applied sciences

MDPI

Review

Combinatorial Optimization Problems and Metaheuristics: Review, Challenges, Design, and Development

Fernando Peres and Mauro Castelli *

Campus de Campolide, Nova Information Management School (NOVA IMS), Universidade NOVA de Lisboa, 1070-312 Lisboa, Portugal; fperes@novaims.unl.pt
* Correspondence: mcastelli@novaims.unl.pt

Abstract: In the past few decades, metaheuristics have demonstrated their suitability in addressing complex problems over different domains. This success drives the scientific community towards the definition of new and better-performing heuristics and results in an increased interest in this research field. Nevertheless, new studies have been focused on developing new algorithms without providing consolidation of the existing knowledge. Furthermore, the absence of rigor and formalism to classify, design, and develop combinatorial optimization problems and metaheuristics represents a challenge to the field's progress. This study discusses the main concepts and challenges in this area and proposes a formalism to classify, design, and code combinatorial optimization problems and metaheuristics. We believe these contributions may support the progress of the field and increase the maturity of metaheuristics as problem solvers analogous to other machine learning algorithms.

Keywords: standardization; framework; metaheuristic; combinatorial optimization problems

check for updates

Citation: Peres, F.; Castelli, M. Combinatorial Optimization Problems and Metaheuristics: Review, Challenges, Design, and Development. *Appl. Sci.* **2021**, *11*, 6449. https://doi.org/10.3390/app11146449

Academic Editor: Federico Divina

Received: 21 June 2021
Accepted: 9 July 2021
Published: 13 July 2021

1. Introduction

Combinatorial optimization problems (COPs), especially real-world COPs, are challenging because they are difficult to formulate and are generally hard to solve [1–3]. Additionally, choosing the proper "solver algorithm" and defining its best configuration is also a difficult task due to the existence of several solvers characterized by different parametrizations. Metaheuristics are widely recognized as powerful solvers for COPs, even for hard optimization problems [2,4–8]. In some situations, they are the only feasible approach due to the dimensionality of the search space that characterizes the COP at hand.

Metaheuristics (MH) are general heuristics that work at a meta-level, and they can be applied in a wide variety of optimization problems. They are problem-agnostic techniques [4,9–13] that establish an iterative search process to find an optimal solution for a given problem [6,14,15]. Metaheuristics are extensively used in several areas such as finance, production management, production, healthcare, financial, telecommunication, and computing applications that are NP-hard in nature [2,10,16].

Nevertheless, a determined metaheuristic cannot have a great performance in all categories of optimization problems because its approach may not work well in several other problems, a hypothesis already proved by "No Free Lunch" theorems [14,15,17–19]. These points have "engaged" the research community to develop a vast number of metaheuristics [6,15,19].

Theoretically, metaheuristics are not tied to a specific problem, thus allowing the same heuristic to be used to solve different types of COPs. Metaheuristics can be adapted to incorporate problem-specific knowledge [4]. However, in practical terms, these incorporations always require some level of customization to adapt them to the peculiarities of a COP. For instance, it is not plug-and-play as occurs with Scikit Learn, where any data and parameters are passed as input for the classifiers.

Consequently, this customization requires a set of efforts to effectively ensure that the metaheuristics can properly manipulate the COP solutions' representation, such as methods for exploration and exploitation compatible with the solutions' encoding. Moreover, the COP formulation also needs to provide features (data and behaviors) required by metaheuristics to guarantee the proper functioning of the search process for finding an optimal solution.

The lack of rules and formalisms to standardize the formulation of COPs and a software design pattern to develop metaheuristics makes this activity more challenging and makes it difficult the progress and consolidation of the field. The absence of standardization and the lack of guidelines to organize COP and metaheuristics [5,7,9,12,17,18] make difficult the classification and comparison of metaheuristics' behaviors and performance.

Consequently, this leads researchers to start "almost" from scratch [3,5,9,16,18] by building their own version of the algorithm. Consequently, this situation decreases the potential of COP (as a "piece of software") and metaheuristic reuse. Especially because they were not designed as independent software units, the interoperability among COPs and metaheuristics without code changes is, most of the time, impossible. Because of this, extra efforts are needed to develop or adjust the metaheuristics to deal with the specific problem [3,4,7,20], reducing the time spent on the analysis and tuning the results.

This study aims to provide a thorough review of the COP and MH by presenting the main concepts, algorithms, and challenges. Using these review inputs as a basis, this research proposes a standardization and a framework for COP and MH, named ToGO (Towards Global Optimal). Therefore, this is a review paper accompanied by a framework proposal to provide answers to open questions and well-known issues raised by the research community. From these perspectives, to guide this research, we formulated five research questions. They were conceived as guidelines to conduct the research and organize the concepts, definitions, discussions, decisions, and conclusions.

1.1. Research Objective and Questions

This review paper aims to organize a body of knowledge by arranging the fundamental concepts and building blocks necessary to design and implement COPMH. Consequently, this paper has two main objectives. Section 1 provides a review of the field and Section 2 propose a standardization. In the review, two central points will be researched: in Section 1.1, the most important concepts and definitions (COPMH) and in Section 1.2 the main challenges of the field. For the second objective, this work must consider at least three classes of behavior: combinatorial optimization problem and its building blocks (Section 2.1), metaheuristics interface and interactions (Section 2.2), and typical operations that a framework structure should consider (Section 2.3). To address these objectives, we formulated five research questions. Before proceeding to the description of each research question, it is important to highlight that *RQ-1* and *RQ-2* are concerned with the review objectives of this paper. Moreover, they were designed, especially *RQ-1*, to provide essential inputs to the other research questions (*RQ-3*, *RQ-4*, and *RQ-5*).

RQ-1: What are the fundamental concepts of COP (combinatorial optimization problem) and metaheuristics?

Rationale: The response to this question aims to offer the necessary knowledge to comprehend this research and its fundamental concepts. These concepts are vital for the reader and provide a broad perspective as a starting point in the COP and metaheuristics field. It also must support the understanding of the standard proposed and its definitions in this study.

RQ-2: What are the main challenges of the COP and metaheuristics research field?

Rationale: This question intends to draw attention to fundamental issues of the COP and metaheuristics research field by investigating several papers and other pieces of knowledge. Consequently, this question also can lead to future studies that may require community attention. Additionally, some of these open questions can provide valuable contributions to *RQ-3*, *RQ-4*, and *RQ-5*, considering that some issues can be treated through some guidelines and formalism.

RQ-3: What are the features of COP and metaheuristics that could be developed as independent methods in a framework structure?

Rationale: By answering this question, the main idea is to identify the elements that should be part of a framework structure because they can be reused in other situations by any COP or heuristic (without coding efforts in future problems). Consequently, avoiding the development of components that can be reused. Additionally, it also reduces the efforts to formulate a problem or develop a metaheuristic, make them leaner and simpler to define.

RQ-4: How can we develop a body of knowledge to formulate combinatorial optimization problems in a standard way?

Rationale: This question has many facets. It involves different aspects, which englobes the conceptual and practical perspectives to design and implement COP in a generic way. Therefore, a problem standard must consider a wide range of different problem specificities and several aspects of communication with heuristics and framework methods. The objective behind this is to provide guidelines to formulate COP in a standard way that can be manipulated by any method that knows its standard.

RQ-5: How can we define a standard design for metaheuristics completely without coding links to the problems?

Rationale: The answer to this question should provide a software pattern for metaheuristics, defining a set of behaviors and standard API (Application Programming Interface). Furthermore, a metaheuristics standard must follow a problem standard that also provides its own API. Therefore, it must know how to manipulate the COP by being aware of its behaviors. Consequently, the answer to *RQ-4* will support the answer to this question.

1.2. Out of Scope (Delimitations)

This review paper does not consider publication intensity. Consequently, it does not list the main sources of research such as conferences, proceedings, journals, etc. The list of references provides several papers that may guide the reader through this research area.

2. Materials and Methods

This chapter provides a comprehensive review of the fundamental definitions of combinatorial optimization problems and metaheuristics. Consequently, this section supplies the needed knowledge to the reader to facilitate the understanding of these subjects and the standard proposed in this work. Therefore, this section also aims to answer *RQ-1: What are the main concepts of COP (combinatorial optimization problem) and metaheuristics?*

2.1. Optimization

Optimization can be summarized as a collection of techniques and algorithms [6,8,16,21,22] to find the best possible solution (optimal or near-optimal) iteratively. From a searching approach perspective, they can be organized into three classes of methods: (1) enumerative, (2) deterministic, and (3) stochastic [10,23–26].

The enumerative methods are based on a brute force approach because it inspects all possible solutions in the search space. For an understandable reason, it always will find the optimal global solution. However, computationally speaking, this approach only works in small search spaces. On the other hand, the most challenging problems, with a vast search space, may require an enormous amount of time to list all solutions, something not feasible most of the time. In these cases, different approaches are required.

The deterministic methods can be seen as an extension of the enumerative approach. They use knowledge of the problem to develop alternative ways to navigate search space. Deterministic methods are excellent problem-solvers for a wide variety of situations, but they are problem-dependent. Consequently, they have difficulties when it is not possible to apply the knowledge of the problem effectively. Deterministic methods receive this classification because their parameters' values and the initialization approach determine their result. In other words, if a particular configuration is used, the output solution will always be the same.

The stochastic methods are problem-agnostic and include uncertainty. They move in the search space using a sophisticated navigation model, applying a probabilistic approach—random but guided by probabilities—to explore the search space [27,28]. Since the stochastic methods are based on randomness aspects, the solution returned will almost always be different. Therefore, it is not possible to determine which will be the final solution and its performance [29]. They are an excellent alternative in scenarios where the data are unpredictable, the problem knowledge cannot be used, and the other approaches (enumerative and deterministic) do not have a good performance or are not feasible.

2.2. Optimization Problem

Human beings face difficulties daily—challenging or contradictory situations—that need to be resolved to reach the desired purpose. These situations, in practical terms, can be called problems. Theoretically, from a psychological perspective, a problem can be summed up as difficulty that triggers an analytical attitude of a subject to acquire knowledge to overcome it [30–32]. From a computer science perspective, a problem is a task that an algorithm can solve or answer. From a practical perspective, computational problems can be broadly classified into six main categories of problems: ordering, counting, searching, decision, function, and optimization. Concerning ordering, counting, and searching, their own definitions describe their objectives as problems. A decision problem is a class of problems that decides after the analysis of a situation. The return of this type of problem is an answer (a judgment) that only is determined by two possible outputs {"Yes", "No"} or {1, 0} [30,31]. Whereas a function problem is an extension of the decision, any data type can be returned.

An optimization problem (OP) is a class of problems resolved by optimization methods that aim to find an optimal solution in a vast set of possible solutions [2,23,33–35]. Consequently, the output of an optimization problem is the solution itself.

Definition 1. *An optimization problem can be formalized in the following way:*

$$P = (S,\ f,\ \Omega) \tag{1}$$

where:

- *P represents the optimization problem;*
- *S symbolizes the search space of the problem domain;*
- *f represents the objective function (OF); and*
- *Ω corresponds to the set of problem's constraints.*

The *search space* S is defined through a set of variables $= \{X_1,\ X_2, \ldots,\ X_n\}$, generally called *decision variables* or *design variables*. They can be seen as the data or the dimensions that are explored to search for a *solution*. The *search space* may be subject to a set of *constraints* Ω or restrictions. These restrictions are a set of definitions used to specify whether a *solution* is feasible or not. In other words, they regulate whether a *solution* can be accepted according to the rules of the problem domain. The *f fitness function* or *objective function* is used to assess and determine the quality of the *solutions* and guide the search process.

Definition 2. *A fitness function has as an output a real number ($\mathbb{R}$), often called fitness.*

$$f : S\ \rightarrow \mathbb{R} \tag{2}$$

$$\forall\, s \in S,\ f(s) = fitness\ (\mathbb{R})$$

This number enables the possibility to compare the performance of all possible solutions of a certain *optimization problem* [6,23,34,36].

Combinatorial Optimization Problems

Optimization problems are divided into two categories of problems, (1) continuous and (2) combinatorial problems [35]. Continuous optimization problems are composed of continuous decision variables, and they can generate an infinite number of valid solutions. Combinatorial optimization problems (COP)—also known as discrete optimization problems—are defined by discrete decision variables and their elements. The solution of a COP is represented by an arrangement of these elements. Since a solution is a combination (or permutation) of elements, a COP has a finite number of solutions. The definition of the encoding rules between combinations or permutations will depend on the needs of the problem and its solution representation.

Example 1. *Considering Travel Salesman Problem (TSP), a solution is a route formed by an ordered arrangement of cities. In other words, the order of the cities is essential, and their repetition is not allowed. Therefore, this solution's encoding must follow the permutation rules where the elements cannot be repeated.*

Continuous optimization can use derivatives and gradients to describe the slope of a function [35]. However, combinatorial optimization does not count with this support. Consequently, combinatorial problems generally are more challenging problems to be solved. They are usually NP-hard problems, and there is no deterministic method capable of solving them in polynomial-time [30,37,38]. In real problems, the search space often is vast, and the landscape is commonly unknown. Moreover, the performance of COP methods requires good strategies to balance the exploration and exploitation to navigate the search and find optimal solutions. These characteristics turn combinatorial optimization problems into a challenging activity [38,39].

2.3. Combinatorial Optimization Problem Classification System

After an extensive review of the combinatorial optimization problem (COP)—capturing different aspects explored in various studies—some critical definitions are organized jointly in this paper. The objective is to consolidate several dimensions of COP classification, providing a bigger view of a COP. However, it is noteworthy this work does not aim to create a classification system for COP. Still, it can provide a preliminary view that can be used as an input for future studies with the clear intention to develop a classification system of COP that the community can widely adopt.

In this paper, we intend to offer a complementary perspective for the formal description of the optimization problems. We consider this activity essential for the study's objectives: to propose a standard and formalism to support the formulation, design, and implementation of COP in a standard way. There, considering this aspiration, the organization provided by this work is synthesized in the following Figure 1. Moreover, a brief description of each dimension is available in the subsequent sections.

2.3.1. Number of Objectives: Single Objective Versus Multi-Objective

From a perspective of the number of objectives, the COP can have two behaviors: (1) single-objective and (2) multi-objective combinatorial optimization problems (MOCOP) [40–45]. In a single-objective problem, the fittest solution is easily determined by the best value returned by the fitness function. In contrast, in multi-objective problems, the optimization process must consider all objectives simultaneously [38]. The performance of a solution in each criterion is not enough, and an analysis that considers all criteria together must support this process. Moreover, the objectives may potentially conflict and concur with each other [19,24,40–42,46,47]. Thus, MOCOPs require additional efforts to assess, compare, and determine which solution performs better than others, balancing multiple criteria.

In practical terms, assessing the solutions' quality is performed by a function that supports the selection process. Essentially, this function receives a set of solutions and the problem's objectives. As a result, it returns a set of "best" solutions that are chosen by a specific assessment approach. Several approaches can be used to perform this evaluation. They can be split into two categories of methods: (1) Pareto and (2) non-Pareto [45].

Remark 1. *These functions used to assess multi-objective problems are generic. Consequently, other optimization problems can reuse these functions because they often are problem-agnostic. However, rarely a particular problem may provide a specific multi-objective assessment. For instance, one method that considers the other fitness values in a weighted formula.*

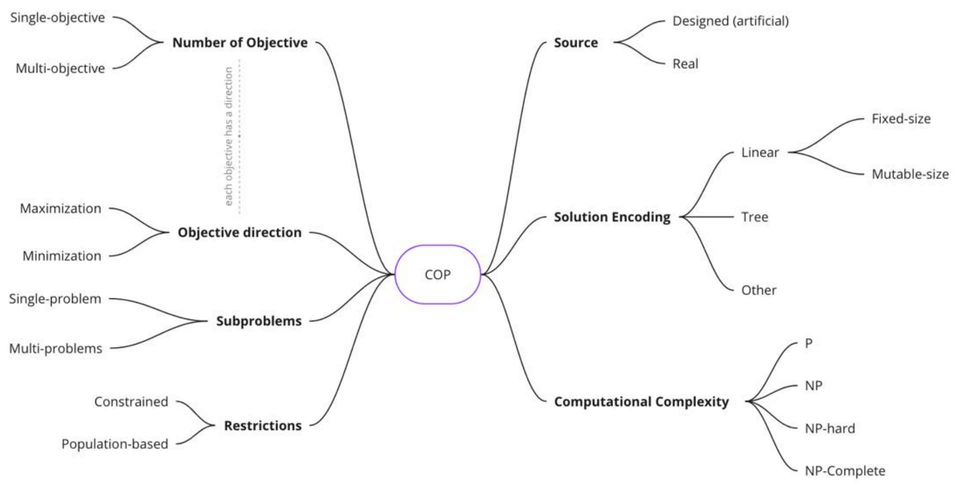


Figure 1. Combinatorial optimization problem—building blocks.

2.3.2. Optimization Objective Direction

Definition 3. *Each objective has an objective function and an objective direction. A fitness function has two main purposes: (1) they determine the quality of the solutions and (2) guide the search by indicating promising solutions [9,40]. The objective direction can be divided into two categories of optimization: (1) minimization or (2) maximization.*

$$Maximization \qquad Minimization \forall\ s\ \in S,\ f(\bar{s})\ \geq f(s) \qquad \forall\ s\ \in S,\ f(\bar{s})\ \leq f(s) \quad (3)$$

In general terms, when the lowest fitness defines the best performance, then it is a (1) minimization problem, and in this particular case, the fitness function also can be called "cost function". On the other hand, when the highest fitness determines the best performance, it is a (2) maximization problem. In this case, the fitness function also can be called "value function" or "utility function".

