





Article

Global Optimisation through Hyper-Heuristics: Unfolding Population-Based Metaheuristics

Jorge M. Cruz-Duarte ^{1,†} , José C. Ortiz-Bayliss ^{1,†} , Ivan Amaya ^{1,*,†}  and Nelishia Pillay ^{2,†} 

¹ School of Engineering and Sciences, Tecnológico de Monterrey, Av. Eugenio Garza Sada 2501 Sur, Monterrey 64849, NL, Mexico; jorge.cruz@tec.mx (J.M.C.-D.); jcbayliss@tec.mx (J.C.O.-B.)

² Department of Computer Science, University of Pretoria, Lynnwood Rd, Hatfield, Pretoria 0083, South Africa; npillay@cs.up.ac.za

* Correspondence: iamaya2@tec.mx; Tel.: +52-(81)-8358-2000

† These authors contributed equally to this work.

Abstract: Optimisation has been with us since before the first humans opened their eyes to natural phenomena that inspire technological progress. Nowadays, it is quite hard to find a solver from the overpopulation of metaheuristics that properly deals with a given problem. This is even considered an additional problem. In this work, we propose a heuristic-based solver model for continuous optimisation problems by extending the existing concepts present in the literature. We name such solvers ‘unfolded’ metaheuristics (uMHs) since they comprise a heterogeneous sequence of simple heuristics obtained from delegating the control operator in the standard metaheuristic scheme to a high-level strategy. Therefore, we tackle the Metaheuristic Composition Optimisation Problem by tailoring a particular uMH that deals with a specific application. We prove the feasibility of this model via a two-fold experiment employing several continuous optimisation problems and a collection of diverse population-based operators with fixed dimensions from ten well-known metaheuristics in the literature. As a high-level strategy, we utilised a hyper-heuristic based on Simulated Annealing. Results demonstrate that our proposed approach represents a very reliable alternative with a low computational cost for tackling continuous optimisation problems with a tailored metaheuristic using a set of agents. We also study the implication of several parameters involved in the uMH model and their influence over the solver performance.

Keywords: metaheuristic; hyper-heuristic; optimisation; algorithm; unfolded metaheuristic

MSC: 65K10; 90C59; 68W50



Citation: Cruz-Duarte, J.M.; Ortiz-Bayliss, J.C.; Amaya, I.; Pillay, N. Global Optimisation through Hyper-Heuristics: Unfolding Population-Based Metaheuristics. *Appl. Sci.* **2021**, *11*, 5620. <https://doi.org/10.3390/app11125620>

Academic Editor: Peng-Yeng Yin

Received: 12 May 2021

Accepted: 09 June 2021

Published: 18 June 2021

Publisher’s Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.



Copyright: © 2021 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

Optimisation has been among us since forgotten times in different implicit human technologies. An unpteen quantity of problems and solutions have appeared chaotically in the current information age. Literature is so prolific that pretending to cover all existing approaches in this manuscript is an impossible task. Instead, we shall focus on one approach that has received a great deal of attention throughout previous years [1]: metaheuristics (MHs). The term covers a plethora of algorithms. Genetic Algorithms [2] and Simulated Annealing [3] are a couple of commonly found examples, which have existed for more than three decades. Others are younger, for example, Reflection-based Optimisation of the Stochastic Spiral Algorithm [4] and Archimedes Optimisation Algorithm [5]. With a rapid literature review, one can notice that innovation in MH has somehow stalled or branched out far from the characteristics that make these methods striking; i.e., hybrids and over-sophisticated approaches. Concerning the stagnation, we refer to the apparent lack of actual novel proposals in terms of mathematical or technical procedures [6]. It is customary for these methods to be accompanied by a corresponding metaphor. This may seem to differentiate two MHs even if they are mathematically equivalent. Hence, it is hard to assess

differences between proposals based on different metaphors. This impoverishes the study of metaheuristics, thus hindering the development of better methods as research efforts can become sparse. In turn, the field of metaheuristics becomes polluted with redundant information, all of them claiming novelty, and its evolution slows. Even so, the only way to detect similar approaches is to run extensive simulations or that, by chance, one researcher gets to know them really well. Still, with the increasing appearance of proposals, it becomes exceedingly difficult to notice this issue. In our previous work, we analysed this situation and proposed a model to formally study metaheuristics pushing away the metaphors and only concentrating on their operative core [7]. However, we noticed that the standard metaheuristic model only comprises up to three simple heuristics or search operators, a fact strongly limiting their exploration and exploitation capabilities and making them highly susceptible to the No-Free-Lunch (NFL) [8] theorem. Another relevant fact to take into account is the sensitivity of these methods to their inherent parameters. So, there may exist a simple alternative to study and propose enhancements in the metaheuristic field.

