-tuning of Algorithms using fractional experimental design and local search. *Oper Res*. 2006;54(1):99–114.
80. Hutter F, Hoos HH, Leyton-Brown K, Stützle T. ParamILS: An automatic algorithm configuration framework. *J Artif Intell Res*. 2009;36(1):267–306.
81. Yuan Z, Montes de Oca MA, Stützle T, Birattari M. Continuous optimization algorithms for tuning real and integer algorithm parameters of swarm intelligence algorithms. *Swarm Intell*. 2012;6:49–75.
82. Hansen N, Ostermeier A. Completely derandomized self-adaptation in evolution strategies. *Evol Comput*. 2001;9(2):159–195.
83. Hutter F, Hoos HH, Leyton-Brown K. Sequential model-based optimization for general algorithm configuration. In: Coello CA, editor. *Learning and intelligent optimization, 5th International Conference, LION 5. Lecture Notes in Computer Science*. Heidelberg (Germany): Springer; 2011. p. 507–523.
84. Pérez Cáceres L, Bischl B, Stützle T. Evaluating random forest models for irace. In: Bosman PAN, editor. *GECCO'17 Companion*. New York (NY): ACM Press; 2017. p. 1146–1153.
85. Hutter F, Hoos HH, Leyton-Brown K. An efficient approach for assessing hyperparameter importance. Paper presented at: Proceedings of the 31th International Conference on Machine Learning, ICML 2014; 2014 June 21–26; Beijing, China.
86. Fawcett C, Hoos HH. Analysing differences between algorithm configurations through ablation. *J Heuristics*. 2016;22(4):431–458.
87. Keijzer M, Merelo JJ, Romero G, and Schoenauer M. Evolving objects: A general purpose evolutionary computation library. Paper presented at: Artificial Evolution: 5th International Conference, Evolution Artificielle, EA 2001; 2001 October 29–31; Le Creusot, France.
88. Cahon S, Melab N, Talbi EG. ParadisEO: A framework for the reusable design of parallel and distributed metaheuristics. *J Heuristics*. 2004;10(3):357–380.
89. Mascia F, López-Ibáñez M, Dubois-Lacoste J, Stützle T. Grammar-based generation of stochastic local search heuristics through automatic algorithm configuration tools. *Comput Oper Res*. 2014;51:190–199.
90. Dréo J, Liefooghe A, Verel S. Paradiseo: From a modular framework for evolutionary computation to the automated design of metaheuristics: 22 years of Paradiseo. In: Chicano F, editor. *GECCO'21 Companion*. New York (NY): ACM Press; 2021. p. 1522–1530.
91. Doerr C, Wang H, Ye F, Van Rijn S, and Bäck T. IOHprofiler: A benchmarking and profiling tool for iterative optimization heuristics. 2018. arXiv:1810.05281.
92. Aziz-Alaoui A, Doerr C, Dreo J. Towards large scale automated algorithm design by integrating modular benchmarking frameworks. In: Chicano F, editor. *GECCO'21 Companion*. New York (NY): ACM Press; 2021. p. 1365–1374.
93. Wagner S, Affenzeller M. Heuristiclab: A generic and extensible optimization environment. Paper presented at: Adaptive and Natural Computing Algorithms, Proceedings of the International Conference; 2005; Coimbra, Portugal.
94. Beham A, Wagner S, Affenzeller M. Algorithm selection on generalized quadratic assignment problem landscapes. In: Aguirre H, editor. *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2018*. New York (NY): ACM Press; 2018. p. 253–260.
95. Durillo J, Nebro A, Alba E. The jMetal framework for multi-objective optimization: Design and architecture. In: Ishibuchi H. *Proceedings of the 2010 Congress on Evolutionary Computation (CEC 2010)*. Piscataway (NJ): IEEE Press; 2010. p. 4138–4325.
96. Deb K, Pratap A, Agarwal S, Meyarivan T. A fast and elitist multi-objective genetic algorithm: NSGA-II. *IEEE Trans Evol Comput*. 2002;6(2):182–197.
97. Kukkonen S, Lampinen J. *GDE3: The third evolution step of generalized differential evolution*. In: *Proceedings of the 2005 Congress on Evolutionary Computation (CEC 2005)*. Piscataway (NJ): IEEE Press; 2005. p. 443–450.
98. Zitzler E, Künzli S. Indicator-based selection in multiobjective search. In: Yao X et al. *Proceedings of PPSN-VIII, Eigth International Conference on Parallel Problem Solving from Nature*. Heidelberg (Germany): Springer; 2004. p. 832–842.