2.3.3. Subproblems (Single-Problem Versus Multi-Problem)

Some problems, due to their complexity, need to be split into subproblems in order to be adequately resolved ("divide to conquest" strategy). Therefore, the problems, considering the number of subproblems, can be classified into two classes: (1) single optimization problems (single-problems), which are composed of only one problem, and (2) multiple optimization problems (multi-problems) [48–50], which are a non-trivial arrangement of two or more problems. A point that needs to be emphasized is that a problem can be composed of two subproblems (multi-problems), where the first subproblem has one objective (single objective) and the second subproblem has two objectives (multi-objective).

Multi-objective problems require extra efforts to select the optimal solutions (Pareto optimal) and also demand additional work because it is necessary to balance the results of the subproblems to attend to the global perspective of the higher-level problem. The higher-level problem may have global objectives that must also be achieved, and they cannot be entirely perceived by the parts separately.

A typical example is the artificial problem named Traveling Thief Problem (TTP) proposed by Bonyadi et al. (2013) [48]. It is an optimization problem composed of two well-known problems: TSP (Travel Salesman Problem) and KP (Knapsack Problem), organized in a higher-level problem. By way of example, guaranteeing the best loading ships and logistics may not ensure the best profitability. In other words, "the fittest" profitability may not be guaranteed by "the fittest" ship loading solution (equivalent to KP part) and "the fittest" ship scheduling (equivalent to TSP part).

2.3.4. Unconstrained Versus Constrained

In general terms, constraints are a set of prerequisites—that sometimes are necessary—to determine if a solution is valid, according to the criteria of a specific problem. Consequently, an optimization problem can be (1) constrained or (2) unconstrained [6]. Therefore, in an unconstrained problem, all generated solutions are feasible. Oppositely, a constrained problem is subjected to some criteria to be viable.

The constraints typically are divided into two types, (1) "hard" or (2) "soft". The "hard constraints" are mandatory prerequisites. They must be satisfied, and if a constrain violation happens, the algorithm must reject the solution. "Soft constraints" are more flexible; they are not mandatory, but desirable. If a constrain violation occurs, the algorithm should not discard the solution. In this case, it could be addressed in the fitness function, making the performance worse and the solution less suitable. Due to this possibility, the "soft constraints" are sometimes called cost components.

2.3.5. Artificial Versus Real-World Problem

Optimization problems, considering their origin, can be (1) "artificially" created by someone or (2) identified in the real world, requiring efforts to understand and specify them. [16,49,50]. Artificial or designed optimization problems are proposed by someone, generally as an abstraction and reduced view of reality. In this situation, the author defines a synthetic and controlled environment. All variables and criteria are mapped and previously specified.

By way of illustration, the required decision variables are explicitly defined to the specific objective of the artificial problem. Consequently, the definition of the solution encoding is facilitated, and they embrace all dimensions required. The objective functions are precise because they were designed to determine which is the best solution. Since the problem is proposed, there is no doubt about the concepts and criteria behind its definition. A good example of designed problems is the Travel Salesman Problem (TSP) proposed by Hamilton and Kirkman.

In contrast, real-world optimization problems are not artificially proposed by anyone. They are challenging and contradictory situations that arise on a daily basis. Consequently, first, they need to be identified, and after being identified, they also must demand a solution. Since a solution is required, a set of efforts can start to understand and design them. However, the reality is more challenging to be captured. Generally, they must be investigated and consider many dimensions of the subject (systemic view).

Moreover, there is no direct feedback to inform if the problem is defined correctly or not. Therefore, their representation will demand a certain level of knowledge about the problem. In other words, the people who will formalize and develop the problem must know its context and peculiarities.

2.3.6. Solution Encoding

In combinatorial optimization problems, any solution is an arrangement of elements, combinations, or permutations. They follow a type of representation (or shape) and encoding rules that arrange the values of the search space's dimensions (variables) to form a solution. The most common types of representation are (1) a linear sequence of elements (as an array) and a (2) tree that organizes the elements recursively. However, any other data structure can be used as a type of encoding. It will depend on the necessity of the problem to represent a solution. Arrays and trees are the most frequent because they are easy to manipulate and transform, but, by way of example, (3) a graph representation that links nodes (a linked list) can also be found. Other formats of representation require heuristics that can handle their data structure and transform them properly.

Each type of representation may be subjected to some encoding rules that regulate how they are created and how they can be manipulated. Solutions of linear shapes are based on set values (symbols). They must consider the size (fixed versus mutable), the elements' type of arrangement, and if the elements can be repeated or if their order is important. Solutions of three shapes generally follow a language composed of functions, terminals, and constants.

2.3.7. Problem Data Environment

As mentioned before, the objective functions (OF) are responsible for determining the quality of the solutions. By standard, the OF interacts directly with the decision variables (data) to determine the solution's quality. However, OF also can interact with a simulated environment. In this scenario, the simulated environment receives a solution and returns feedback which is interpreted by an OF that concludes the solution's quality [10]. It is analogous to the functioning of reinforcement learning (RL), where agents interact with an environment and receive feedback (state and rewards).

The usage of simulation in optimization is an essential tool to deal with particular scenarios of problems [10,11,27,51] where a simulation is required. Consequently, the COP can be divided into three classes: (1) decision variable-based, (2) simulation environment-based, and (3) both. The challenge of simulation usage in COP is the design and implementation of the simulated environment properly for each type of problem.

2.3.8. Problem's Computational Complexity

The search space, which is the set of admissible solutions for an optimization problem, is usually exponential with respect to the size of the input. This happens for the vast majority of the real-world combinatorial problems [35]. In particular, the size of the input is determined according to the specific problem. For instance, solving an instance of the minimum spanning tree problem requires in input a weighted graph, and the input size corresponds to the number of nodes and edges of such a graph. This example suggests that different instances of the same optimization problem are characterized by a different size of the input. As a consequence, it is not possible, for non-trivial problems, to define an algorithm that provides an optimal solution for all the possible problem instances within a time constraint that is polynomial in the input size [52]. For this reason, it is fundamental to take into account the time needed by an algorithm to provide the final solution. The time also depends on the hardware available, and the same algorithm executed on the same instance, but different computers may provide significantly different running times. For this reason, to perform a fair comparison, it is common to consider a different concept that is the computational complexity of an algorithm. The computational complexity allows calculating the time an algorithm needs to return the final answer or solution as a function of the input size. Thus, computational complexity is the commonly used method for comparing the running time of two different algorithms because it is independent of the hardware on which the algorithms are executed. Computational complexity considers the number of basic operations to be performed during the execution of the algorithm and uses this information to provide an indication of the running time as a function of the input size. For instance, an algorithm that must visit all the vertices V of a graph given in input has a complexity of O(V). In other words, its complexity is linear with respect to the input size (i.e., the number of nodes of the graph). To distinguish between easy and difficult decision problems, one should consider the class of problems that are solvable by a deterministic Turing machine in polynomial time and problems that are solvable by a nondeterministic Turing machine in polynomial time. Without going into formal detail related to the Turing machines, it is possible to provide an intuitive definition of the two classes of problems, which are known as (1) P and (2) NP [35]. In particular, a decision problem is in P if and only if it can be solved by an algorithm in polynomial time. On the other hand, a problem is called NP (nondeterministic polynomial) if its solution can be guessed and verified in polynomial time; nondeterministic means that no particular rule is followed to make the guess. Thus, the algorithm used to solve a problem in NP consists of two steps. The first consists of a guess (generated in a nondeterministic way) about the solution, while the second step consists of a deterministic algorithm that verifies if the guess is a solution to the problem [48,53]. Finally, if a problem is NP and all other NP problems are polynomial-time reducible to it, the problem is NP-complete. In other terms, determining an efficient algorithm for any NP-complete problem implies that an efficient algorithm can be found for all the NP problems.

This distinction is important in the context of metaheuristics: problems that are in NP are commonly addressed using metaheuristics. In this case, the algorithms run in polynomial time and provide a solution that approximates the optimal one.

2.4. Heuristics

Before proceeding further to a more in-depth description of metaheuristics, it is necessary to briefly present what a heuristic is. According to Judea Pearl (Pearl, 1984), *heuristics are criteria, methods, or principles for deciding which among several alternative courses of action promises to be most efficient in order to achieve some goal. They represent compromises between two requirements: (1) the need to make such criteria simple, and at the same time, (2) the desire to see them discriminate correctly between good and bad choices*. In general terms, heuristics in computer science may be summed up as the usage of useful principles and search methods designed to resolve problems in an exploratory way, especially when it is not feasible to find an exact solution due to its complexity in a feasible amount of time. Therefore, it tends to involve a certain degree of uncertainty and the result is often not deterministic.

The heuristics, considering how they create solutions, is often divided into three categories: (1) constructive and (2) improvement, and (3) hybrid heuristics [54]. The constructive approach generates a solution element by element by analyzing the decision variables. For instance, they take the best next piece without considering the entire solution until they form a complete solution. In the improvement approach, the method generally starts with a random initial solution that is improved iteratively, altering some parts to search for a better solution. Hybrid heuristics, as the name suggests, uses both approaches, aiming to take advantage of their methods.

2.5. Metaheuristics

Glover introduced the term "metaheuristics" (Glover, 1986) by combining two words, meta (Greek prefix *metá*, "beyond" in the sense of "higher-level") and heuristic (Greek *heuriskein* or *euriskein*, "find" or "discover" or "search"). Metaheuristics are habitually described as (1) problem-independent algorithms that can be (2) adapted to incorporate problem-specific knowledge and definitions [4,27,33]. Consequently, they can resolve several categories of optimization problems efficiently in a flexible way because they do not have "strong" links with any specific problem.

A metaheuristic (MH) can be seen as a *higher-level heuristic*, composed of a master heuristic (*metaheuristic cycle*) that drives and coordinates a set of subordinated heuristics (SH)—in an iterative cycle—to explore the search space [2,4,6,14,55–57]. This relationship between master heuristics and subordinated heuristics (SH) is illustrated in Figure 2. Each subordinate heuristic performs a different role in the metaheuristic's search strategy. Thereby, the MH cycle arranges the SH to work together—in a higher level of abstraction—to search for an optimal solution for a given problem in a coordinated way.

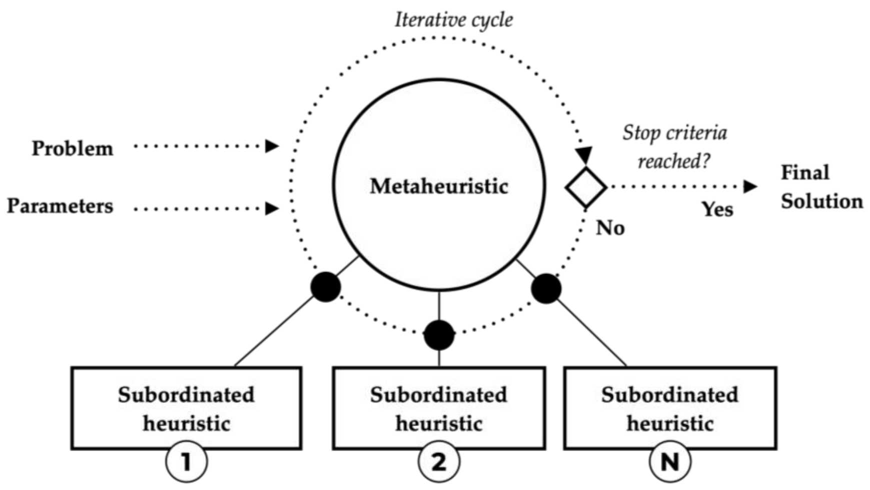


Figure 2. Metaheuristics and subordinated heuristics relationship.

Informally, it is possible to say that the metaheuristics perform "blind exploration" in a vast search space that they do not know. Consequently, they cannot "see" the whole landscape to identify "hills" and "valleys" (optimal solutions). For that reason, the strategies used by the heuristics (master and subordinated) to navigate in the search space are crucial to investigate and map promising spots of the search space. Thus, everything that a metaheuristic knows about the landscape is the "visited places" (solutions) and the quality of these solutions (fitness). Therefore, after spending considerable time exploring the search space, the algorithm will have some information about parts of the search space. As a result, the best solution found is the return of the algorithm.

A metaheuristic will be successful in discovering optimal solutions on a given optimization problem if it can conduct a good balance between exploration and exploitation [2,16,34,58,59]. *Exploration* and *exploitation* are related concepts in some sense because both techniques are employed to navigate and examine the search space. However, there is a subtle difference between them. *Exploration* can be seen as diversification because it aims to identify promising areas in the search space with high-quality solutions. In contrast, *exploitation* can be understood as intensification because it intensifies the search in a good region (solutions with excellent quality) of the search space to find a better solution [2,4,34,44,58].

2.6. Metaheuristic System of Classifications

The classification of metaheuristics is a fuzzy and debatable topic. There are various ways and distinct dimensions to classify the metaheuristics [12,15,19,53,57,60–62]. Some researchers generally choose the most appropriate classification criteria according to their point of view or work [12,19,62]. This issue will be better explained in Section 3.1, Lack of Guidelines, to organize the field.

It is essential to highlight that this article does not offer exhaustive analysis and does not intend to propose a classification system for metaheuristics. Even so, after conducting an extensive review, we observed some frequent classifications and distinctive perspectives of how to classify metaheuristics. For this reason, this study attempted to unify and accommodate all of them in a single view. As a result, we believe that this organization can be used as an input that may contribute to a future study focused on creating a classification system for metaheuristics. Additionally, the real idea behind this analysis is to understand all possible perspectives of metaheuristics and use this information to support the creation of a formalism to design and develop metaheuristics.

Moreover, after seeing a significant number of different classification methods, we believe that the most appropriate way to classify metaheuristics is to simplify the organization. It should be through the usage of several complementary orthogonal dimensions—in other words, a system analogous to a tag classification, where the dimensions are independent and do not have links. We also understand that hierarchical relationships among different dimensions should be avoided. The idea behind this is to avoid the potential problems of incompatibility when multiple dimensions are arranged together.

Figure 3 synthesizes our work to unify and organize the most important and distinctive dimensions found in this revision. The next sections will provide a brief explanation of each one of them.

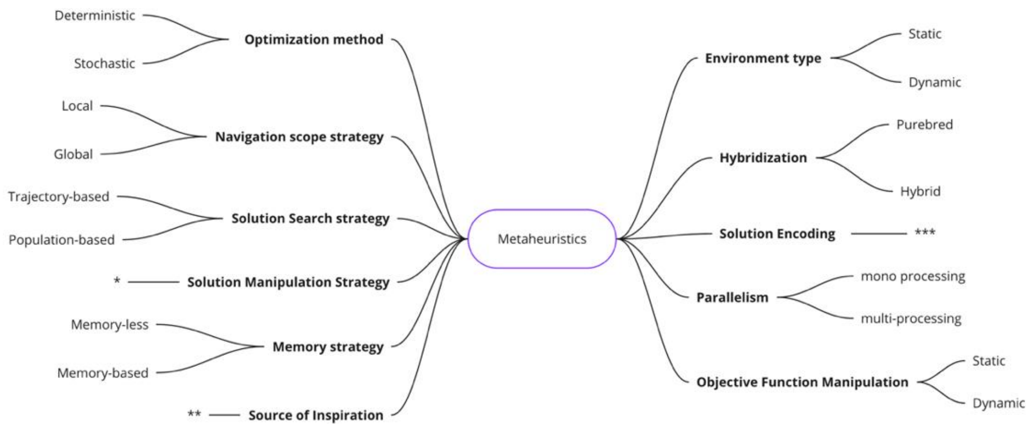


Figure 3. The most common dimensions of metaheuristics classifications found in the literature. * The Solution Manipulation Strategy requires a deeper explanation of how the metaheuristics can be subdivided. ** The source of inspiration has several different views in the literature. *** The same division found in the COP system of classification.

2.6.1. Optimization Method

This dimension organizes the metaheuristic according to its optimization method. They are divided into (1) deterministic and (2) stochastic [23,24,63]. An explanation of these methods was already provided in Section 2.1, Optimization.

2.6.2. Navigation Scope Strategy

This dimension aims to point out the differences in a fundamental aspect of metaheuristics' navigation system by analyzing its *meta-strategy* to cover the search space. Essentially, the methods in this perspective can be distributed into two groups of navigation scope: (1) local and (2) global searchers [6,34,64].

Remark 2. *The term "navigation scope strategy" is defined in this study because it was considered the most suitable term to organize the definition for (1) local and (2) global searchers.*

The (1) local searchers are specialized in seeking the nearest optimum solution until achieving the peak of the region (e.g., Hill Climbing). On the other hand, the (2) global searchers attempt to examine other areas of the search space, even if it involves moving to worse solutions during the search (e.g., Genetic Algorithms).

2.6.3. Solution Search Strategy

The solution search strategy dimension also reveals another vital aspect of the metaheuristics navigation system. It is also related to the way a metaheuristic covers the search space. However, in this case, it is associated with the number of solutions that are optimized in each iteration. Regarding this characteristic, the methods can be split into (1) trajectory-based (single solution) or (2) population-based (set of solutions) [2,16,34,65–67].

Remark 3. *The term "solution search strategy" is defined in this study because it was pondered as the most appropriate term to arrange the definition for (1) trajectory-based (single solution) or (2) population-based.*

The trajectory-based methods work by improving a single solution iteratively, moving from one solution to another, forming a trajectory (track). In contrast, population-based metaheuristics operate by improving a set of solutions (population) simultaneously. In other words, some operations are applied in each iteration, and the current population is replaced by a new one (with some improved solutions). A metaheuristic population-based can be further divided into single-population (Section 2.1) or multi-population (Section 2.2). Multi-population is generally used to increase the diversity of the solutions [68] and establish a parallel search through subpopulations that evolve together. Moreover, a multi-population search may require additional mechanisms and methods to manipulate individuals among populations.

2.6.4. Solution Creation Strategy

This dimension also reveals another crucial aspect of the metaheuristics navigation system. In this case, it defines the strategy used to generate and transform solutions during the search process. It also can be considered one of the most important behaviors of a metaheuristic. The solution creation and manipulation strategy can be divided into two main classes of algorithm behavior: (1) combination and (2) movement [12].