Concerning the selection of proper hyper-parameters of metaheuristics, it leads to a varying metaheuristic performance. For example, the Unified Particle Swarm Optimisation algorithm includes an explicit parameter for controlling the balance between exploration and exploitation of the search domain [9]. Other strategies include parameters of a categorical nature, leading to entirely different behaviours from one value to the next, e.g., the mutation and crossover schemes in Genetic Algorithms [10]. Because of this, some authors have sought ways for incorporating self-tuning capabilities into metaheuristics [11–14]. The previous comments should suffice to evidence that the parameter setting of a metaheuristic affects its performance. This complicates comparisons across different strategies since experimental conditions may not be the same or because the same experimental conditions can lead to a different performance level for each process. Despite the success of self-tuning approaches, the process remains a complex issue. This is mainly because the set of suitable parameters may change from one problem to another or even across instances of the same problem. As mentioned above, it leads us too far from the simplicity that characterises metaheuristics.

Tuning a metaheuristic is not the only way to improve its performance, and the research community has sought other ways for doing so. For example, Genetic Programming has been used to redefine existing solvers, as evidenced by Miranda et al. in [15]. In their work, the authors focused on the well-known Particle Swarm Optimisation and obtained improved results over a dataset of 60 standard continuous problems. Nonetheless, their approach is not easy to extend to other metaheuristics. Hence, a significant effort would be required to generalise it. Another path where research has focused is on ways to combine different solvers. This is known as hybridisation [16,17]. Bear in mind that hybrids may extend beyond the realm of metaheuristics. Thus, they may also integrate other approaches, e.g., Nelder–Mead Simplex [18]. Despite this, hybrids fall within the umbrella of metaheuristics. For instance, Hassan and Pillay used a Genetic Algorithm for automating the design of hybrid metaheuristics [19]. There are more examples that tune operators for a metaheuristic mould [20,21]. However, we do not delve into them for the sake of brevity. We must mention, nonetheless, that there is another enticing strategy that transcends the field of metaheuristics. Such an approach is commonly known as Hyper-Heuristics (HHs), and they strive to provide a higher-level solver [22]. They do this by using the available heuristics in different ways. Thus, HHs stand as a broad field with different variants [23]. Amongst them reside selection hyper-heuristics, which combine the available heuristics into a procedure for solving a given optimisation problem. This approach has been tested on the generation of metaheuristics based on two or more simple heuristics, where authors have achieved an excellent performance in particular problems [24]. However, the authors failed to provide a concrete mathematical model so that further systematic research on metaheuristics could be pursued.

In the previous lines, we have commented about the exacerbation of ‘novel’ metaheuristic proposals that lack a proper justification. Sörensen et al. warned about this issue

and its risks [6]. Still, remember that the NFL theorem reigns supreme, and so the quest for better solvers cannot halt. As such, a track worth pursuing deals with finding a way for combining different approaches, and different parameter settings, into a more robust solver. There are, indeed, some proposals for fusing metaheuristics, as we just mentioned. However, often times, such approaches are not easy to generalise. Moreover, there is no common ground regarding terminology and operations of metaheuristics, as most of them relate to the associated metaphor. Thus, it is essential to establish a standardised model, e.g., by using a generalised metaheuristic mould. We have taken a preliminary step in this direction by proposing a modelling scheme that allows merging Search Operators (SOs) from different metaheuristics [7]. Even so, it may be worthwhile to set up a branched model with different paths in such a way that different sets of SOs are applied to a given problem state to provide a more diverse behaviour. It may also be advantageous to extend the looping scheme so that the sequence of SOs can change throughout the solution process. This, however, is not a straightforward task with the previously proposed model. Therefore, in this work we extend upon the model and, thus, this manuscript offers five major contributions:

1. Provides a broader metaheuristic model that allows for more complex structures, such as those containing topologies or sub-populations;
2. Presents an overview of the effect of having long sequences of heterogeneous search operators that can be applied to continuous optimisation problems;
3. Shows the effect of three features (differentiability, separability, and unimodality) over the performance of the generated metaheuristics;
4. Studies the nature of search operators selected when solving problems with different characteristics and dimensionalities;
5. Stipulates a hyper-heuristic model for furnishing metaheuristics that can outperform standard ones in a majority of scenarios.