99. Zitzler E, Thiele L, Deb K. Comparison of multiobjective evolutionary Algorithms: Empirical results. *Evol Comput*. 2000;8(2):173–195.
100. Deb K, Thiele L, Laumanns M, Zitzler E. Scalable test problems for evolutionary multiobjective optimization. In: Abraham A, Jain L, Goldberg R, editors. *Optimization EM*. London (UK): Springer; 2005. p. 105–145.

101. Huband S, Hingston P, Barone L, While L. A review of multiobjective test problems and a scalable test problem toolkit. *IEEE Trans Evol Comput*. 2006;10(5):477–506.
102. Zitzler E, Thiele L. Multi objective evolutionary Algorithms: A comparative case study and the strength Pareto evolutionary algorithm. *IEEE Trans Evol Comput*. 1999;3(4):257–271.
103. Van Veldhuizen DA, Lamont GB. Evolutionary computation and convergence to a pareto front. In: Koza JR, editor. *Genetic Programming 1998: Proceedings of the Third Annual Conference, Late Breaking Papers*. California: Stanford University; 1998. p. 221–228.
104. Knowles JD, Thiele L, Zitzler E. A tutorial on the performance assessment of stochastic multiobjective optimizers. TIK-Report 214. Revised version. Computer Engineering and Networks Laboratory (TIK)—Swiss Federal Institute of Technology (ETH), Zürich, Switzerland, 2006.
105. Nebro AJ, López-Ibáñez M, Barba-González C, García-Nieto J. Automatic configuration of NSGA-II with jMetal and irace. In: López-Ibáñez M, editor. *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2019*. New York (NY): ACM Press; 2019. p. 1374–1381.
106. Doblas D, Nebro AJ, López-Ibáñez M, García-Nieto J, Coello Coello CA. Automatic design of multi-objective particle swarm optimizers. In: Dorigo M, Hamann H, López-Ibáñez M, García-Nieto J, Engelbrecht A, Pinciroli C, Strobel V, Camacho-Villalón C. *Swarm Intelligence, 13th International Conference, ANTS 2022. Lecture Notes in Computer Science*. Springer; 2022. p. 28–40.
107. Pagnozzi F, Stützle T. Automatic design of hybrid stochastic local search algorithms for permutation flowshop problems. *Eur J Oper Res*. 2019;276:409–421.
108. Franzin A, Stützle T. Revisiting simulated annealing: A component-based analysis. *Comput Oper Res*. 2019;104:191–206.
109. Sörensen K, Sevaux M, Glover F. A history of metaheuristics. In: *Handbook of heuristics*. New York City (New York): Springer; 2018. p. 791–808.
110. Swan J, Adriaensen S, Brownlee AE, Hammond K, Johnson CG, Kheiri A, Krawiec F, Merelo JJ, Minku LL, Özcan E, et al. Metaheuristics' in the large'. *Eur J Oper Res*. 2022;297(2):393–406.
111. Hooker JN. Testing heuristics: We have it all wrong. *J Heuristics*. 1996;1:33–42.
112. Bartz-Beielstein T, Doerr C, Berg Dvd, et al. Benchmarking in optimization: Best practice and open issues. arXiv:2007.03488 2020.
113. Sagan C. *Broca's brain: Reflections on the romance of science*. New York City (New York): Random House; 1979.
114. Eftimov T, Petelin G, Korošec P. DSCTool: A web-service-based framework for statistical comparison of stochastic optimization algorithms. *Appl Soft Comput*. 2020;87(6):Article 105977.
115. Eftimov T, Korošec P, Seljak BK. A novel approach to statistical comparison of metaheuristic stochastic optimization algorithms using deep statistics. *Inf Sci*. 2017;417(C):186–215.
116. Derrac J, García S, Molina D, Herrera F. A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. *Swarm Evol Comput*. 2011;1(1):3–18.
117. Sheskin DJ. *Handbook of parametric and nonparametric statistical procedures*. New York City (New York): Chapman & Hall/CRC; 2011.
118. R Development Core Team. *R: A language and environment for statistical computing*. Vienna (Austria): R Foundation for Statistical Computing; 2008.
119. Hooker JN. Needed: An empirical science of Algorithms. *Oper Res*. 1994;42:201–212.
120. Bartz-Beielstein T, Chiarandini M, Preuss PL. *Experimental methods for the analysis of optimization algorithms*. Berlin (Germany): Springer; 2010.