The (1) combination strategy, as the name suggests, is the group of algorithms that are based on operations that are applied to mix the content of some solutions to form other solutions. Additionally, they did not elect a reference solution to be followed. Therefore, any solution of a set of solutions can be used to form the new ones. The Genetic Algorithm (GA) is the most known and symbolic algorithm of this group. They can still be subdivided into two groups, mixture (Section 1.1) and stigmergy (Section 1.2).

The methods based on "mixture" perform a *direct combination* of existing solutions to form a new one. The Genetic Algorithm (GA) is the most famous metaheuristic that uses the combination strategy. The algorithms based on stigmergy, on another side, use an ***indirect combination*** strategy, by using an intermediary structure got from the various solutions used to produce new solutions. Ant Colony Optimization (ACO) is the most known algorithm in this category.

On the other hand, the algorithms that employ (2) movement strategy use a pattern found in the current set of solutions—by using a differential vector that calculates the intensity and direction of the movement—to generate a new solution. In addition, these algorithms define a solution used as the reference and it is followed by others. The newly developed solutions can replace the reference solution or compete with other followers and replace them. The Particle Swarm Optimization is the most emblematic algorithm of this category.

The methods based on movement can be further split into three subdivisions, all population (Section 2.1), representative (Section 2.2), and group (Section 2.3). As the name suggests, all population members define the movement in all population (Section 2.1). However, each member has a different level of influence based on the distance of the reference solution. In representative (Section 2.2), each solution is influenced by a reduced representative solution group (PSO is classified in this subdivision). The group-based algorithms (Section 2.3) just consider a subset of solutions to form the new solution. These groups do not use representative criteria. They use neighborhood or subpopulations strategy to define the groups to support the calculation and generate new solutions.

Remark 4. *It is noteworthy to emphasize, to organize this dimension, we used as the main reference the inputs provided by Molina et al., 2020 [12]. They offered an outstanding critical analysis and recommendations about metaheuristics classification. Essentially, we used Chapter 4 (Taxonomy as Behavior), where they analyzed several metaheuristics and organized them according to the mechanism employed to build and modify solutions. It should be noted that this study adapted the nomenclature to accommodate these inputs with other perspectives. For instance, "Solution Creation" was renamed to "Combination" (1), and "Differential Vector movement" was renamed to*

"Movement" (2). Moreover, the term "Solution Creation and Manipulation Strategy" is created to arrange this dimension because the metaheuristics can have many other types of behaviors.

2.6.5. Source of Inspiration

In the past few decades, the creation of metaheuristics algorithms was intense [6,12,15,69–72] due to its success in solving hard problems. These algorithms usually obtain their inspiration from some characteristics found in nature [13,70,73–79]. Nevertheless, some researchers have criticized the excess of new metaheuristics more focused on its metaphor than their quality as optimization methods [12,15,80]. This issue will be discussed in-depth in Section 3, which covers research challenges and gaps.

Furthermore, the classification of algorithms from a source of inspiration is a fuzzy topic in the literature [6,12,15,19,20]. It is common to find divergent points of view to classify them according to the inspiration applied. This confusion is generally related to the absence of ground rules or due to partial views that do not consider a more extensive classification system [12].

Moreover, some "subfields" are more mature or successful than others. Consequently, it is easier to perceive some families of algorithms than others (e.g., Evolutionary Algorithms and Swarm Intelligence). Despite this, even for these well-known families, it is common to see distinct ways to organize them, especially in a more extensive system that considers other types of inspirations. In other words, a family can be found at different levels of the hierarchy in literature. For instance, some classifications organize swarm intelligence inside the bio-inspired, and others consider swarm intelligence at the same bio-inspired level.

We believe, as a ground-rule, that science already has an organization to classify the areas of study (sciences). Metaheuristics did not need to reinvent the wheel. Since they are based on science areas, the taxonomy of inspiration sources should also follow a similar structural organization but considering a more straightforward way with fewer hierarchy levels. Based on this assumption, this study arranged the most found source of inspiration presented in the literature. Again, it is not an exhaustive list but rather a unified view of the most relevant sources of inspiration, and it is illustrated in Figure 4.

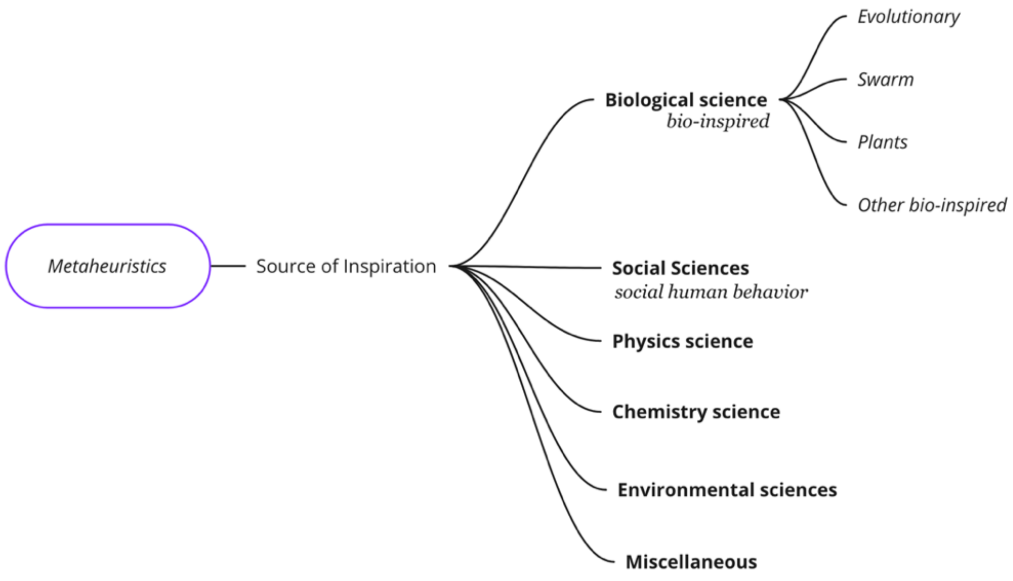


Figure 4. Families of metaheuristics by source inspiration or metaphor used.

Remark 5. *This paper does not have the objective to go deeper into this subject. Many studies have already stressed this analysis of the source of inspiration. For further information about this, please* *consult the list of references. This section contains a list of recommended readings. You can find a section with excellent reviews about the "source of inspiration" and its implications.*

2.6.6. Memory Strategy

This is a dimension that can classify metaheuristics according to the presence of the memory. Basically, the metaheuristics can be classified as (1) memory-less and (2) memory-based [36]. Memory-less are metaheuristics that do not have any way to store information during the search process and only consider the current state of solution(s). An example of memory-less is Simulated Annealing (SA). In contrast, memory-based metaheuristics use a mechanism to accumulate information during the iterative process to guide the search. An example of memory-based metaheuristics is Tabu Search (TS).

2.6.7. Purebred Versus Hybrid

A metaheuristic can be (1) purebred or (2) hybrid [34,81–85]. A purebred is focused on its methods only. Meanwhile, a hybrid metaheuristic generally has a method as a basis and uses pieces of other metaheuristics or employs any other auxiliary methods [15] to take advantage of other approaches.

2.6.8. Data Environment

A metaheuristic can interact directly with (1) the search variables or a (2) simulated environment. The metaheuristics that take advantage of simulation are also known as simheuristics. They extend the standard metaheuristics by combining metaheuristics (optimization) and simulation [11,51,86].

In fact, the interaction with decision variables or a simulated environment is performed by the Objective Function. Therefore, this dimension is linked to the behavior of the Objective Function (covered in Section 2.3.7, Problem Data Environment). Consequently, as well as the decision variables, the simulated environment must be provided by the problem, since they are problem-related. Thus, it must be transparent for the metaheuristics. Nevertheless, simheuristics may require some framework components to support the usage of simulated environment and other components (e.g., fuzzy component [51]).

2.6.9. Solution's Encoding

Each metaheuristic method is prepared to deal with a specific type of solution encoding. Consequently, metaheuristics can only solve problems with an encoding that they can manipulate. Likewise, a particular solution's encoding can only be resolved by a restricted number of metaheuristics that know how to handle its encoding. Thus, it is possible to classify metaheuristics according to the encoding that they can manage. The differences in the solution encoding are covered in Section 2.3.6, Solution Encoding.

By way of illustration, Genetic Algorithms (GA) and their variations of subordinated heuristics are designed to manipulate a linear sequence of characters (an array of elements). In comparison, Genetic Programming (GP) is prepared to deal with elements organized recursively in a tree.

2.6.10. Parallel Processing

The capacity to perform parallel processing is another characteristic of the metaheuristics, and it can be seen as an improvement to explore the search space more efficiently. The metaheuristics can be classified into (1) mono processing and (2) multiple processing. The mono processing is not prepared to perform parallel processing. On the other hand, the multi-processing has an architecture prepared to perform parallel processing and consolidate these multiple searches [2,6,16]. It is an option to improve any metaheuristic because they can incorporate and activate this capability when needed. In the literature, there are several strategies of parallelization to reduce the time of the search and explore more areas of the search space simultaneously.

2.6.11. Capacity to Alter the Fitness Function

Considering the capacity to manipulate the fitness function, the MH can be subdivided into two categories, (1) static and (2) dynamic [9,11,82]. The static category is the standard way, where the MHs use the objective function as the problem designed it. In contrast, some algorithms may change the OF behavior by incorporating some obtained knowledge during the search process. The main idea behind this is trying to escape local optima [87]. Guided Local Search (GLS) is an example of MH with this capacity.

2.7. Metaheuristics Require Experiments to Evaluate Algorithms' Performance

The performance evaluation is an imperative criterion for metaheuristic. Experiments support the determining of how well a metaheuristic (MH) investigates the search space. Additionally, considering the consequences of the "No Free Lunch" theorems, it is hard to know in advance which MH will perform better than others. Moreover, since the most successful MH are stochastic—due to their probabilistic nature (uncertainty)—the results of MH tend to vary. Therefore, by considering an instance of a problem, the same algorithm may have variances in its performance. Consequently, to compare a set of metaheuristics' performances and determine which is the best, it is necessary to design and execute an experiment to conduct an appropriate evaluation.

An experiment for metaheuristics evaluation involves a set of coordinated steps in a controlled environment, and an experiment should guarantee its reproducibility and comparability. In a summarized way, an experiment considers the following steps: (1) objectives definition, (2) design, (3) execution, and (4) conclusions [5,58,87–89].

The first step, (1) objectives definition, aims to define the experiments' drivers, goals, and research questions. The design step (2) intends to specify, plan, and prepare the experiment. It commonly considers the following activities: defining the measures and terminology, selecting the metaheuristics, defining the parametrization tuning strategy, determining the report format, etc. Each algorithm may have several possible parameters configuration, and the parametrization tuning strategy defines how these configurations will be tested. It is essential to highlight that parameter setting directly influences the algorithm's performance [90] and it varies by the given problem. Generally speaking, it can happen manually by defining a set of possible configurations, or it can be conducted automatically, supported by an optimization method. Furthermore, when it is not a real problem, it should be necessary to choose benchmark problems and their "data" to compare performance among designed problems.

The step experiment execution (3) is dedicated to performing the experiment, running each configuration, and collecting its data. It is necessary to emphasize that each configuration must be executed enough times to conclude whether a specific configuration is effectively better than others (e.g., thirty times may be sufficient to make conclusions statistically). Consequently, the data analysis must consider a statistical approach to understand the variance to determine whether the best performances are statistically. Consequently, it regards dispersion statistics measures (e.g., mean, median, standard deviation, and outliers' analysis) organized in visualizations or reports to support the decision. The final step, (4) conclusions, aims to analyze the results and organize the conclusions of the experiment.

3. Challenges, Research Gaps, and Issues

This section aims to synthesize the main challenges, research gaps, and issues found in this research. Several papers provided valuable inputs for this topic in their conclusions. At the same time, we also decided to consider our experience and difficulties in developing COP and metaheuristics. While it is not an exhaustive review, by covering these points, this section offers a satisfactory answer for RQ-2: *What are the main challenges of the COP and metaheuristics research field?* This section is organized by arranging the main findings into groups according to the subject. A brief description of each one of them is available in the following subsections.

3.1. Lack of Guidelines to Organize the Field

As mentioned in Section 2.6, the classification of metaheuristics is a fuzzy and debatable topic [3,12,19,57,60–62]. Because of this, it is common to find several different ways to classify metaheuristics in the literature [91]. Additionally, researchers tend to choose the most appropriate classification criteria according to their point of view [12,19] or their subfield. Moreover, some researchers have mixed up some concepts and definitions, and it is common to see contradictory classifications in the literature.

These divergent perspectives are one of the challenges for the organization and progress of the metaheuristics field. Many researchers point that the root cause of this phenomenon lies in the absence of a consistent definition and guidelines to organize the field [3,12,15,19,57]. Considering all mentioned issues—and many others not covered in this study—we emphasize the necessity for an in-depth review to develop guidelines to classify and organize the metaheuristics.

Nevertheless, we also understand that the definition of a classification system requires consensus and community adoption. Consequently, we argue that it is an open question that requires a study with this purpose. Therefore, we believe it is a promising line of research for future work.

Furthermore, a system of classification could be supported by a database to classify metaheuristics and organize their metadata. Consequently, this database could consider all the orthogonal dimensions to register metaheuristics. It also could be extended to an open-source pool of metaheuristics supported by a framework.

3.2. Focus in Developing New Algorithms Versus "No Focus in Consolidation"

Due to the large number of algorithms, it gradually became clear that part of these algorithms does not necessarily lead to suitable optimization methods [15]. Thereby, some researchers have started to criticize the excess of new metaheuristics algorithm publications [12,13,15,19,57]. The main argument lies in the overvaluation of the source of inspiration. Some researchers tend to be more concerned with the algorithm's metaphor and novelty than with their quality as optimization methods.

Sörensen [15,57] called this phenomenon of "metaphor fallacy" he expects that this mindset turns quickly. He argued that *most "novel" metaheuristics based on a new metaphor take the field of metaheuristics a step backward rather than forward.* Therefore, the community must be more concerned about developing more efficient methods and avoid the "metaphor fallacy" trap.

Many researchers also argue that the metaheuristics' behaviors are more important than the source of inspiration [12,19,92]. Generally, the metaphor is more ludic and generally does not explain the real functioning of the algorithm. Because of this, the metaphor used may be difficult to understand the algorithm's proposal because it can overshadow the actual functioning.

Furthermore, these new algorithms (the novelty) are often an extension of existing algorithms with little functional alterations [16,91,92]; they can be seen as variants, and the metaphors are highlighted as the novelty. Molina et al., 2020 [12], also analyzed the most influential algorithms in their review, and they state that five algorithms (their standard implementations) influenced the majority of other algorithms. According to them, the algorithms most influential are (1) Genetic Algorithms (GA), (2) Particle Swarm Optimization (PSO), (3) Ant Colony Optimization (ACO), (4) Differential Evolution (DE), and (5) Artificial Bee Colony (ABC).

This issue must highlight two aspects. Firstly, this research production trend may be related to the existing publication mindset, where the chances to publish a new algorithm are high. Weinand et al., 2021 [91], pointed out that the intensity of Combinatorial Optimization grew exponentially since 1989. Secondly, obviously, the new algorithms were crucial to the development of the metaheuristics field. All the proposed metaphors helped to bring different ways to explore the search space, such as anarchy societies, ants, bees, black holes, cats, cuckoos, consultants, clouds, dolphins, fishes, flower pollination, gravity, water drops, and so on so forth. The critic is related to the overestimation of the source of inspiration as a primary driver with few concerns whether it effectively presents novelty for optimization methods (innovation in search strategy perspective).

This issue is an open question in the past few years, at least since 2013 when Sörensen published "Metaheuristic—the metaphor exposed" [57] and the attention for this has increased. It is a research mindset that requires change because, in some sense, the progress and consolidation of the metaheuristics field are difficult. The answer for this challenge may demand a methodology to define COP and MH and provide a fair way to compare metaheuristics features, strategies, and their performance.

3.3. *Experiments and Performance Evaluation Issues*

Conventionally, the field of optimization problems has focused on metaheuristic performance, where an algorithm is considered good if it has a good performance compared to some benchmark [15,47]. Additionally, the studies that proved superiority through a comparison have higher chances of being published (also known as the "up-the-wall game").

Nevertheless, in recent years, some researchers have perceived issues in some studies that focus on evaluating the performance of new metaheuristics [6,18,92,93]. The main cause lies in the fact that some experiments may induce inaccurate analysis because they do not promote fair performance comparisons among metaheuristics. This may lead to biased analysis. For example, the set of problems used as benchmarking could be selected because the new algorithm performs well in these cases. Another example is that standard versions of successful algorithms, such as genetic algorithms, are used in the comparison, but the field has evolved since its creation, and there are many variations of GA that added new features that improved the canonical version of the algorithm.

Another aspect of the performance issues is related to the fact that experiments generally only use dispersion statistics measures (e.g., mean, median, standard deviation, and outliers' analysis) or use basic statistical tests. Some researchers such as Boussaïd and Hussain [2,6] have pointed out the necessity to add other statistical analyses to promote more sophisticated comparisons and evaluate other aspects of metaheuristic performance.

3.4. *Absence of Formalism to Design and Develop COP and Metaheuristics*

In the literature review, no evidence was found that a standardization or formalism to design and implement COP and metaheuristics exists. There are some clues or definitions of how to implement them, but there is no standard that could be used as a reference and followed to build COP, metaheuristics, or subordinated heuristics in an integrated way. This absence of standardization leads to different ways to represent COPs and makes difficult its reuse. Generally, researchers understand the basic functioning of the algorithm and implement their own version and adds the problem definition.

Although metaheuristics are problem-agnostic, they often have some level of attachment to the problem in practice (code). Additionally, they are also attached to subordinated heuristics, and vice versa. In other words, in most implementations, it is not easy to separate the code of the subordinated heuristics, metaheuristics, and problems. Therefore, in practical terms, several efforts are necessary to prepare the algorithms for the needs of a specific problem before starting any search for a solution.