The remainder of this document is organised as follows. Section 2 presents background information about the fundamental concepts of our research. Then, Section 3 illustrates our proposed approach. Section 4 provides an overview of the testing that we carry out, while Section 5 shows the corresponding data. Finally, Section 6 wraps up the manuscript with the main insights and the paths for future work. In addition, we provide an appendix section containing all the detailed information about the search operators utilised in this work.

2. Theoretical Foundations

We begin this section by presenting the key concepts that support our proposed model. Be aware that although concepts may seem trivial, their meaning varies across disciplines. Thus, it is critical to have a well-settled background for avoiding misunderstandings and controversies. For example, the terms ‘heuristic’ and ‘metaheuristic’ could be swapped for a given problem, but they may have clearly differentiated meanings in other areas.

2.1. Optimisation

Optimisation is a routine procedure for all living beings. It refers to the decisions taken for reaching the best outcome in a given situation. From this point onward, we assume that an optimisation problem is such where an objective function must be minimised within a feasible continuous domain, where such elements can be described as follows.

Definition 1 (Minimisation problem). *Let $\mathfrak{X} \subseteq \mathfrak{S}$ be a feasible domain, since \mathfrak{S} is an arbitrary domain, and let $f(\vec{x})$ be an objective function to be minimised, known as cost function, defined on a nonempty domain $\mathfrak{X} \neq \emptyset$ such as $f(\vec{x}) : \mathfrak{X} \mapsto \mathbb{R}$. Thus, a minimisation problem represented with the tuple (\mathfrak{X}, f) is stated as*

$$\vec{x}_* = \underset{\vec{x} \in \mathfrak{X}}{\operatorname{argmin}} \{f(\vec{x})\}, \quad (1)$$

where $\vec{x}_* \in \mathfrak{X}$ is the optimal vector (or solution) that minimises the objective function within the feasible domain, i.e., $f(\vec{x}_*) \leq f(\vec{x}), \forall \vec{x} \in \mathfrak{X}$.

Remark 1 (Particular domain). *This feasible domain is somehow a general representation of any domain delimited by simple constraints, which can be extended to a more complex one by considering constraints of a different nature.*

Remark 2 (Maximisation problem). *In several applications, the objective function models the revenue, profit or utility of a process, so this function needs to be maximised. Let $\hat{f}(\vec{x})$ be the utility function such that we can state the optimisation problem with (1) by using a simple transformation such as $f(\vec{x}) = -\hat{f}(\vec{x})$.*

We deal with two kinds of problem domains, continuous and combinatorial, at different levels of abstraction. For the former, $\mathfrak{S} = \mathbb{R}^D$ and D stand for the number of dimensions that the problem contains. Under some conditions, such as when dealing with engineering applications, the user has prior knowledge about this domain. Moreover, D is usually fixed and represents the number of design variables. For the combinatorial problem, we say that $\mathfrak{S} = \mathfrak{H}^\omega$ with \mathfrak{H} as the heuristic space and ω as the cardinality of the sequence. Further details about this problem are provided below.

2.2. Heuristics

Heuristics can be used to create, evaluate or modify a ‘solution’ to a problem. It is customary to create these kinds of solvers based on existing knowledge about the problem, and they are fairly common in combinatorial optimisation [22], but seldom used in the continuous one [23]. In addition, the ideas presented by [22,23] can be used to define three groups of heuristics, depending on their pattern of actions: *low-level*, *mid-level*, and *high-level*. The first one is rather straightforward, as it contains one fixed action. The next one refers to an array of such actions. Finally, the last group is reserved to those approaches that can be reconfigured or that execute a variable number of instructions. In turn, each group relates to *simple heuristics*, *metaheuristics*, and *hyper-heuristics*, respectively. Some authors use these terms indistinctly, although correctly. The reason is that based on the surrounding conditions, such terms overlap. In this sense, one may call a ‘metaheuristic’ a ‘heuristic’, or even conclude that a ‘metaheuristic’ is given by a partial solution of a ‘hyper-heuristic’. In the long run they are all heuristics that operate with different specifications. In the subsequent sections, we describe briefly these concepts, which were fully detailed in our previous manuscript [7]. Furthermore, since the majority of heuristics operate over a population (i.e., a set of agents), we need to lay some concepts:

Definition 2 (Population). *Let $X(t)$ be a finite set of N candidate solutions for an optimisation problem given by \mathfrak{X} and $f(\vec{x})$ (cf. Definition 1) at time t in an iterative procedure, i.e., $X(t) = \{\vec{x}_1(t), \vec{x}_2(t), \dots, \vec{x}_N(t)\}$. Then, $\forall n \in \{1, \dots, N\}$, $\vec{x}_n(t) \in \mathfrak{X}$ denotes the n -th search agent position of, let us say, the population $X(t)$ of size N . For a single-agent approach ($N = 1$), the nomenclature is preserved and one can write $\vec{x}(t) = \vec{x}_1(t)$.*

Definition 3 (Best solution). *Let $Z(t)$ be an arbitrary set of candidate solutions, which can be designated as, e.g., the entire population $Z(t)$ is $X(t)$, the n -th neighbourhood $Z(t)$ is $Y_n(t)$, and the historical evolution of the n -th candidate $Z(t)$ is $\{\vec{x}_n(0), \vec{x}_n(1), \dots, \vec{x}_n(t)\}$. Therefore, let $\vec{x}_*(t) \in Z(t)$ be the best position from $Z(t)$, i.e., $\vec{x}_*(t) = \operatorname{arginf}\{f(Z(t))\}$.*

2.2.1. Simple Heuristics

Simple Heuristics (SHs) are the cornerstone of search techniques. They tackle the problem domain directly and can be of a *constructive* or *perturbative* nature. The former builds a solution from the ground up. The latter alters current ones [25]. Nonetheless, we also require a SH that validates if the process must continue. In simple words, this new category analyses the ongoing situation and selects the next SH to use. Perhaps it has no relevance as a standalone heuristic. Still, in the context of metaheuristics, the finaliser is the “master strategy” controlling the search procedure that many authors use to define the metaheuristics [26]. Bearing this in mind, we briefly define the simple heuristics and

their three categories. Consider that these concepts are applicable to an arbitrary problem domain \mathfrak{S} .

Definition 4 (Simple Heuristic). Let \mathfrak{H} be a set of simple heuristics, or heuristic space, with a composition operation $\circ : \mathfrak{H} \times \mathfrak{H} \mapsto \mathfrak{H}$. Let $\mathfrak{H}_i, \mathfrak{H}_o, \mathfrak{H}_f \subset \mathfrak{H}$ be subsets of heuristics that produces, modifies, and chooses between two operators, respectively.

Definition 5 (Initialiser). Let $h_i : \mathfrak{S} \mapsto \mathfrak{X} \subseteq \mathfrak{S}$ be a simple heuristic that generates a candidate solution $\vec{x} \in \mathfrak{X}$ within the feasible domain from scratch, i.e., $\vec{x} = h_i\{\mathfrak{X}\}$.

Definition 6 (Search Operator). Let $h_o : \mathfrak{X} \mapsto \mathfrak{X}$ be a simple heuristic that obtains a new position $\vec{x}(t+1) \in \mathfrak{X}$ from the current position $\vec{x}(t) \in \mathfrak{X}$, since t indicates the current iteration, i.e., $\vec{x}(t) = h_o\{\vec{x}\}$. A search operator mostly comprises two basic operations: perturbation and selection. Hence, let $h_p, h_s \in \mathfrak{H}_o$ be also simple heuristics that modify and update the current solution $\vec{x}(t)$, respectively; then $\vec{y} = h_p\{\vec{x}\}$ and $\vec{x}(t+1) = h_s\{\vec{y}\}$. These are called perturbator and selector. A perturbator always precedes a selector, so a search operator corresponds to $h_o = h_s \circ h_p$. In many implementations, selectors may require additional information to operate.