121. García-Martínez C, Gutiérrez PD, Molina D, Lozano M, Herrera F. Since CEC 2005 competition on real-parameter optimisation: A decade of research, progress and comparative analysis's weakness. *Soft Comput*. 2017;21(19):5573–5583.
122. Campelo F, Takahashi F. Sample size estimation for power and accuracy in the experimental comparison of algorithms. *J Heuristics*. 2019;25(1):305–338.
123. López-Ibáñez M, Branke J, Paquete L. Reproducibility in evolutionary computation. *ACM Trans Evol Learn Optim*. 2021;1(4):1–21.
124. Hoos HH. Programming by optimization. *Commun ACM*. 2012;55(2):70–80.
125. Qu R, Kendall G, Pillay N. The general combinatorial optimization problem: Towards automated algorithm design. *IEEE Comput Intell Mag*. 2020;15(2):14–23.
126. Song H, Triguero I, Özcan E. A review on the self and dual interactions between machine learning and optimisation. *Prog Artif Intell*. 2019;8:143–165.
127. Talbi EG. Machine learning into metaheuristics: A survey and taxonomy. *ACM Comput Surv (CSUR)*. 2021;54(6):1–32.
128. Gambella C, Ghaddar B, Naoum-Sawaya J. Optimization problems for machine learning: A survey. *Eur J Oper Res*. 2021;290(3):807–828.
129. Karimi-Mamaghan M, Mohammadi M, Meyer P, Karimi-Mamaghan AM, Talbi EG. Machine learning at the service of meta-heuristics for solving combinatorial optimization problems: A state-of-the-art. *Eur J Oper Res*. 2022;296(3):393–422.
130. Hooker G. Generalized functional ANOVA diagnostics for high-dimensional functions of dependent variables. *J Comput Graph Stat*. 2012;16(3):709–732.
131. Hutter F, Hoos HH, Leyton-Brown K. *Identifying key algorithm parameters and instance features using forward selection*. In: Learning and Intelligent Optimization, 7th International Conference, LION 7. Ed. by Pardalos PM, Nicosia G. Vol. 7997. Lecture Notes in Computer Science. Springer, Heidelberg, Germany, 2013:364–81.
132. Swan J, Adriænsen S, Barwell AD, Hammond K, White DR. Extending the open-closed principle to automated algorithm configuration. *Evol Comput*. 2019;27(1):173–193.
133. López-Ibáñez M, Stützle T, Dorigo M. Ant colony optimization: A component-wise overview. In: Martıé R, Pardalos PM, Resende MGC. *Handbook of heuristics*. Springer International Publishing; 2017. p. 1–37.
134. López-Ibáñez M, Stützle T. The automatic design of multi-objective ant colony optimization algorithms. *IEEE Trans Evol Comput*. 2012;16(6):861–875.
135. Liao T, Stützle T, Montes de Oca MA, Dorigo M. A unified ant colony optimization algorithm for continuous optimization. *Eur J Oper Res*. 2014;234:3, 597–609.
136. Aydın D, Yavuz G, Stützle T. ABC-X: A generalized, automatically configurable artificial bee colony framework. *Swarm Intell*. 2017;11:1–38.

137. de Nobel J, Vermetten D, Wang H, Doerr C, Bäck T. Tuning as a means of assessing the benefits of new ideas in interplay with existing algorithmic modules. In: Chicano F, editor. *GECCO'21 Companion*. New York (NY): ACM Press; 2021. p. 1375–1384.
138. Fortin FA, De Rainville FM, Gardner MA, Parizeau M, Gagné C. DEAP: Evolutionary algorithms made easy. *J Mach Learn Res*. 2012;13:2171–2175.
139. Boks R, Wang H, Bäck T. A modular hybridization of particle swarm optimization and differential evolution. In: Coello CAC. *GECCO'20 Companion*. New York (NY): ACM Press; 2020. p. 1418–1425.
140. Camacho-Villalón CL, Dorigo M, Stützle T. PSO-X: A component-based framework for the automatic design of particle swarm optimization algorithms. *IEEE Trans Evol Comput*. 2022;26(3):402–416.
141. KhudaBukhsh AR, Xu L, Hoos HH, Leyton-Brown K. SATenstein: Automatically building local search SAT solvers from components. In: Boutilier C, editor. *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence*. Menlo Park (CA): AAAI Press; 2009. p. 517–524.
142. KhudaBukhsh AR, Xu L, Hoos HH, Leyton-Brown K. SATenstein: Automatically building local search SAT solvers from components. *Artif Intell*. 2016;232:20–42.