Consequently, this absence of standardization leads researchers to start "almost" from scratch by creating their own version of the algorithm and adjusting the metaheuristics to deal with the specific problem or developing the required subordinated heuristics (related to the solutions encoding manipulation). After this, several tests are necessary to check if they return the expected result and avoid errors and bugs. Moreover, these tasks are time-consuming because they demand preparation, which reduces the time spent on the analysis and tuning the results [4,6,15,20].

To complete the view of this issue, it requires an analogy with other more mature methods of machine learning (e.g., neural network, clustering, decision trees, etc.) For instance, a decision tree is an algorithm that resolves classification tasks; they classify data in classes. In informal terms, they receive as input (1) data, (2) a set of features to analyze, and (3) parameter settings, and as a result, they return (4) a solution (classification model). Evaluation functions evaluate the returned solution and determine its quality as a classifier. All these steps occur without any change in the structure of the algorithm. In other words, it does not demand any code changes, and only new data were provided (analogous to problem instance).

Therefore, the idea is not to eliminate the flexibility that enables the metaheuristic and COP extensibility. The goal is to define a standard behavior to eliminate the dependency and links of metaheuristics and COP building blocks and increase the reuse and reducing or eliminating the necessity of code changes. While you just need to provide a problem instance and analyze the solutions, keeping the possibility to add new components every time it is necessary. Consequently, a standard to design and develop COP and MH is vital to avoid starting from scratch and it is the basis for the next issue.

3.5. Support Structure to Not Start from Scratch

This issue highlights the necessity of a software framework to solve COP by using metaheuristics without building algorithms from scratch [9,20] and reusing a set of built-in components and functions [4,15,20]. Because of this, a software framework must also provide facilities that support its functioning and help its users achieve their objectives.

Generally speaking, the foundational premise to support a software framework is defined by the common behavior of its components and their API (Application Interface Programming). These elements define how the components can communicate with each other and keep their independence and generalization aspects to potentialize their reuse.

Considering a COP and MH framework, the definition of two levels of standards is required. The first level is related to the core components, which define the standard to design and implement COP and MH (a necessity explained in the previous section). The second level is concerned with defining the standard of the auxiliary components that supports MH in the search activities and other required supplemental tasks. The challenge is to identify these auxiliary components and implement them as stand-alone software structures which can be reused by any other components. The other challenge is to keep the framework capacity for extensibility of existing algorithms or add new ones.

Again, to complete the view of this issue, an analogy also needs to be provided. In this case, a comparison with a more mature machine learning framework is necessary. For instance, Scikit Learn provides several machine learning algorithms and many auxiliary functions required to build and evaluate the machine learning models. Additionally, it does not require any changes in the structure of the algorithms. Only the data and the algorithm configuration must be provided. Furthermore, Scikit Learn is open source and is widely used by academics and industry. Therefore, we argue that the same maturity level should be required of a metaheuristic framework.

Therefore, this issue aims to emphasize the necessity of a COP and MH Methodological and Software Framework to consolidate and promote the progress of this field. A framework with these characteristics must be an open-source project supported by the community. Moreover, it also should promote more cooperation of different metaheuristics approaches than only in competition or hybridization. It also opens space for metaheuristics ensembles, for instance.

4. Results: Standardization, Methodology, and Framework for COP and Metaheuristics

This section, supported by the answers of *RQ-1* and *RQ-2*, aims to present a proposal for combinatorial optimization problems (COP) and metaheuristics standards. It covers a formalism to design and implement COP and metaheuristics, a methodology to design problems, and a conceptual framework. Moreover, this section will address the answers for *RQ-3*, *RQ-4*, and *RQ-5*.

4.1. Framework Structure

This section aims to provide an answer for *RQ-3: What are the features of COP and metaheuristics that could be developed as independent methods in a framework structure?* It leads us to investigate and identify which are the vital elements that can make part of a software framework and support COP and MH experiments. By identifying these elements, the objective is to avoid rework and spend time developing components that should already be available; in other words, "not reinvent the wheel" all the time. Additionally, it also reduces the possibility of bugs and errors because these general components were already tested and checked before. Therefore, the idea is to take advantage of these components by reusing them (without code changes) and focusing on the problem definition or developing a new metaheuristic (or subordinated heuristics), increasing productivity and efficiency.

The following sections provide a brief description of these essential components arranged by this study. In Figure 5 is the high-level view of the proposed architecture with these important support components.

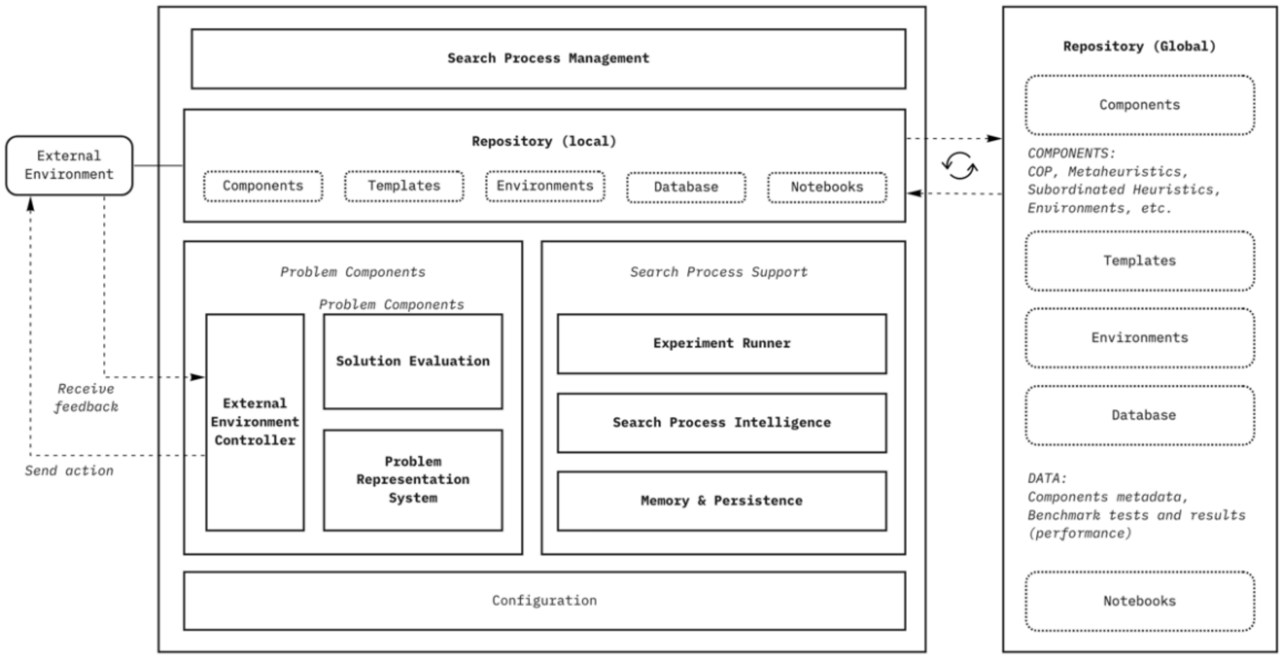


Figure 5. Proposed framework architecture—high-level view of the main components.

Nevertheless, before proceeding to the explanation of the components, we would like to highlight the main higher-level features expected of a framework for COP and MH, and they are the following:

1. Extensibility of software components;
2. Flexibility to exchange behavior of the components;
3. Componentization driven (component-based development); and
4. Cooperation among algorithms.

By considering the infinitude of problems and metaheuristics algorithms, a framework for COP and MH needs to be conceived with the ability of extensibility. In other words, it must be able to add "pieces of software" (POS) without affecting the structure or core code of the framework. We are talking specifically about the possibility of adding problem templates—since they are also "programs"—and metaheuristics or subordinated heuristics methods, extending the framework. Consequently, it would ultimately increase the framework's capabilities in solving combinatorial problems. It requires a core level of components and a software standard definition that should be followed by the new pieces of software, and supports the extensibility capacity.

Extensibility, accompanied by the flexibility to change components' behavior, is essential to make algorithms more effective in searching. An algorithm behavior can be modified by configuration or by exchanging its software pieces (subcomponents). Additionally, the ideal behavior change should be possible both in the design time and in the run-time without code changes in the basic structure. It also should not consider hierarchical inheritance where a class inherits from a master class because it is hard to maintain. Moreover, a few changes in the base class can easily break the other subclasses that reuse the master class. It should consider a software design pattern that avoids this condition and prioritize the change in subcomponents to incorporate or modify behaviors.

Moreover, this flexibility capacity must consider at least two levels, (1) operational and (2) strategical. The operation must provide the functionality of exchange behaviors. In other words, make possible the concrete exchanges of parameters or subcomponents no matter the moment. It also requires a well-designed API and a software design pattern and standards. The strategical level must follow the search, recognize situations (e.g., when the MH is trapped in a local optimum), recommend changes when needed, and comprehend the results and feedback of its recommendations.

A componentization-driven framework comes with superpowers because it enables the possibility of writing and deploying new software pieces independently. They take advantage of framework existing services, and they are developed only focused on their objectives. Therefore, these new "pieces of software" (e.g., COP templates and instances, metaheuristics, or subordinated heuristics) can run above the existing framework environment. To increase their reusability potential, a repository is required, where the authors could share and keep these "pieces of software" and the community can easily use them. The cooperation among metaheuristics is also a crucial high-level feature. A well-designed API and a software design pattern can provide an environment where metaheuristics could cooperate to resolve a particular problem simultaneously or asynchronously.

4.1.1. Repository (Local and Global)

The repository is designed to maintain custom components, widgets, and templates developed to take advantage of framework preexisting services. These components must follow the software design pattern (standards) defined by the framework. By way of illustration, these components can be (1) combinatorial optimization problem templates, (2) metaheuristics, (3) subordinated heuristic, (4) external environment, (5) benchmark test (problem instances and a particular search configuration), etc.

The repository was designed into two levels, (1) global and (2) local. Global repo is accessible to everyone, and the community sustains it; they can download or upload components. The local repo represents local storage, where a person who works in a COP experiment downloaded components from the global repo or kept his/her developed new "pieces of software".

Furthermore, the repository must be supported by a database that maintains information about the components. The concrete component in the repository must have metadata to describe them, and it should be stored in the database. Additionally, COP or metaheuristics could consider the preliminary classification system presented in Sections 2.3 and 2.6 as a primary reference to form their metadata. Consequently, each type of component should be split into categories, and each type of component defines its own metadata. It should also consider the component author/owner (code), algorithm creator, paper that proposed the metaheuristics (if it is sharable—copyright), etc. Therefore, these components can be tagged properly according to their type. This metadata system also can support a search engine, enabling the filtering of the components by some criteria. Additionally, this database could also supply the main concepts of metaheuristics and COP (analogous to a wiki).

Moreover, the repository must provide a benchmarking process to compare metaheuristic performance. Thus, the database also should keep benchmarks of metaheuristic performance. This section of the database, in a summarized way, needs to keep two types of information. The first type of information is the benchmark test, and they can be seen as templates that the community will reuse. A benchmark test template comprises several problem instances (data) designed to evaluate metaheuristic performance in investigating these search spaces (data). The second category of information results from metaheuristic performance after performing a specific benchmark test (template). These tests must follow an experimental design and provide statistical information about their performances. Figure 6 shows the basic relationship between metaheuristics and benchmark tests.

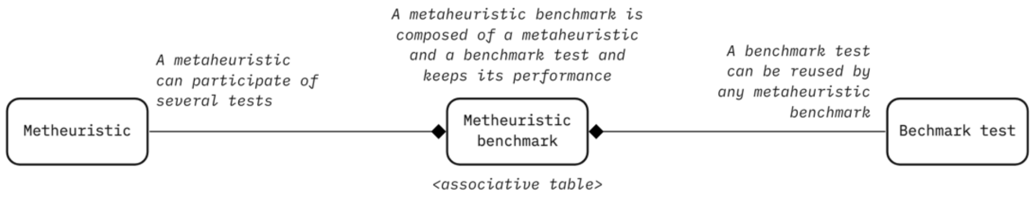

A metaheuristic can participate of several tests
A metaheuristic benchmark is composed of a metaheuristic and a benchmark test and keeps its performance
A benchmark test can be reused by any metaheuristic benchmark
Metheuristic
Metheuristic benchmark
Bechmark test
<associative table>

Figure 6. Basic relationship between metaheuristic and benchmark test.

The critical success factor is to develop wide and challenging tests by covering several aspects to defy the metaheuristics capacity. Consequently, a good test could consider different levels of difficulty and search space landscapes. These tests also could consider different types of problems such as routing, planning, scheduling, etc. Additionally, they can consider problems with a huge number of dimensions and so on so forth. The objective behind this is to generate *reference tests* that can be used as scientific standards to appraise and certify metaheuristics' performance.

Therefore, from a community perspective, it demands proposals of challenging benchmark tests by covering several aspects to defy the metaheuristics capacity. We believe it is an open question, and it could be a good research line to develop the field. Research questions such as the following could be addressed. (1) What tests are capable of evaluating the capacity of a metaheuristic as a searcher method? Or still, (2) what are the dimensions that should be explored and considered in these tests?

Thus, several metaheuristics can be subjected to the same batch of tests and their performance can be compared. Additionally, the tests already performed do not need to be executed again because they are already stored in the database. Furthermore, considering a new metaheuristic, only this algorithm should perform the same batch of tests and after this, its results can be compared with the entire database.

Furthermore, a new metaheuristic to be added in the global repository should provide the metadata information and must be checked by some standard tests that evaluate and keep this information in the database. Consequently, any person who is interested in using the algorithm can know several characteristics and the general performance of the algorithm. The database could also provide a benchmarking view, where it is possible to compare all tested metaheuristics by test or still considering all available tests.

As much the community feeds this database, more complete will be the knowledge, and more conclusions can be found about metaheuristics. For example, for a new metaheuristic, it is possible to understand in advance its general performance. If it is not performing well, the researcher can try other things and test again. It is also possible to assume in which scenarios (kind of problems) a metaheuristic may work well. Other more general guesses also can be made, such as which behaviors are more suitable for a determined class of problems, and so on.

4.1.2. Search Process Management

The Search Process Management (SPM) component aims to encapsulate and orchestrate the search process in a standard way. It acts like a “wrapper” and “adapter”, and it is responsible for preparing and automatizing the search process by interpreting all specifications and instantiating all the objects that are necessary to perform a specific search process (e.g., concrete versions of COP and MH). No matter the metaheuristic used, it also will provide the same functioning mechanism, in other words, the same API. Consequently, it will interact directly with the concrete metaheuristics, and it must be transparent for the author of the experiment. He/she only needs to pass some parameters to SPM such as (1) problem (template and instance), (2) metaheuristic (or metaheuristics list), (3) metaheuristics settings, and (4) search settings. The only mutable spec is the metaheuristics settings (3) because they depend on their parameters' definition.

Moreover, the SPM should provide several searching modes; the most common identified by this study are described as follows. (1) Single-mode is performed by only one metaheuristic. In contrast, (2) multiple-mode is accomplished by a list of metaheuristics. However, they do not share information. (3) The cooperative mode also is realized by several metaheuristics but cooperatively. Therefore, they must be able to share results. (4) The experiment mode is where a configuration manages a set of possible executions and keeps all the data. As a result, an experiment report by comparing all performances tested should be provided (depending on the experiment management component). It is also possible to perform the search in a (5) meta or automatic-mode, where a search is performed in a meta-level, where a metaheuristic orchestrates a target metaheuristic and operates hyper-parametrization tuning automatically.

Furthermore, the modes that consider more than one algorithm or multi-population search also can activate a parallel processing feature. It enables several processes to run simultaneously.

The SPM has a configuration component that is responsible for keeping and interpreting a framework configuration. The local environment has a master configuration that defines the default behavior of the framework and its core components. However, it also must permit custom configurations. If a custom configuration is passed as a parameter for a search process instance, it will override the standard behavior. Still, if it is necessary, the master configuration can be modified, directly changing some behaviors used by default.

4.1.3. Problem Representation System

The Problem Representation System (PRS) aims to organize problem definitions and all required mechanisms to create and manipulate solutions in a generic way. It is based on a state machine that interprets this encoding to manipulate solutions. In this case, the state is defined by an arrangement of elements that compose a solution instance. For instance, by receiving a solution's encoding, a solution factory component (SFC) should be able to create any solution for a determined problem. In the same way, a manipulation engine component (MEC), by receiving this encoding, should be able to recommend transformations that are requested by subordinated heuristics. For example, considering that one position of the solution must be changed, the MEC can return a proper alternative to replace the current element. It is a generic method that knows how to generate valid elements randomly according to the encoding and without code changes.

Furthermore, the PRS also is supported by a Problem Syntax, a set of rules that defines and evaluates if a problem is well-defined. It should have a component responsible to test and check if the problem defined is working properly, independently of the metaheuristic that will be used. It is also based on defining how problem templates and instances should be coded.

Moreover, since the solution factory can create any solution for a determined problem is only based on the encoding, the framework needs to take advantage of this and provide a built-in subordinate heuristic for random initialization. In other words, by interpreting the solutions' encoding, they can initialize any metaheuristic randomly. It enables the experiment's author(s) to focus on the problem definition because the framework already provides a standard way to deal with problems' definitions. This system supports all subordinated heuristics by providing knowledge of the problem to enable proper transformations in the solutions.

However, the framework should provide the possibility to the authors to use a custom "build solution function" as well as an alternative "initialization SH". In fact, it is a fundamental assumption for the framework design.

4.1.4. Problem and Solution Evaluation

The Problem and Solution Evaluation Component (PSEC) is designed to generalize the way solutions will be evaluated. They were primarily conceived to evaluate the performance (quality) and feasibility of solutions. However, they also must be extended to consider other solution metrics such as population diversity. It should also be considered as an extensible feature, where new metrics can be added to the repository and configured to be evaluated in a specific search. The primary input is the problem because it has the objective function(s) and the feasibility function (it calculates if a solution is admissible if the problem has constraints). Multi-objective problems can also pass a parameter to choose an existing multi-objective analysis method. Alternatively, a new method that performs multi-objective analysis can be passed as a parameter.