Definition 7 (Finaliser). Let $h_f : \mathfrak{X} \times \mathbb{Z}_2 \mapsto \mathfrak{H}$ be a simple heuristic that evaluates the current solution quality and chooses which search operator to apply. To do so, it uses information about the iterative procedure (e.g., the current solution, its fitness value, the current iteration, the previous candidate solutions, and other measurements) in a criteria function $c_f : (\mathbb{Z}_+, \mathbb{R}, \mathfrak{X}, \dots) \mapsto \mathbb{Z}_2$. Then, $h_f \in \mathfrak{H}_f$ is called a finaliser and is defined as

$$h_f(h_o)\{\vec{x}\} \triangleq \begin{cases} h_e\{\vec{x}\}, & \text{if } c_f(t, f, X, \dots) = 1, \\ h_f \circ h_o\{\vec{x}\}, & \text{otherwise,} \end{cases} \tag{2}$$

since h_o and h_e are two search operators selected according to c_f ; h_e is the identity operator, so $\vec{x} = h_e\{\vec{x}\}$.

Remark 3 (Criteria function). This function c_f ‘decides’ whether an additional iteration is required in the searching procedure. Hence, c_f checks one or many custom stopping (or convergence) criteria using information from the iterative process to make such a decision. Several criteria have been proposed in the literature for numerical methods in general, so their selection primarily depends on the problem application. The simplest one corresponds to a limited number of iterations t_{max} , such as $c_f(t) \triangleq H(t - t_{max})$, where H is the standard Heaviside function. Other complexes include a measure of computing budget, fitness improvement threshold, restricted stagnation counts, and so forth. Further information is provided in Appendix A.4.

It is worth noting that a selector and a finaliser are pretty similar in their functioning; the only difference is that the former deals with the problem domain, whilst the latter deals with the heuristic space.

2.2.2. Metaheuristics

A metaheuristic (MH) can be defined as a policy for handling a set of heuristics. This approach has become mainstream throughout the years [27,28]. In the following definitions, we describe how SHs serve as mathematical “building blocks” that can be merged into a MH. Do note that this liberates the process from the need of an inspirational metaphor and that it can describe existing or new metaheuristics.

Definition 8 (Metaheuristic). Let $MH : \mathfrak{S} \mapsto \mathfrak{X}$ be an iterative procedure called metaheuristic that renders an optimal solution \vec{x}_* for a given optimisation problem (\mathfrak{X}, f) , cf. Definition 1. A metaheuristic can be mathematically defined in terms of three basic components as shown,

$$MH_o \triangleq \langle h_i, h_o, h_f \rangle = h_f(h_o) \circ h_i, \tag{3}$$

since $h_i \in \mathfrak{S}_i$ is an initialiser, $h_f \in \mathfrak{S}_f$ is a finaliser, and $h_o \in \mathfrak{S}_o$ is a search operator, according to Definition 4. Regard that h_o represents one or more search operators combined under the composition closure, i.e., $h_o = h_1 \circ h_2 \circ \dots \circ h_\omega$. These operators can be described in an arrangement such as $\vec{h}_o = (h_1, h_2, \dots, h_\omega)^\top$, where ω corresponds to the metaheuristic cardinality, $\#MH_o = \#\vec{h}_o = \omega$.

2.2.3. Hyper-Heuristics

Hyper-Heuristics (HHs) are high-level methods that operate at a higher level of abstraction than metaheuristics and share several commonalities. Among their differences, the most distinctive one, at least for this work, is the domain where they operate. While an MH deals with continuous optimisation problems, an HH tackles a combinatorial problem domain because it is a “heuristic choosing heuristics” [29]. In the following lines, we define a hyper-heuristic using the previously described terms.

Definition 9 (Hyper-Heuristic). Let $\vec{h} \in H \subseteq \mathfrak{S}^\omega$ be a heuristic configuration from a feasible heuristic collection H within the heuristic space \mathfrak{S} (cf. Definition 4). Let $Q(\vec{h}|\mathfrak{X}, f) : H \times \mathfrak{X} \rightarrow \mathbb{R}_+$ be a metric that measures the performance of \vec{h} when it is applied on (\mathfrak{X}, f) , cf. Definition 1. Therefore, let the hyper-heuristic (HH) be a technique that solves

$$\vec{h}_* = \operatorname{argmax}_{\vec{h} \in H} \left\{ Q(\vec{h}|\mathfrak{X}, f) \right\}. \quad (4)$$

In other words, the HH searches for the optimal heuristic configuration \vec{h}_* that best approaches to the solution of (\mathfrak{X}, f) with the maximal performance $Q_{\max} = Q(\vec{h}_*|\mathfrak{X}, f)$.

Remark 4 (Heuristic Configuration). When performing a hyper-heuristic process, a heuristic configuration $\vec{h} \in \mathfrak{S}^\omega$ is a way of referring to a metaheuristic (cf. Definition 8).

Remark 5 (Performance Metric). There is no unique expression for determining the performance $Q(\vec{h}|\mathfrak{X}, f)$ since it depends on the desired requirements for the heuristic sequence \vec{h} . A numerical and practical way to assess this measurement, due to the stochastic nature of almost all the heuristic sequences, is to combine different statistics from several independent runs of \vec{h} over the same problem (\mathfrak{X}, f) . For example, the negative sum of median and interquartile range of the last fitness values $f(\vec{x}_{*,r})$ achieved in the runs, $r = 1, \dots, N_r$. More elaborated formulae could include other fitness statistics, time measurements, stagnation records, and so on.

2.3. Metaheuristic Composition Optimisation Problem

In the previous sections, crucial concepts of optimisation and heuristics (simple heuristics, metaheuristics, and hyper-heuristics) were introduced. Particularly, metaheuristics essentially comprise an initialiser, search operators (perturbators and selectors), and a finaliser. These components are usually combined using a manual design by a researcher or practitioner.

However, Qu et al. presented in [30] a taxonomy for the automated design of search algorithms. This taxonomy includes three categories: automated algorithm configuration, automated algorithm selection, and automated algorithm composition. The authors define the General Combinatorial Optimization Problem (GCOP) to involve optimising the combination of components of search algorithms to solve combinatorial optimisation problems. The GCOP is illustrated by applying selection perturbative hyper-heuristics to automate the composition of search algorithms for solving the capacitated vehicle routing and nurse rostering problems. When solving the GCOP, the perturbative heuristics are the components comprising the search algorithms.

In this work, we focus on the automated algorithm composition of a specific category of search algorithms, namely, metaheuristics. Hence, we define a version of the GCOP, i.e., the Metaheuristic Composition Optimization Problem (MCOP). Furthermore, MCOP applies to both combinatorial and continuous optimisation domains. The research presented

in this paper employs a selection perturbative hyper-heuristic (cf. Definition 9) to solve the MCOP for continuous optimisation. As in [30], the low-level heuristics are the components of the metaheuristic, and the optimisation aims to produce a solution \vec{x}_* by exploring a heuristic space of metaheuristic components instead of the solution space. Details of the proposed hyper-heuristic are presented in the next section.

3. Proposed Model

In this section, we describe our proposed model. Do keep in mind that we build it upon the concepts and terminology detailed in the previous section. In addition, we must make clear that our proposal stems from the idea of ‘unfolding’ a metaheuristic into a heuristic sequence found by solving the MCOP with a selection perturbative hyper-heuristic. What does ‘unfolding’ mean? Please take a look at Figure 1 where a metaheuristic is depicted in terms of simple heuristics (cf. Definition 8). Notice the feedback path (dashed stroke) in the metaheuristic diagram. The finaliser controls this path (cf. Definition 4). For each step, it either returns to the search operator (h_o) or concludes the procedure, leading to h_e . The signal-flow representation of metaheuristics is detailed in [7].

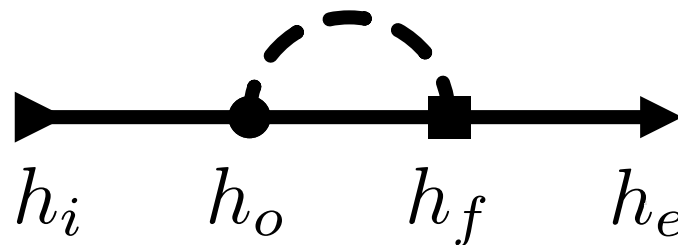


Figure 1. Signal-flow representation of a conventional metaheuristic comprising an initialiser h_i , a search operator h_o , and a finaliser h_f . h_e is the identity heuristic.