The PSEC also should provide a report module to present the analysis and results. This component is supported by the Memory and Persistence and Experiment Runner. The reports can be implemented as templates that can be reused and present results in the same format.

Moreover, this evaluation system also should consider metrics to assess exploitation and exploration behaviors. It also demands a way to recognize the landscape coverage and a way to simplify high-multidimensional combinatorial optimization model.

4.1.5. External Environment (Simulation)

Since some problems need simulation or feedback from other entities, the framework must provide a way to develop and communicate with an external environment (EE). In fact, the objective functions will interact directly with these environments. However, the framework also should control the status of these environments. Therefore, the framework could test, register, start, or stop when necessary and inform the environment's status (e.g., active or inactive).

To increase the power of the usage of EE, a software design pattern should be considered to guide the development of external environment compatible framework standards. The external environment must follow a protocol and API of the framework. It tends to facilitate the process. However, if an environment that does not follow the standards that need to be used, it is necessary to know this environment interface to create the objective function, such as how to send information and interpret feedback from the environment. Moreover, the status of the environment may not be recognized by the framework directly.

4.1.6. Search Process Intelligence

This component was conceived to recommend behavior changes to a metaheuristic during a search. This component works by analyzing the search progress, making some conclusions, and recommending a change. These changes can be the adjustment of a parameter value or the exchange of a subordinated heuristic. It is intrinsically a trial-and-error task to improve the metaheuristics capacity to navigate the search space. The core mechanism can be based on a built-in method or can be implemented by a custom method. It can use the data history of many runs. Additionally, it opens a space for the use of machine learning methods to recommend the changes. It also can be implemented as a model-based method analogous to the reinforcement learning model-based approach. Therefore, there are several ways to implement and use the search process intelligently.

4.1.7. Memory and Persistence

The Memory and Persistence Component (M&PC) was designed to offer memory and persistence to the search process. Therefore, no matter whether an MH has an internal memory resource, the search process may provide a memory, and this information can be stored to be accessed after. It also can consider that a metaheuristic can continue from a point where they stopped, enabling continuous navigation (analogous to continuous learning). Still, this saved memory also could be used by other metaheuristics as an initialization that loads what other MHs examined.

Moreover, it ensures that during a cooperative search process, metaheuristics can share the same memory (collective memory) or exchange their own memories with each other. It also requires mechanisms to evaluate the exchanging of knowledge among them, and many metrics could be used (e.g., keep diversity). Additionally, a shared memory can keep numerous good solutions discovered by several searches performed by distinct metaheuristics. It can keep everything, no matter the method used, about the problem instance landscape.

4.1.8. Experiment Runner

The Experiment Runner Component (ERC) aims to provide a standard and reusable way to perform experiments. It should provide all the structure to run, interpret, and save experiments (all data gathered and formulated conclusions should be kept). It should be designed to receive an experiment specification that defines the experimental design. This experimental design (or specification), in a summarized way, is composed of:

- A set of metaheuristics (their SH variations);
- Metaheuristics' parametrization settings strategies;
- Set of problem instances; and
- Statistical tests that must be performed.

The ERC is supported by Problem and Solution Evaluation Component and Memory and Persistence Component. As result, an experiment must return a report with all dimension analysis and select the main visualization to lead and support the conclusions.

Furthermore, the ERC also can be used to perform benchmark tests. Therefore, the benchmark analysis is executed by the ERC. A benchmark test is an experiment design (template) with a set of problem instances and the chosen statistical tests. As input, it will receive a metaheuristic and its parametrization settings strategies. The results are saved as benchmarking in the database (local) and uploaded to the global repository. It also can compare its performance with other metaheuristic performances (benchmarks already executed) stored in the database.

4.2. Problem Representation: Formulation and Standards

To understand, design, and implement a combinatorial optimization problem is a challenging activity because it requires knowledge of the problem and technical skills related to optimization problems and coding. This section aims to answer RQ-4: *How can we develop a body of knowledge to formulate combinatorial optimization problems in a standard way?* By answering this question, we expect to identify the vital elements that should be part of a COP and arrange them in a general and independent software unit. Additionally, the answer to this question is a foundational premise that sustains the entire framework definition because it is the core knowledge item that will be manipulated by the algorithms. This is the reason and purpose of the framework's existence.

Moreover, to answer this question properly, we need to take into consideration the following four assumptions. The first is related to the COP composition and structure. We also need to answer the question: *What are the building blocks necessary to accommodate several categories (or all) problems in a standard design?* The second must concern conceiving COP as a self-contained unit (single unit) that provides a common behavior that can be manipulated (used or reused) by any method that knows its functioning standards. The third assumption lies in the fact that problems must be defined by a specification. A problem specification is a higher-level definition of the problem that defines the behavior of the problem class. The fourth must consider that problem specifications cannot affect or demand code changes from any algorithm of the framework (e.g., metaheuristics, subordinated heuristics, even in problem core components).

Before proceeding to the COP building blocks explanations, an essential distinction must be provided.

From a practical point of view, optimization problems have two levels of abstraction considering their specification: (1) abstract level, also known as problem template (PT), and (2) concrete level, also known as a problem instance (PI). A problem template is a conceptual model of the problem that specifies a class of problems with a specific objective. Thus, a PT encompasses the general definitions of the problem that can be followed (or reused) by several problem instances. On the other hand, a PI is based on a PT and since the template gives the conceptual level, an instance must provide the specific data that instantiate a particular case of the abstract problem. Consequently, a problem template P can have an infinite number of instances $P = \{p_1,\ p_2, \ldots,\ p_n\}$.

Therefore, a PT provides the upper-level building blocks representing the high-level problem definition and can be reused by any instance of this problem (e.g., objective(s), objective function(s), constraints requirements, solution encoding rules, decision variables design, etc.) In contrast, a PI must provide the data of the concrete problem (e.g., decision variable values and constraints values).

4.2.1. Problem Template (PT) Definitions

Definition 4. *In practical terms, since the problem template defines the higher-level building blocks, it can be defined in the following way:*

$$P = (O, S, E, \Omega|\phi|A) \ \rightarrow \{p_1,\ p_2, \ldots,\ p_n, \ldots\} \tag{4}$$

where:

- *P represents the problem template and if it is followed, it may originate an infinite number of problem instances $\{p_1,\ p_2, \ldots,\ p_n, \ldots\}$;*
- *O corresponds to a nonempty set of objectives $O = \{o_1,\ o_2,\ \ldots,\ o_n\}$;*
- *S is the search space design or decision variables design $S = \{s_1,\ s_2,\ \ldots,\ s_n\}$;*
- *E defines a nonempty set of solution encoding rules $E = \{\varepsilon_1,\ \varepsilon_2,\ \ldots,\ \varepsilon_n\}$;*
- *Ω (optional) represents set of constraint design $\Omega = \{\omega_1,\ \omega_2,\ \ldots,\ \omega_n\}$;*
- *ϕ (optional) is the admissibility function; and*
- *A (optional) represents a set of additional algorithms $A = \{\alpha_1,\ \alpha_2,\ \ldots,\ \alpha_n\}$ that can be used by instances of P.*

Definition 5. *Any objective* $\mathrm{o_i}$ *of* $\mathrm{O} = \{\mathrm{o_1},\ \mathrm{o_2},\ \ldots,\ \mathrm{o_n}\}$ *defines the objective direction and has function to determine the quality of a solution, and it can be expressed by:*

$$\mathrm{O} = \{\mathrm{o_1},\ \mathrm{o_2},\ \ldots,\ \mathrm{o_n}\},\ \forall\, \mathrm{o} \in \mathrm{O},\ \mathrm{o} = (\mu, F) \ \rightarrow \ \mathbb{R} \tag{5}$$

where:

- *μ represents the objective direction (minimization or maximization); and*
- *F corresponds to a set of the objective functions for an objective.*

A PT was envisioned to enable the possibility of a problem to have multiple objective functions. The idea is to provide the possibility of a PT to deal with more than one solution's encoding. It is vital to enable other classes of metaheuristics—based on a different encoding—that can also manipulate this problem. Therefore, potentially, it can make the problem resolvable by any metaheuristics without any code changes from both sides. The challenge will be to make these encodings equivalent. It also demands additional objective function and admissibility function.

Definition 6. *F corresponds to a set of the pairs defined by the objective function and the solution's encoding it is linked to; therefore, F can be defined as:*

$$\mathrm{F} = \{(\mathrm{f}_1, \varepsilon_1),\ (\mathrm{f}_2, \varepsilon_2) \ldots,\ (\mathrm{f}_\mathrm{n}, \varepsilon_\mathrm{n})\} \tag{6}$$

S represents the search space design (SSD), and it is a conceptual view of the search space. From a practical perspective, it shapes the decision variables' scheme by specifying their roles in the search and data types. As a result, it specifies how problem instances must supply the concrete decision variables to work properly and designed by the problem template.

Definition 7. *Therefore, a decision variable design is synthesized as:*

$$S = \{s_1,\ s_2,\ \ldots,\ s_\mathrm{n}\},\ \forall\, s \in S,\ s = (r, d) \tag{7}$$

where:

- *r represents the decision variable role in search; and*
- *d corresponds to the data type of the decision variable (the terminology for d is defined by the framework).*

Remark 4. *The SSD is not merely a convention. These specifications must be followed by the objective function, actual decision variables, etc. They are interconnected, and the "roles" link them and guarantee that they will perform adequately in a standard way.*

The solution encoding rules E is a set of higher-level definitions used to form a solution instance (representation) for a determined class of problems. This definition is the basis of the "solution factory" provided by the framework. In summary, it is a state machine that receives a solution encoding as input and can produce random solutions.

Definition 8. *A solution encoding is defined by:*

$$\forall\, \varepsilon \in \mathrm{E},\ \varepsilon = (\mathcal{S}, \mathcal{T}, \mathcal{X} | B) \ \rightarrow\ \{x_1,\ x_2,\ \ldots,\ x_n\} \tag{8}$$

where:

- *ε describes a solution encoding definition and it may originate a finite number of solutions* $\{x_1,\ x_2,\ \ldots,\ x_n\}$*;*
- *$\mathcal{S}$ defines the shape of the solution representation (model's format or data structure);*
- *$\mathcal{T}$ represents the syntax type used as basis to form;*
- *$\mathcal{X}$ defines the syntax (rules) form a valid solution; and*
- *B symbolizes the additional software that can be added to a problem instance.*

The framework must define the syntax types, and it must be designed as an extensible list of classes of syntaxes that are supported by the framework. Therefore, it enables the evolution of the framework capacities according to future needs. Moreover, each syntax type is defined by a particular syntax (rules) that embraces all the points necessary to guide the construction of the solutions, such as tokens (elements), and how they can be arranged (analogous to lexical and grammatical levels of a computer language). Again, these solution syntaxes definitions must be supported by the framework. Furthermore, the framework also needs to provide test tools to certify that a solution representation is well designed, and facilitate this definition. It also may consider a COP design tool to create a level of abstraction to guide and support this process.

Ω, the constraint's design, and ϕ, the admissibility function, are optional. Therefore, a problem must be operational without them. Their presence will depend on the characteristics and needs of the problem. Ω and ϕ are only in a problem template definition if it is a constrained optimization problem and it also requires an admissibility function to determine if a problem is feasible or not.

Definition 9. *A constraint design is defined by:*

$$\Omega = \{\omega_1,\ \omega_2,\ \ldots,\ \omega_n\}, \forall\ \omega \in\ \Omega,\ \omega = (c, w,\ t) \tag{9}$$

where:

- *c defines the variable (or concept) subjected to a constraint;*
- *w defines the comparison operator to apply the constraint rule; and*
- *t defines the constraint data type.*

Remark 6. *Generally, an admissibility function can evaluate all constraints. Thus, in our analysis, it is not necessary to implement one admissibility function per constraint. If it is necessary by a problem not perceived in our review, the constraint design also could consider an admissibility function.*

The A, set of additional algorithms $\mathrm{A} = \{\alpha_1,\ \alpha_2,\ \ldots,\ \alpha_n\}$, opens the possibility of adding "pieces of software" to the problems, generally using a heuristic of the problem or guessing some aspects of search space characteristics. Therefore, it is a way to add problem knowledge to COP in a standard way. For example, it can provide alternative "initialization subordinated heuristics" that can attempt to apply a problem-specific heuristic to initialize the metaheuristics taking advantage of this assumption.

Definition 10. *The additional algorithm α is expressed by:*

$$\mathrm{A} = \{\alpha_1,\ \alpha_2,\ \ldots,\ \alpha_\mathrm{n}\},\ \forall\ \alpha \in A\ ,\ \alpha = (a, g) \tag{10}$$

where:

- *a corresponds to the additional algorithm itself; and*
- *g indicates the class of the algorithm (group).*

Remark 6. *The class of the algorithm must be defined by the framework terminology to classify the stand-alone algorithms (self-contained pieces) of the framework according to their class. For example: "subordinated heuristic initialization", "build solution function", "subordinated heuristic for selection", etc.*

4.2.2. Problem Instance (PI) Definitions

As a complementary part, a problem instance provides the concrete elements that fulfill a problem template's definitions. Therefore, a PI must follow the PI definition to work properly. The framework is not designed to create a hierarchy of COP. Thus, a new problem template is not a subclass of a problem template superclass. The framework was envisioned to have only one problem class that is fed by PT and PI specifications. This feature is supported by a helper class named "problem decorator" that instantiates the problem by interpreting these definitions.

Definition 11. *Therefore, a problem instance is defined as follows:*

$$\forall\ p \in P,\ p = (P,\ \overline{S},\ \overline{\Omega}, |B)\ \rightarrow \{x_1,\ x_2,\ \ldots,\ x_n\} \tag{11}$$

$$\textit{based on } P = (O, S, E, \Omega|\phi|A) \rightarrow \{p_1,\ p_2,\ \ldots,\ p_n\}$$

where:

- *p is a problem instance that follows a P problem template), and it may originate a finite number of solutions $\{x_1,\ x_2,\ \ldots,\ x_n\}$;*

- *P is the problem template definition followed by the problem instance (therefore, the problem instance is self-contained unit of software);*
- *$\overline{S}$ represents the concrete search space (data) that is composed of decision variable instances $\overline{S} = \{\overline{s}_1, \overline{s}_2, \ldots, \overline{s}_n\}$;*
- *R defines the roles by linking which decision variable;*
- *$\overline{\Omega}$, corresponds to the actual values of the constraint $\overline{\Omega} = \{\overline{\omega}_1, \overline{\omega}_2, \ldots, \overline{\omega}_n\}$ and it is only relevant for the instance of the problem; and*
- *B is a list of additional algorithms $B = \{\beta_1, \beta_2, \ldots, \beta_n\}$ that can be provided by an instance of the problem.*

Definition 12. *Therefore, a decision variable instance is defined as follows:*

$$\overline{S} = \{\overline{s}_1, \overline{s}_2, \ldots, \overline{s}_n\}, \forall\, \overline{s} \in \overline{S},\ \overline{s} = \left(r, \overline{d}\right) \tag{12}$$

where:

- *r represents the decision variable role in search; and*
- *$\overline{d}$ corresponds to the data of the decision variable.*

The search space template $S = \{s_1, s_2, \ldots, s_n\}$ defines how the search space instance must be provided $\overline{S} = \{\overline{s}_1, \overline{s}_2, \ldots, \overline{s}_n\}$. In other words, each decision variable design $s = (r, d)$ requests a variable instance $\overline{s} = \left(r, \overline{d}\right)$ when a problem is built (PI).

Definition 13. *A constraint value is defined by:*

$$\overline{\Omega} = \{\overline{\omega}_1, \overline{\omega}_2, \ldots, \overline{\omega}_n\},\ \forall\, \overline{\omega} \in \overline{\Omega},\ \overline{\omega} = (c, v) \tag{13}$$

where:

- *c defines the variable (or concept) subjected to a constraint; and*
- *v defines the constraint value.*

The constraint's design $\Omega = \{\omega_1, \omega_2, \ldots, \omega_n\}$ defines how the constraint values must be provided $\overline{\Omega} = \{\overline{\omega}_1, \overline{\omega}_2, \ldots, \overline{\omega}_n\}$. Unless otherwise stated, each constraint design $\omega = (c, w, t)$ requires a constraint value $\overline{\omega} = (c, v)$ when the problem is created (PI).

The B, a set of additional algorithms $B = \{\beta_1, \beta_2, \ldots, \beta_n\}$, enables the possibility to add "pieces of software" in a problem instance. The mechanism is similar to the set A provided in the problem template. However, it is only applicable to the instance and cannot be reused by other instances. If the additional algorithm needs to be shared by other instances, it needs to be added in A. It is conceived to make a problem instance flexible permitting the addition of problem heuristics.

Definition 14. *The additional algorithm β is expressed similarly to* A, *and therefore is defined in the following way:*

$$B = \{\beta_1, \beta_2, \ldots, \beta_n\},\ \forall\, \beta \in B\ ,\ \beta = (a, g) \tag{14}$$

where:

- *a correspond to additional algorithm itself; and*
- *g indicates the class of the algorithm (group).*

These combinatorial optimization problem definitions are synthesized in Figure 7, which provides a higher-level representation of the COP building blocks in a class diagram. This diagram summarizes the crucial relationships of the building blocks to design and implement a general COP that can be manipulated by any metaheuristics that follow these framework standards.

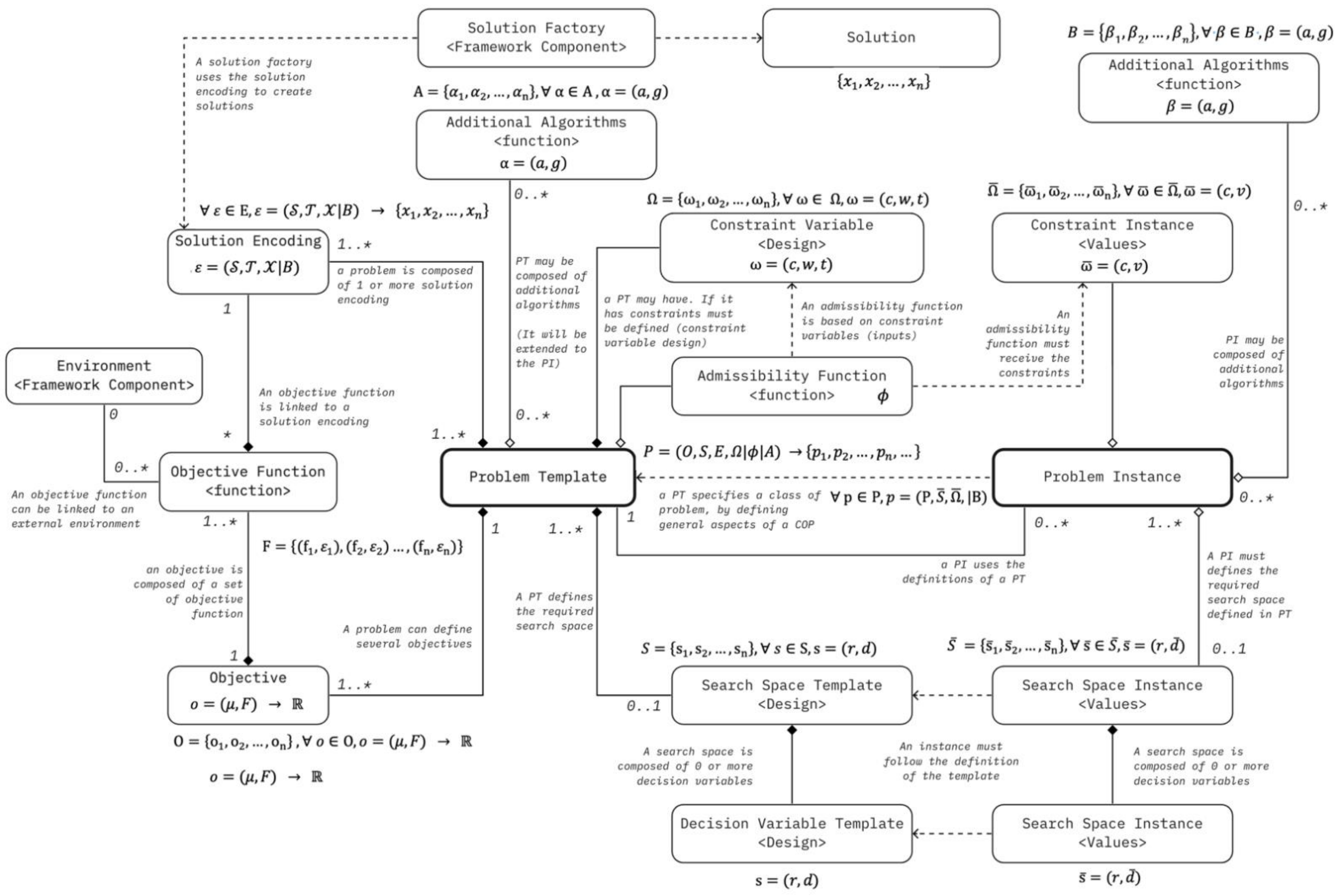

Solution Factory
<Framework Component>
A solution factory uses the solution encoding to create solutions
Solution
$\{x_1, x_2, \dots, x_n\}$
$B = \{\beta_1, \beta_2, \dots, \beta_n\}, \forall \beta \in B, \beta = (a, g)$
Additional Algorithms
<function>
$\beta = (a, g)$
$A = \{\alpha_1, \alpha_2, \dots, \alpha_n\}, \forall \alpha \in A, \alpha = (a, g)$
Additional Algorithms
<function>
$\alpha = (a, g)$
0..*
$\forall \varepsilon \in E, \varepsilon = (\mathcal{S}, \mathcal{T}, \mathcal{X}|B) \rightarrow \{x_1, x_2, \dots, x_n\}$
Solution Encoding
$\varepsilon = (\mathcal{S}, \mathcal{T}, \mathcal{X}|B)$
1..*
a problem is composed of 1 or more solution encoding
1
PT may be composed of additional algorithms
(It will be extended to the PI)
$\Omega = \{\omega_1, \omega_2, \dots, \omega_n\}, \forall \omega \in \Omega, \omega = (c, w, t)$
Constraint Variable
<Design>
$\omega = (c, w, t)$
$\bar{\Omega} = \{\bar{\omega}_1, \bar{\omega}_2, \dots, \bar{\omega}_n\}, \forall \bar{\omega} \in \bar{\Omega}, \bar{\omega} = (c, v)$
Constraint Instance
<Values>
$\bar{\omega} = (c, v)$
0..*
a PT may have. If it has constraints must be defined (constraint variable design)
An admissibility function is based on constraint variables (inputs)
An admissibility function must receive the constraints
PI may be composed of additional algorithms
Environment
<Framework Component>
0
An objective function is linked to a solution encoding
Admissibility Function
<function> ϕ
0..*
*
1..*
0..*
Objective Function
<function>
Problem Template
$P = (O, S, E, \Omega|\phi|A) \rightarrow \{p_1, p_2, \dots, p_n, \dots\}$
Problem Instance
An objective function can be linked to an external environment
1..*
1
1..*
1
a PT specifies a class of problem, by defining general aspects of a COP
$\forall p \in P, p = (P, \bar{S}, \bar{\Omega}, |B)$
0..*
0..*
1..*
$F = \{(f_1, \varepsilon_1), (f_2, \varepsilon_2) \dots, (f_n, \varepsilon_n)\}$
a PI uses the definitions of a PT
an objective is composed of a set of objective function
A PT defines the required search space
A PI must defines the required search space defined in PT
A problem can define several objectives
$S = \{s_1, s_2, \dots, s_n\}, \forall s \in S, s = (r, d)$
$\bar{S} = \{\bar{s}_1, \bar{s}_2, \dots, \bar{s}_n\}, \forall \bar{s} \in \bar{S}, \bar{s} = (r, \bar{d})$
0..1
1
Objective
$o = (\mu, F) \rightarrow \mathbb{R}$
1..*
Search Space Template
<Design>
0..1
Search Space Instance
<Values>
$O = \{o_1, o_2, \dots, o_n\}, \forall o \in O, o = (\mu, F) \rightarrow \mathbb{R}$
$o = (\mu, F) \rightarrow \mathbb{R}$
A search space is composed of 0 or more decision variables
An instance must follow the definition of the template
A search space is composed of 0 or more decision variables
Decision Variable Template
<Design>
$s = (r, d)$
Search Space Instance
<Values>
$\bar{s} = (r, \bar{d})$

Figure 7. Problem building blocks and their relationships.

4.3. Metaheuristics General Specification

Defining a software design pattern is crucial to develop more robust metaheuristics in a standard way. A software standard permits that a metaheuristic can be more easily implement and reused. It also opens a new page to enable a top-notch system of cooperation among metaheuristics. Consequently, this section aims to answer *RQ-5: How can we define a standard design for metaheuristics and completely without coding links with the problems?* The answer to this question must be accompanied by a foundational pillar, a COP standard. Therefore, the proposal of this metaheuristic software design pattern follows all the definitions supplied in Section 4.2, Problem Representation: Formulation and Standards.

It is considered that metaheuristics can be seen as a master heuristic that orchestrates a set of subordinated heuristics in an iterative cycle until certain conditions are achieved. Suppose the "content" of subordinated heuristics is removed. In that case, a metaheuristic is just a higher-level cycle that calls subordinated heuristics in a specific order according to their "roles" in the search process. A "role" can be seen as an activity that must be performed and contribute to the metaheuristic's strategy to navigate the search space.

Therefore, this higher-level cycle determines what a metaheuristic algorithm is. Additionally, the same metaheuristic algorithm can present a vast number of variations, and it will depend on the SH variation that will be used in each role. Each variation of a subordinated heuristic can change the behavior of metaheuristics by implementing a different mechanism of navigation in the search space. Additionally, the metaheuristic behavior can be changed by its parametrization that regulates some decisions during the search.

Moreover, by considering these variation necessities, a metaheuristic design pattern must also provide a system that enables an interoperability software structure with the capacity to exchange subordinated heuristics (algorithms)—in other words, the ability to replace SHs transparently in design and run time. Consequently, to make this possible, metaheuristics also must be defined as two levels of abstraction: (1) a conceptual level and (2) a concrete level. The conceptual level defines an abstract metaheuristic algorithm by specifying a higher-level cycle (meta-algorithm), the roles required by SHs in the strategy, MH parameters needs, and metadata behaviors.

Definition 15. *Therefore, an abstract metaheuristic can be defined in the following way:*

$$M = (\sigma,\ C, \Delta, \mathrm{K}) \to \{m_1, m_2,\ \ldots,\ m_n\} \tag{15}$$

where:

- *M represents an abstract metaheuristic algorithm and it can originate several possible variations of this metaheuristic algorithm $\{m_1, m_2,\ \ldots,\ m_n\}$;*
- *σ represents the metadata that describe some aspects of the metaheuristic behaviors that can be interpreted by the framework (e.g., solution encoding rules);*
- *C defines the higher-level cycles that arranges the SH calls;*
- *Δ is a list of "search role requested" $\{\delta_1,\ \delta_2,\ \ldots,\ \delta_{1n}\}$ by higher-level cycle and it will be performed by SHs; and*
- *K defines the set of parameters' specifications $K = \{k_1,\ k_2,\ \ldots,\ k_n\}$ necessary to regulate the metaheuristics' decisions during the search.*

After an intensive review and analyzing various subordinated heuristics, we concluded that they can be organized into categories according to their functioning in the search process. The types considered by this framework proposal (version 1) are the following:

- Initialization—concerns to create the initial set of solutions;
- Selection—selects solution(s) according to some criteria;
- Navigation—responsible for performing the moves in the search space;
- Replacement—aims to make decisions of how to perform the replacement of a current set of solutions by a new set of solutions which is obtained in each iteration;
- Calculation—responsible for providing calculations to support decisions; and
- Termination—used to finalize the search process and return the best solution.

The set of "search role requests" defined by $\Delta = \{\delta_1,\ \delta_2,\ \ldots,\ \delta_{1n}\}$ can also been seen as "placeholders" that must be occupied by a subordinated heuristic. Since the SH may have variants, they can be replaced by an SH that performs a similar role. Additionally, an SH must belong to the demanded by "search role requests". Moreover, a "role" is a semantic function defined by a metaheuristic algorithm and it represents a vital aspect of its search metaphor. For example, a Genetic Algorithm requires an SH that will perform the "mutation role" that belongs to the type "navigation" SH.

Definition 16. *Therefore, a "search role request" δ is defined in the following way:*

$$\Delta = \{\delta_1,\ \delta_2,\ \ldots,\ \delta_{1n}\},\ \ \forall\, \delta \in \Delta\ ,\ \delta = (\Theta, \gamma\ ) \tag{16}$$

where:

- *Θ represents a search role requested by the metaheuristic cycle; and*
- *γ defines the type of subordinated heuristic required.*

Definition 17. *The γ is defined by the framework as Γ, a finite set of SH types Γ. Thus, it is defined in the following way:*

$$\Gamma = \{initialization,\ selection,\ navigation,\ replacement,\ calculation,\ termination\} \tag{17}$$

In this context, a subordinated heuristic is a self-contained piece of software that performs a determined function. Consequently, an SH must work by receiving inputs and returning a result in a defined format, independently of who is calling it. Therefore, SH must follow an API definition and each type must specify the same signature. It is the base feature to enable the SH exchange. Moreover, SHs also must have metadata to inform their behaviors, classifying their abilities.

For example, to inform solution encoding that the SH knows how to manipulate. Additionally, the metadata is essential to register the SH in a pool (it also applies to MH metadata). It is crucial to support the replacement of SH during the search process and enable the "search process manager" to replace SH properly because it knows their behaviors and can select an equivalent.

Definition 18. *Therefore, a subordinated heuristic is expressed as:*

$$H = \{h_1,\ h_2,\ \ldots,\ h_n\}, \forall\ h \in H\ , h = (\sigma, \gamma,\ a\ ) \tag{18}$$

where:

- *h is an SH of a set of all SH implemented in the framework $H = \{h_1,\ h_2,\ \ldots,\ h_n\}$;*
- *σ represents the metadata that describe some aspects of SH behaviors; and*
- *γ defines the type of subordinated heuristic; and*
- *a symbolizes the algorithm implementation, the SH itself.*

Definition 19. *A parameter k is stated in the following way:*

$$\mathrm{K} = \{k_1,\ k_2,\ \ldots,\ k_n\}, \forall\ k \in \mathrm{K}\ ,\ k = (c, \mathrm{t}\ ) \tag{19}$$

where:

- *k is a parameter of a set defined by the metaheuristic $K = \{k_1,\ k_2,\ \ldots,\ k_n\}$;*
- *c defines the parameter name; and*
- *t is the parameter data type.*

Again, as defined in the problem, there is no metaheuristic superclass that subclasses can follow or extend. In fact, for each metaheuristic algorithm, there must exist only one class that receives a specification that determines its behavior. This class in design time is abstract and only implements the higher-level cycle. It will only have an actual function in run-time when the SHs received as input are linked to their roles. This behavior is based on Strategy Design Pattern. Consequently, the framework must provide a helper class that parses the specification, and instantiate the metaheuristic variation properly.

Definition 20. *Therefore, metaheuristic specification can be defined in the following way:*

$$\overline{m} = \left(\overline{\Delta}, \overline{K},\ p\right) \to x_{''best''} \tag{20}$$

where:

- *$\overline{m}$ is a metaheuristic variation of the metaheuristic algorithm M that after a search running return an optimal-solution and return the best-found solution $x_{''best''}$;*
- *$\overline{\Delta}$ represents the "search roles decisions" $\overline{\Delta} = \{\overline{\delta}_1,\ \overline{\delta}_2,\ \ldots,\ \overline{\delta}_n\}$; and*
- *$\overline{K}$ defines the parameters values $\overline{K} = \left\{\overline{k}_1,\ \overline{k}_2,\ \ldots,\ \overline{k}_n\right\}$; and*
- *p represents the problem instance that will be resolved by the metaheuristic.*

Definition 21. *A "search role decision" $\overline{\delta}$ is expressed as:*

$$\overline{\Delta} = \{\overline{\delta}_1,\ \overline{\delta}_2,\ \ldots,\ \overline{\delta}_n\}\ ,\ \forall\ \overline{\delta} \in \overline{\Delta}\ ,\ \overline{\delta} = (\Theta, h) \tag{21}$$

where:

- *Θ represents a role defined by the metaheuristic cycle; and*
- *h represents a subordinated heuristic that will perform a determined role in the search process.*

The "search roles requests" $\Delta = \{\delta_1,\ \delta_2,\ \ldots,\ \delta_{1n}\}$ defines how a metaheuristic specification must provide the subordinated metaheuristic in each required role $\overline{\Delta} = \{\overline{\delta}_1,\ \overline{\delta}_2,\ \ldots,\ \overline{\delta}_n\}$ (search roles decisions). Consequently, each search role requested $k = (c, \mathrm{t}\ )$ requires a search role decision $\overline{\delta} = (\Theta, h)$ when a metaheuristic is created.

Definition 22. *A parameter value $\overline{k}$ is defined as:*

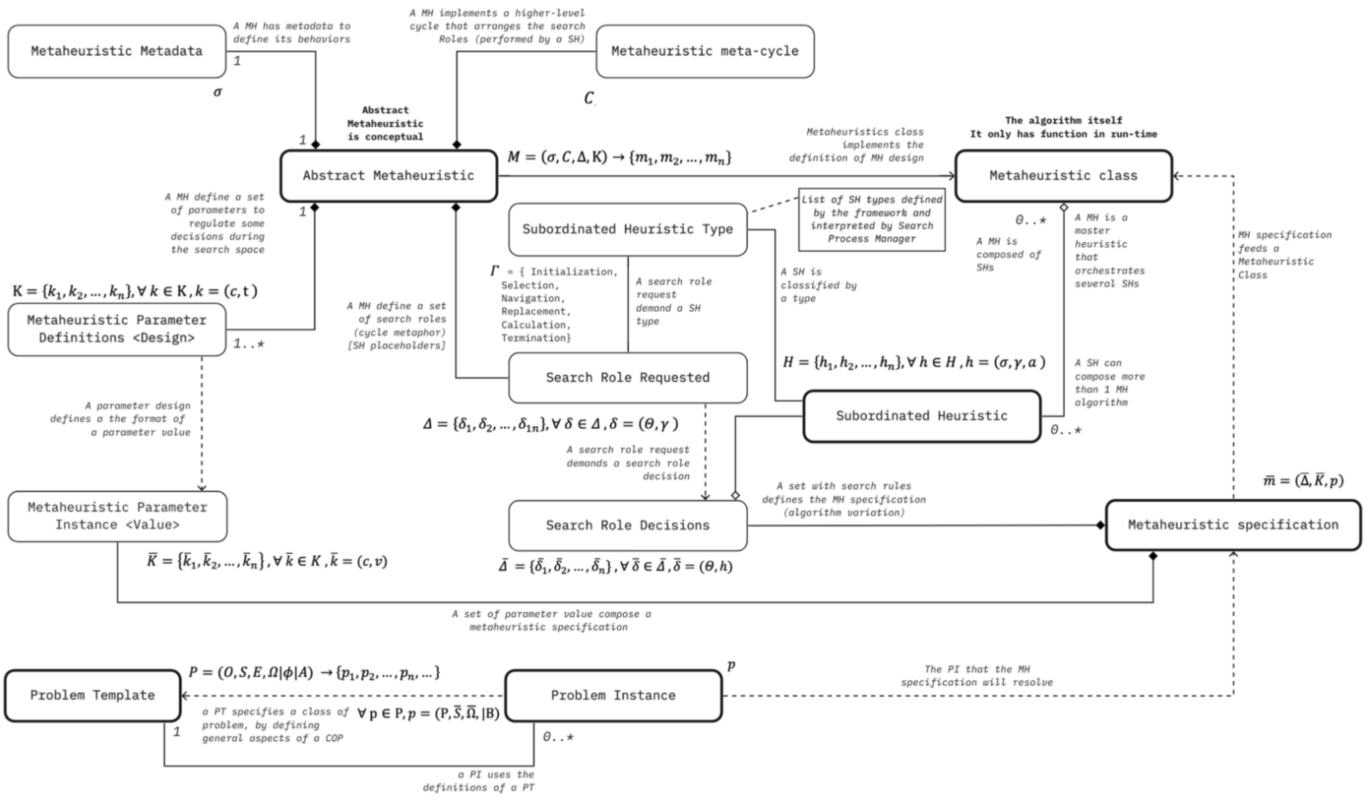
$$\overline{\mathrm{K}} = \left\{\overline{k}_1,\ \overline{k}_2,\ \ldots,\ \overline{k}_n\right\},\ \forall\ \overline{k} \in \mathrm{K}\ ,\ \overline{k} = (c, v) \tag{22}$$

where:

- *c defines the parameter name; and*
- *v defines the parameter value.*

The parameters' specification $\mathrm{K} = \{k_1,\ k_2,\ \ldots,\ k_n\}$ defines how a metaheuristic specification must provide the parameters values $\overline{\mathrm{K}} = \left\{\overline{k}_1,\ \overline{k}_2,\ \ldots,\ \overline{k}_n\right\}$. In other words, each parameter specification $k = (c, \mathrm{t}\ )$ requires a parameter value $\overline{k} = (c, v)$ when a metaheuristic is created.

Figure 8 below synthetizes all metaheuristics definitions and their relationships. It is expressed in a higher-level class diagram.



Metaheuristic Metadata
A MH has metadata to define its behaviors
1
σ
A MH implements a higher-level cycle that arranges the search Roles (performed by a SH)
Metaheuristic meta-cycle
C
Abstract Metaheuristic is conceptual
1
Abstract Metaheuristic
$M = (\sigma, C, \Delta, \mathrm{K}) \rightarrow \{m_1, m_2, \ldots, m_n\}$
Metaheuristics class implements the definition of MH design
The algorithm itself It only has function in run-time
Metaheuristic class
A MH define a set of parameters to regulate some decisions during the search space
1
Subordinated Heuristic Type
List of SH types defined by the framework and interpreted by Search Process Manager
0..*
A MH is composed of SHs
A MH is a master heuristic that orchestrates several SHs
MH specification feeds a Metaheuristic Class
$\mathrm{K} = \{k_1, k_2, \ldots, k_n\}, \forall\, k \in \mathrm{K}, k = (c, \mathrm{t})$
Metaheuristic Parameter Definitions <Design>
1..*
A MH define a set of search roles (cycle metaphor) [SH placeholders]
Γ = { Initialization, Selection, Navigation, Replacement, Calculation, Termination}
A search role request demand a SH type
A SH is classified by a type
Search Role Requested
$H = \{h_1, h_2, \ldots, h_n\}, \forall\, h \in H, h = (\sigma, \gamma, a)$
A SH can compose more than 1 MH algorithm
A parameter design defines a the format of a parameter value
$\Delta = \{\delta_1, \delta_2, \ldots, \delta_{1n}\}, \forall\, \delta \in \Delta, \delta = (\theta, \gamma)$
Subordinated Heuristic
0..*
A search role request demands a search role decision
A set with search rules defines the MH specification (algorithm variation)
$\overline{m} = (\overline{\Delta}, \overline{K}, p)$
Metaheuristic Parameter Instance <Value>
Search Role Decisions
Metaheuristic specification
$\overline{K} = \{\overline{k}_1, \overline{k}_2, \ldots, \overline{k}_n\}, \forall\, \overline{k} \in K, \overline{k} = (c, v)$
$\overline{\Delta} = \{\overline{\delta}_1, \overline{\delta}_2, \ldots, \overline{\delta}_n\}, \forall\, \overline{\delta} \in \overline{\Delta}, \overline{\delta} = (\theta, h)$
A set of parameter value compose a metaheuristic specification
$P = (O, S, E, \Omega|\phi|A) \rightarrow \{p_1, p_2, \ldots, p_n, \ldots\}$
p
The PI that the MH specification will resolve
Problem Template
Problem Instance
a PT specifies a class of problem, by defining general aspects of a COP
$\forall\, \mathrm{p} \in \mathrm{P}, p = (\mathrm{P}, \overline{\mathrm{S}}, \overline{\Omega}, |\mathrm{B})$
1
0..*
a PI uses the definitions of a PT

Figure 8. Metaheuristic definitions.

5. Research Gaps and Future Work

In this study, we explored, from various perspectives, concepts related to combinatorial optimization problems and metaheuristics, with the aim of highlighting the main challenges and issues that do not permit real progress in the field. Consequently, this study highlights several potential future studies that can contribute to the actual development of the area.

The main challenges and open question that represent interesting research avenues are summarized as follows:

- Guidelines to organize the field are essential to avoid misinterpretation of concepts and to compare metaheuristics characteristics and behaviors more efficiently and standardly. Furthermore, a classification system is essential to promote effective knowledge management and discussions of algorithms. However, a classification system must be developed, recognized, and adopted by the community in the first place. It is a clear, open question that must be answered soon.
- There is an absence of rigor and baseline tests to compare the performance of metaheuristics. As a result, the studies commonly present a biased or poor/unfair performance analysis. Thus, it demands a research line to develop standard benchmarking tests that can be reused as a reference by studies that compare metaheuristic performance. Moreover, the existence of a benchmarking database can facilitate and improve the process of performance comparison. For example, a new metaheuristic could be compared with all tested metaheuristics across several benchmarking tests.
- There is a necessity to add more statistical analyses—beyond dispersion statistics measures (e.g., mean, median, standard deviation, and outliers' analysis)—to promote more sophisticated comparisons and evaluate other aspects of metaheuristics' performance.
- It is necessary to remedy the absence of formalism to design and develop COP and metaheuristics in a standard way to enable their scalability, extensibility, and reusability. It is also an important aspect to avoid creating algorithms from scratch or developing algorithms that work only with one COP or are hard to adapt to a new situation.
- The demand for mathematical methods to evaluate other aspects of the metaheuristics building blocks such as metrics and parameters for balance ratio of exploitation and exploration, search space coverage, and solutions characteristics is essential. These needs are important to support a dynamic (or more intelligent) behavior in a metaheuristic during the search.

Nevertheless, we understand that many other research lines are also required, and, unfortunately, we could not cover them due to some limitations and research decisions. Consequently, we also strongly reinforce the necessity of research to evaluate other aspects that could unleash the metaheuristic potential and promote the field's evolution.

As future work, we are focused on the development of this framework, named ToGO (Towards Global Optimal). In fact, it has already started and soon it will be shared with the community. ToGO is an extensible open-box and open-source framework to solve COP by using metaheuristics. ToGO also was conceived to enable the sharing of COP and metaheuristics experiments; for example, by sharing their performance in a global database that can be used by other studies.

Therefore, to evolve this framework, the research community must be involved. In fact, it must be maintained by the community, as it is a fundamental assumption and critical factor of success for this project. This is vital, especially considering the wide variety of metaheuristics algorithms and support components that can be shared to improve the quality of problems' solutions. Thus, community cooperation can occur on three levels: (1) the framework core components (version of distribution), (2) extensible components that run above the structure, and (3) sharing data and metrics.

6. Discussion and Conclusions

The success of metaheuristics to solve the COP was demonstrated by the past 30 years of intensive publications. They have proven their worth as efficient methods for solving challenging problems. In this period, numerous metaheuristics were created by exploring several different approaches to investigate the search space. The research focus on novelty was essential to the progress of the field.

Nevertheless, in the past decade, many new algorithms may not have necessarily led to innovation. Since around 2017, some studies have started to criticize this excessive focus on novelty (a factor proven in many studies since then). The research community also has begun to highlight the lack of focus on field consolidation due to this strong focus on new algorithm research. Additionally, they have pointed out the lack of standards for organizing the field and guiding the design and development of COP and metaheuristics.

These points clearly show that we are living in a transition time in the field of COP and metaheuristics. We are moving from a mindset focused on new algorithms—responsible for past success—to a mindset more focused on increasing metaheuristics' maturity as a scientific field. Moreover, the COP and metaheuristics field also must follow the same path that other machine learning algorithms take, such as deep learning, which evolved considerably in past years.

This evidence, in a certain way, was already addressed by other studies. However, even so, this page looks hard to be turned because the chances of publishing a new algorithm are still high. Therefore, we argue that a new mindset for the future of COP and metaheuristics must guide the research community to intensify the studies to put this field at a higher maturity level.

In this study, we researched different aspects to feed this new required mindset, focusing on the matureness of metaheuristics as a scientific field. The main challenges that do not permit real progress of the field were highlighted. Moreover, the framework proposed several potential solutions or paths to overcome these issues and contribute to the progress of the field. The proposed framework aims to contribute to two perspectives by providing (1) conceptual and (2) concrete drivers.

The conceptual level arranges all the framework definitions and the references to develop and extend components of the framework (which defines the software pieces' structure and function signatures). It also defines the ecosystem of components, their roles and relationships, and the framework's architecture. These conceptual definitions can also be used as a basis to develop another framework for COP and MH. The concrete level is the framework code itself and it will be delivered soon to the community.

A standard formalism tends to provide several benefits such as reusability, scalability, extensibility, productivity, etc. Nevertheless, despite the advantages, a software framework and a standard formalism may produce some side effects. These issues can be translated into several questions that may lead to insights and improvements for the framework development, such as:

- How can we reduce the learning curve to use and develop components for the framework?
- How can we guarantee the proper level of freedom to create new algorithms without compromising the framework services and without compromising innovation and creativity?
- How can we provide the required understanding and visibility of the built-in components (black box to white box)?
- How can we enable changes in "core" components' behaviors or permit the exchange of "core" components with customized ones?
- How can we mitigate the possibility of reducing the performance of the algorithms? (Since a software framework also may add some overhead and multi-purpose tends to be slower than single-purpose algorithms.)
- How can we guarantee the quality of all contributions during the development of the framework?

As a consequence, by recognizing the existence of potential issues, the ToGO framework was envisioned by thinking on strategies necessary to mitigate or eliminate these side effects. However, the questions above are only a small fraction of vital points and challenges that must be considered. Thus, it is necessary to evaluate more potential issues and find suitable solutions for the framework development.

By way of illustration, the learning curve issue can be mitigated by good framework documentation, examples of implementations, and a forum of discussions. In addition, a simple software structure and a well-defined API can facilitate its usage and adoption—in other words, create a software framework prepared and designed to facilitate its use and extensibility (user-centered).

In addition, it is noteworthy that the ToGO was conceived as an open-source, white-box, and extensible framework. Thus, it provides standard built-in modules that can be reused when needed. Furthermore, it was designed as an effective interchangeable software mechanism, where any "software piece" (core or not) can be replaced by another "customized software piece" since it follows the framework API definitions. Therefore, the ToGO was envisioned to maintain the possibility to develop algorithms from scratch and replace any core components if it is necessary. The idea is to keep the framework flexible by providing a proper level of freedom to create new algorithms and, at the same time, offer a layer of several services. The nature and diversity of metaheuristics and COP demand this feature from the framework to be a wide-range and robust software solution. For instance, the lists of additional algorithms $\mathrm{A} = \{\alpha_1, \alpha_2, \ldots, \alpha_\mathrm{n}\}$ and $B = \{\beta_1, \beta_2, \ldots, \beta_n\}$ were conceived with this idea in mind, passing customized algorithms, problem knowledge, or heuristics to the search process.

Moreover, the framework development will see several advantages of Python programming language by using code injection and features of functional, object-oriented programming paradigms. Additionally, the ecosystem of components is based on various design patterns (i.e., strategy, decorator, template, builder, factory, etc.). Over and above that, all software pieces are merely conceptual (placeholders) in design-time. These placeholders are roles to be played that concrete software pieces offered in run-time will fulfill.

Furthermore, the framework does not use inheritance to reduce overhead. Thus, the behavior changes occur by using interchangeable software mechanisms or by configuration management. Therefore, a search process, COP, or MH is assembled, function by function, in run-time. Consequently, we believe this approach will take advantage of the multi-purpose algorithms, but at the same time, the algorithms will behave like specialist algorithms. However, this hypothesis must be tested and evaluated during the development to mitigate the possible performance effects.

Nevertheless, we recognize that many other implications exist that must be considered in the framework development. Consequently, dealing with these side effects is a critical success factor in developing the framework properly. We also understand that the first version will not resolve all implications. Therefore, the issues must be identified and prioritized, and a roadmap of versions must cover these points. Again, we expect that the community can perform a crucial role by raising issues and developing solutions to strengthen the framework. Because of this, the active cooperation of the community is vital for the progress of the field.

The framework can originate excellent software for educational and academic usage initially. However, after its adoption and evolution as a tool, we also hope it could be used in the business context, while keeping its open-source nature. As a final thought, this research started based on the hypothesis that COP and MH can act as a "front-end" of any artificial intelligence technique by considering that any technique configuration can be translated into a combinatorial optimization problem. Consequently, the framework empowers the metaheuristic approach by enabling them to act in a higher level of abstraction. Therefore, after the consolidation of the ToGO, it may lead to a universal AI tool (front-end) with low code and humanless automated features without compromising the interference (customization) and understanding (white-box) of the user.

Author Contributions: Conceptualization, F.P. and M.C.; methodology, F.P.; validation, F.P., M.C.; investigation, F.P.; resources, M.C.; writing—original draft preparation, F.P.; writing—review and editing, M.C.; visualization, F.P.; supervision, M.C.; project administration, F.P. and M.C.; funding acquisition, M.C. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by national funds through the FCT (Fundação para a Ciência e a Tecnologia) by the projects GADgET (DSAIPA/DS/0022/2018).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Jahwar, A.F.; Abdulazeez, A.M. Meta-Heuristic Algorithms for K-Means Clustering: A Review. *PalArch's J. Archaeol. Egypt* **2021**, *17*, 20.
2. Boussaïd, I.; Lepagnot, J.; Siarry, P. A Survey on Optimization Metaheuristics. *Inf. Sci.* **2013**, *237*, 82–117. [CrossRef]
3. Fister, I., Jr.; Yang, X.-S.; Fister, I.; Brest, J.; Fister, D. A Brief Review of Nature-Inspired Algorithms for Optimization. *arXiv* **2013**, arXiv:1307.4186.
4. Parejo, J.A.; Ruiz-Cortés, A.; Lozano, S.; Fernandez, P. Metaheuristic Optimization Frameworks: A Survey and Benchmarking. *Soft Comput.* **2012**, *16*, 527–561. [CrossRef]
5. Eshtay, M.; Faris, H.; Obeid, N. Metaheuristic-Based Extreme Learning Machines: A Review of Design Formulations and Applications. *Int. J. Mach. Learn. Cybern.* **2019**, *10*, 1543–1561. [CrossRef]
6. Hussain, K.; Salleh, M.N.M.; Cheng, S.; Shi, Y. Metaheuristic Research: A Comprehensive Survey. *Artif. Intell. Rev.* **2019**, *52*, 2191–2233. [CrossRef]
7. Gogna, A.; Tayal, A. Metaheuristics: Review and Application. *J. Exp. Theor. Artif. Intell.* **2013**, *25*, 503–526. [CrossRef]
8. Abdmouleh, Z.; Gastli, A.; Ben-Brahim, L.; Haouari, M.; Al-Emadi, N.A. Review of Optimization Techniques Applied for the Integration of Distributed Generation from Renewable Energy Sources. *Renew. Energy* **2017**, *113*, 266–280. [CrossRef]
9. Brownlee, A.E.I.; Woodward, J.R.; Swan, J. Metaheuristic Design Pattern: Surrogate Fitness Functions. In Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation, Madrid, Spain, 11–15 July 2015; pp. 1261–1264.
10. Juan, A.A.; Faulin, J.; Grasman, S.E.; Rabe, M.; Figueira, G. A Review of Simheuristics: Extending Metaheuristics to Deal with Stochastic Combinatorial Optimization Problems. *Oper. Res. Perspect.* **2015**, *2*, 62–72. [CrossRef]
11. Juan, A.A.; Kelton, W.D.; Currie, C.S.M.; Faulin, J. Simheuristics Applications: Dealing with Uncertainty in Logistics, Transportation, and Other Supply Chain Areas. In Proceedings of the 2018 Winter Simulation Conference (WSC), Gothenburg, Sweden, 9–12 December 2018; pp. 3048–3059.
12. Molina, D.; Poyatos, J.; Del Ser, J.; García, S.; Hussain, A.; Herrera, F. Comprehensive Taxonomies of Nature- and Bio-Inspired Optimization: Inspiration versus Algorithmic Behavior, Critical Analysis and Recommendations. *Cogn. Comput.* **2020**, *12*, 897–939. [CrossRef]
13. Del Ser, J.; Osaba, E.; Molina, D.; Yang, X.-S.; Salcedo-Sanz, S.; Camacho, D.; Das, S.; Suganthan, P.N.; Coello Coello, C.A.; Herrera, F. Bio-Inspired Computation: Where We Stand and What's Next. *Swarm Evol. Comput.* **2019**, *48*, 220–250. [CrossRef]
14. Glover, F.; Kochenberger, G.A. *Handbook of Metaheuristics*; Springer: New York, NY, USA, 2003.
15. Sorensen, K.; Sevaux, M.; Glover, F. A History of Metaheuristics. *arXiv* **2017**, arXiv:1704.00853v1.
16. Nesmachnow, S. An Overview of Metaheuristics: Accurate and Efficient Methods for Optimisation. *Int. J. Metaheuristics* **2014**, *3*, 320. [CrossRef]
17. Wolpert, D.H.; Macready, W.G. No Free Lunch Theorems for Optimization. *IEEE Trans. Evol. Comput.* **1997**, *1*, 67–82. [CrossRef]
18. Piotrowski, A.P. Regarding the Rankings of Optimization Heuristics Based on Artificially-Constructed Benchmark Functions. *Inf. Sci.* **2015**, *297*, 191–201. [CrossRef]
19. Stegherr, H.; Heider, M.; Hähner, J. Classifying Metaheuristics: Towards a Unified Multi-Level Classification System. *Nat. Comput.* **2020**, 1–17. [CrossRef]
20. Swan, J.; Adriaensen, S.; Brownlee, A.E.I.; Hammond, K.; Johnson, C.G.; Kheiri, A.; Krawiec, F.; Merelo, J.J.; Minku, L.L.; Özcan, E.; et al. Metaheuristics "In the Large". *arXiv* **2021**, arXiv:2011.09821Cs.
21. Mitchell, T.M. *Machine Learning*; McGraw-Hill Series in Computer Science; McGraw-Hill: New York, NY, USA, 1997; ISBN 9780070428072.
22. Antoniou, A.; Murray, W.; Wright, M.H. *Practical Optimization: Algorithms and Engineering Applications*; Springer: New York, NY, USA, 2007; ISBN 9780387711065.
23. Ponce-Ortega, J.M.; Hernández-Pérez, L.G. *Optimization of Process Flowsheets through Metaheuristic Techniques*; Springer International Publishing: Cham, Switzerland, 2019; ISBN 9783319917214.
24. Tang, K.S. (Ed.) *Multiobjective Optimization Methodology: A Jumping Gene Approach*; Industrial Electronics Series; CRC Press: Boca Raton, FL, USA, 2012; ISBN 9781439899199.

25. Yang, X.-S. Metaheuristic Optimization. *Scholarpedia* **2011**, *6*, 11472. [CrossRef]
26. Yang, X.-S. *Nature-Inspired Optimization Algorithms*, 1st ed.; Elsevier: Amsterdam, The Netherland; Boston, MA, USA, 2014; ISBN 9780124167438.
27. Bianchi, L.; Dorigo, M.; Gambardella, L.M.; Gutjahr, W.J. A Survey on Metaheuristics for Stochastic Combinatorial Optimization. *Nat. Comput.* **2009**, *8*, 239–287. [CrossRef]
28. Stork, J.; Eiben, A.E.; Bartz-Beielstein, T. A New Taxonomy of Continuous Global Optimization Algorithms. *Nat. Comput.* **2020**, 1–24. [CrossRef]
29. Gendreau, M.; Potvin, J.-Y. (Eds.) *Handbook of Metaheuristics*; International Series in Operations Research & Management Science; Springer International Publishing: Cham, Switzerland, 2019; Volume 272, ISBN 9783319910857.
30. Korte, B.; Vygen, J. *Combinatorial Optimization*; Algorithms and Combinatorics; Springer: Berlin/Heidelberg, Germany, 2012; Volume 21, ISBN 9783642244872.
31. KumarBhati, R.; Rasool, A. Quadratic Assignment Problem and Its Relevance to the Real World: A Survey. *Int. J. Comput. Appl.* **2014**, *96*, 42–47. [CrossRef]
32. Dostál, J. Theory of Problem Solving. *Procedia Soc. Behav. Sci.* **2015**, *174*, 2798–2805. [CrossRef]
33. Bandaru, S.; Deb, K. Metaheuristic Techniques. In *Decision Sciences*; Sengupta, R., Gupta, A., Dutta, J., Eds.; CRC Press: Boca Raton, FL, USA, 2016; pp. 693–750. ISBN 9781466564305.
34. Blum, C.; Roli, A. Hybrid Metaheuristics: An Introduction. In *Hybrid Metaheuristics*; Blum, C., Aguilera, M.J.B., Roli, A., Sampels, M., Eds.; Studies in Computational Intelligence; Springer: Berlin/Heidelberg, Germany, 2008; Volume 114, pp. 1–30. ISBN 9783540782940.
35. Papadimitriou, C.; Steiglitz, K. Combinatorial Optimization:Algorithms and Complexity. *IEEE Trans. Acoust. Speech Signal Process.* **1984**, *32*, 1258–1259. [CrossRef]
36. Blum, C. Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison. *ACM Comput. Surv.* **2003**, *35*, 41. [CrossRef]
37. Woon, S.F.; Rehbock, V. A Critical Review of Discrete Filled Function Methods in Solving Nonlinear Discrete Optimization Problems. *Appl. Math. Comput.* **2010**, *217*, 25–41. [CrossRef]
38. Liu, Q.; Li, X.; Liu, H.; Guo, Z. Multi-Objective Metaheuristics for Discrete Optimization Problems: A Review of the State-of-the-Art. *Appl. Soft Comput.* **2020**, *93*, 106382. [CrossRef]
39. Sergienko, I.V.; Shylo, V.P. Problems of Discrete Optimization: Challenges and Main Approaches to Solve Them. *Cybern. Syst. Anal.* **2006**, *42*, 465–482. [CrossRef]
40. Xin, B.; Chen, L.; Chen, J.; Ishibuchi, H.; Hirota, K.; Liu, B. Interactive Multiobjective Optimization: A Review of the State-of-the-Art. *IEEE Access* **2018**, *6*, 41256–41279. [CrossRef]
41. Sawaragi, Y.; Nakayama, H.; Tanino, T. *Theory of Multiobjective Optimization*; Acadamic Press: Orlando, FL, USA, 1985; p. 311.
42. Peitz, S.; Dellnitz, M. A Survey of Recent Trends in Multiobjective Optimal Control—Surrogate Models, Feedback Control and Objective Reduction. *Math. Comput. Appl.* **2018**, *23*, 30. [CrossRef]
43. Boissier, M.; Schlosser, R.; Uflacker, M. Hybrid Data Layouts for Tiered HTAP Databases with Pareto-Optimal Data Placements. In Proceedings of the 2018 IEEE 34th International Conference on Data Engineering (ICDE), Paris, France, 16–19 April 2018; pp. 209–220.
44. Chinchuluun, A.; Pardalos, P.M.; Migdalas, A.; Pitsoulis, L. (Eds.) *Pareto Optimality, Game Theory and Equilibria*; Springer Optimization and Its Applications; Springer: New York, NY, USA, 2008; Volume 17, ISBN 9780387772462.
45. Li, M.; Yang, S.; Liu, X. Pareto or Non-Pareto: Bi-Criterion Evolution in Multiobjective Optimization. *IEEE Trans. Evol. Comput.* **2016**, *20*, 645–665. [CrossRef]
46. Baños, R.; Gil, C.; Paechter, B.; Ortega, J. A Hybrid Meta-Heuristic for Multi-Objective Optimization: MOSATS. *J. Math. Model. Algorithms* **2007**, *6*, 213–230. [CrossRef]
47. Doerner, K.F.; Maniezzo, V. Metaheuristic Search Techniques for Multi-Objective and Stochastic Problems: A History of the Inventions of Walter J. Gutjahr in the Past 22 Years. *Cent. Eur. J. Oper. Res.* **2018**, *26*, 331–356. [CrossRef]
48. Bonyadi, M.R.; Michalewicz, Z.; Przybylek, M.R.; Wierzbicki, A. Socially Inspired Algorithms for the Travelling Thief Problem. In Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation, Vancouver, BC, Canada, 12–16 July 2014; pp. 421–428.
49. Przybylek, M.R.; Wierzbicki, A.; Michalewicz, Z. Multi-Hard Problems in Uncertain Environment. In Proceedings of the Genetic and Evolutionary Computation Conference 2016, Denver, CO, USA, 20–24 July 2016; pp. 381–388.
50. Przybylek, M.R.; Wierzbicki, A.; Michalewicz, Z. Decomposition Algorithms for a Multi-Hard Problem. *Evol. Comput.* **2018**, *26*, 507–533. [CrossRef] [PubMed]
51. Oliva, D.; Copado, P.; Hinojosa, S.; Panadero, J.; Riera, D.; Juan, A.A. Fuzzy Simheuristics: Solving Optimization Problems under Stochastic and Uncertainty Scenarios. *Mathematics* **2020**, *8*, 2240. [CrossRef]
52. Neumann, F.; Witt, C. *Combinatorial Optimization and Computational Complexity*; Springer: Berlin/Heidelberg, Germany, 2010.
53. Fister, I.; Fister, I., Jr.; Yang, X.-S.; Brest, J. A Comprehensive Review of Firefly Algorithms. *Swarm Evol. Comput.* **2013**, *13*, 34–46. [CrossRef]
54. Nordin, N.N.; Lee, L.-S. Heuristics and Metaheuristics Approaches for Facility Layout Problems: A Survey. *Pertanika J. Sch. Res. Rev.* **2016**, *2*, 15.

55. Adetunji, K.E.; Hofsajer, I.W.; Abu-Mahfouz, A.M.; Cheng, L. A Review of Metaheuristic Techniques for Optimal Integration of Electrical Units in Distribution Networks. *IEEE Access* **2021**, *9*, 5046–5068. [CrossRef]
56. Almonacid, B. AutoMH: Automatically Create Evolutionary Metaheuristic Algorithms Using Reinforced Learning. *Math. Comput. Sci.* **2021**. [CrossRef]
57. Sörensen, K. Metaheuristics-the Metaphor Exposed. *Int. Trans. Oper. Res.* **2015**, *22*, 3–18. [CrossRef]
58. Črepinšek, M.; Liu, S.-H.; Mernik, M. Exploration and Exploitation in Evolutionary Algorithms: A Survey. *ACM Comput. Surv.* **2013**, *45*, 1–33. [CrossRef]
59. Xu, J.; Zhang, J. Exploration-Exploitation Tradeoffs in Metaheuristics: Survey and Analysis. In Proceedings of the 33rd Chinese Control Conference, Nanjing, China, 28–30 July 2014; pp. 8633–8638.
60. Tzanetos, A.; Fister, I.; Dounias, G. A Comprehensive Database of Nature-Inspired Algorithms. *Data Brief* **2020**, *31*, 105792. [CrossRef] [PubMed]
61. Lones, M.A. Metaheuristics in Nature-Inspired Algorithms. In Proceedings of the Companion Publication of the 2014 Annual Conference on Genetic and Evolutionary Computation, Vancouver, BC, Canada, 12–16 July 2014; pp. 1419–1422.
62. Lones, M.A. Mitigating Metaphors: A Comprehensible Guide to Recent Nature-Inspired Algorithms. *SN Comput. Sci.* **2020**, *1*, 49. [CrossRef]
63. Akhtar, A. Evolution of Ant Colony Optimization Algorithm—A Brief Literature Review. *arXiv* **2019**, arXiv:1908.08007.
64. Moles, C.G. Parameter Estimation in Biochemical Pathways: A Comparison of Global Optimization Methods. *Genome Res.* **2003**, *13*, 2467–2474. [CrossRef] [PubMed]
65. Shi, J.; Zhang, Q. A New Cooperative Framework for Parallel Trajectory-Based Metaheuristics. *Appl. Soft Comput.* **2018**, *65*, 374–386. [CrossRef]
66. Talbi, E.-G. A Taxonomy of Hybrid Metaheuristics. *J. Heuristics* **2002**, *8*, 541–564. [CrossRef]
67. Talbi, E.-G. *Metaheuristics: From Design to Implementation*; John Wiley & Sons: Hoboken, NJ, USA, 2009; ISBN 9780470278581.
68. Li, Z.; Tam, V.; Yeung, L.K. An Adaptive Multi-Population Optimization Algorithm for Global Continuous Optimization. *IEEE Access* **2021**, *9*, 19960–19989. [CrossRef]
69. Toczé, K.; Nadjm-Tehrani, S. A Taxonomy for Management and Optimization of Multiple Resources in Edge Computing. *Wirel. Commun. Mob. Comput.* **2018**, *2018*, 1–23. [CrossRef]
70. Anantharaj, B.; Balaji, N.; Basha, M.S.S.; Vengattaraman, T. A Survey of Nature Inspired Algorithms. *Int. J. Appl. Eng. Res.* **2015**, *10*, 13.
71. Chakraborty, A.; Kar, A.K. Swarm Intelligence: A Review of Algorithms. In *Nature-Inspired Computing and Optimization*; Patnaik, S., Yang, X.-S., Nakamatsu, K., Eds.; Modeling and Optimization in Science and Technologies; Springer International Publishing: Cham, Switzerland, 2017; Volume 10, pp. 475–494. ISBN 9783319509198.
72. Rajakumar, R.; Dhavachelvan, P.; Vengattaraman, T. A Survey on Nature Inspired Meta-Heuristic Algorithms with Its Domain Specifications. In Proceedings of the 2016 International Conference on Communication and Electronics Systems (ICCES), Coimbatore, India, 21–22 October 2016; pp. 1–6.
73. Crespo-Cano, R.; Cuenca-Asensi, S.; Fernández, E.; Martínez-Álvarez, A. Metaheuristic Optimisation Algorithms for Tuning a Bioinspired Retinal Model. *Sensors* **2019**, *19*, 4834. [CrossRef]
74. Kamath, S.S.; Ananthanarayana, V.S. A Bio-Inspired, Incremental Clustering Algorithm for Semantics-Based Web Service Discovery. *Int. J. Reason. Based Intell. Syst.* **2015**, *7*, 261. [CrossRef]
75. Li, G.; Jin, Y.; Akram, M.W.; Chen, X.; Ji, J. Application of Bio-Inspired Algorithms in Maximum Power Point Tracking for PV Systems under Partial Shading Conditions—A Review. *Renew. Sustain. Energy Rev.* **2018**, *81*, 840–873. [CrossRef]
76. Martarelli, N.J.; Nagano, M.S. Unsupervised Feature Selection Based on Bio-Inspired Approaches. *Swarm Evol. Comput.* **2020**, *52*, 100618. [CrossRef]
77. Dokeroglu, T.; Sevinc, E.; Kucukyilmaz, T.; Cosar, A. A Survey on New Generation Metaheuristic Algorithms. *Comput. Ind. Eng.* **2019**, *137*, 106040. [CrossRef]
78. Baghel, M.; Agrawal, S.; Silakari, S. Survey of Metaheuristic Algorithms for Combinatorial Optimization. *Int. J. Comput. Appl.* **2012**, *58*, 21–31. [CrossRef]
79. Bhattacharyya, S. (Ed.) *Hybrid Metaheuristics for Image Analysis*; Springer International Publishing: Cham, Switzerland, 2018; ISBN 9783319776248.
80. Blum, C.; Puchinger, J.; Raidl, G.; Roli, A. Hybrid Metaheuristics. In *Hybrid Optimization*; Van Hentenryck, P., Milano, M., Eds.; Springer Optimization and Its Applications; Springer: New York, NY, USA, 2011; Volume 45, pp. 305–335. ISBN 9781441916433.
81. Muthuraman, S.; Venkatesan, V.P. A Comprehensive Study on Hybrid Meta-Heuristic Approaches Used for Solving Combinatorial Optimization Problems. In Proceedings of the 2017 World Congress on Computing and Communication Technologies (WCCCT), Tiruchirappalli, India, 2–4 February 2017; pp. 185–190.
82. Urli, T. Hybrid Meta-Heuristics for Combinatorial Optimization. *Constraints* **2015**, *20*, 473. [CrossRef]
83. Pellerin, R.; Perrier, N.; Berthaut, F. A Survey of Hybrid Metaheuristics for the Resource-Constrained Project Scheduling Problem. *Eur. J. Oper. Res.* **2020**, *280*, 395–416. [CrossRef]
84. Birattari, M.; Paquete, L.; Stützle, T. *Classification of Metaheuristics and Design of Experiments for the Analysis of Components*; Darmstadt University of Technology: Darmstadt, Germany, November 2001.

85. Barr, R.S.; Golden, B.L.; Kelly, J.P.; Resende, M.G.C.; Stewart, W.R. Designing and Reporting on Computational Experiments with Heuristic Methods. *J. Heuristics* **1995**, *1*, 9–32. [CrossRef]
86. McGeoch, C.C.; Moret, B.M.E. How to Present a Paper on Experimental Work with Algorithms. *ACM SIGACT News* **1999**, *30*, 85–90. [CrossRef]
87. Huang, C.; Li, Y.; Yao, X. A Survey of Automatic Parameter Tuning Methods for Metaheuristics. *IEEE Trans. Evol. Comput.* **2020**, *24*, 201–216. [CrossRef]
88. Weinand, J.M.; Sörensen, K.; Segundo, P.S.; Kleinebrahm, M.; McKenna, R. Research Trends in Combinatorial Optimization. *Int. Trans. Oper. Res.* **2021**. [CrossRef]
89. 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*, 3764–3786. [CrossRef]
90. García-Martínez, C.; Gutiérrez, P.D.; 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*, 5573–5583. [CrossRef]
91. Črepinšek, M.; Liu, S.-H.; Mernik, L.; Mernik, M. Is a Comparison of Results Meaningful from the Inexact Replications of Computational Experiments? *Soft Comput.* **2016**, *20*, 223–235. [CrossRef]
92. Hussain, K.; Salleh, M.N.M.; Cheng, S.; Naseem, R. Common Benchmark Functions for Metaheuristic Evaluation: A Review. *Int. J. Inf. Vis.* **2017**, *1*, 218. [CrossRef]
93. Swan, J.; De Causmaecker, P.; Martin, S.; Özcan, E. A Re-characterization of Hyper-Heuristics. In *Recent Developments in Metaheuristics*; Amodeo, L., Talbi, E.-G., Yalaoui, F., Eds.; Operations Research/Computer Science Interfaces Series; Springer International Publishing: Cham, Switzerland, 2018; Volume 62, pp. 75–89. ISBN 9783319582528.