Now, consider a problem with a limited computational budget, which is common in real-world applications. In this case, the finaliser only anticipates the loop exit when an additional condition is met, e.g., a fitness value threshold. Nevertheless, if the finaliser role is delegated to a superior controller (say, an automated designer), we can unfold the metaheuristic scheme as a heuristic sequence. The reason: said controller not only decides when to stop applying a search operator but also which one (from a pool of search operators or heuristic space) to apply next. Figure 2 illustrates this idea in a simple chronological plot from top to bottom. This figure also renders the signal-flow diagram of the MH with an additional axis: the time or number of iterations. We included a dummy z axis for illustrative purposes; thus, the diagram is presented with a three-dimensional perspective (check the thumbnail cube with faces coloured according to the curves). Thus, Figure 2 depicts the unfolding timeline where the signal-flow diagram of a standard metaheuristic (bluish curves) is transformed into a homogeneous heuristic sequence (greenish curves). Note that this intermediate step in the timeline corresponds to a sequence of t implementations of the same search operator (h_o) over the population. This can also be considered as a heuristic sequence, although a trivial one. For that reason, we labelled it as a homogeneous heuristic sequence to avoid confusion. Subsequently, it is quite natural to think about not using the same operator over and over, so we can move to a more general sequence of different (non-exclusive) search operators. This signal-flow diagram at the bottom of Figure 2 displays a heterogeneous heuristic sequence (orangish box). We called it an unfolded metaheuristic (uMH). It is worth remarking that shorter sequences are desirable, as long as they preserve the performance level. Therefore, our proposed unfolded metaheuristic model can be framed into the MCOP described in Section 2.3. Beware that, in this case, the MCOP is targeted at solving continuous optimisation problems at the lowest level of abstraction.

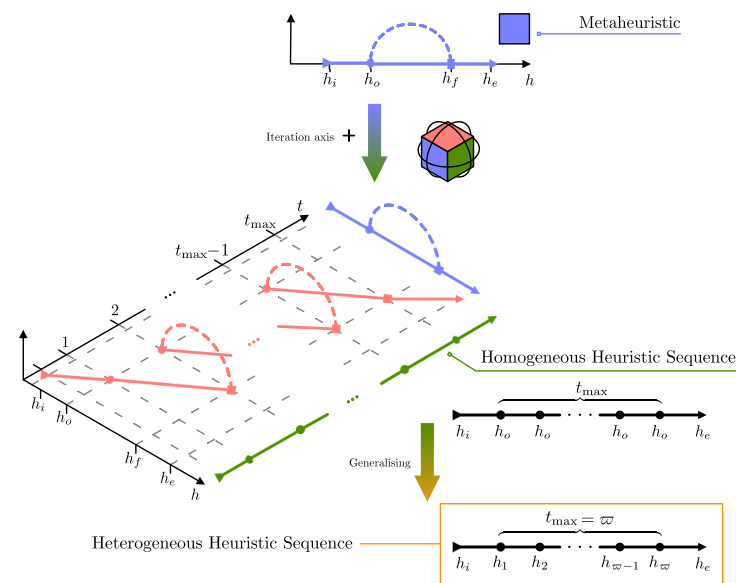


Figure 2. Graphical representation of a metaheuristic and how it becomes an unfolded metaheuristic (or a heterogeneous heuristic sequence) passing through a homogeneous heuristic sequence, which is obtained by considering an additional degree of freedom (the iteration axis) when observing the standard model.

4. Methodology

For this manuscript, we used Python 3.7 and a Dell Inc. PowerEdge R840 Rack Server with 16 Intel Xeon Gold 5122 CPUs @ 3.60 GHz, 125 GB RAM, and CentOS Linux release 7.6.1810-64 bit system for running our experiments. Moreover, we employed the framework CUSTOMHyS v1.0 to test the proposed model with ease. This open-access framework can be found at <https://github.com/jcrvz/customhys>, accessed on 10 May 2021, and it is documented in [31]. Thus, we needed to specify two domains at different levels of abstraction for testing and implementing the idea. Table 1 illustrates these domains, as well as their level of abstraction and role. The symbols and terminology from this table follow those from previous sections. Plus, each domain is detailed below.

At the lowest level, we selected a total of 107 benchmark functions as continuous optimisation problem domains. We considered a different number of dimensions D such as 2, 5, 10, 20, 30, 40, and 50. Plus, we categorised these problems by using their qualitative characteristics such as differentiability and unimodality in four duads of binary-encoded features, i.e., DU. With this information, we followed three manners of grouping the functions for analysing the obtained results. First, we regarded the whole set of functions, i.e., without discriminating categories, for procuring an overview of the behaviour. Then, we studied the functions grouped by feature duads: DU equals 11 (Differentiable and Unimodal problems), 10, 01, and 00 (Non-differentiable and Multimodal problems). Third, we performed a similar procedure as before, but only considering a marginal grouping, i.e., differentiability or unimodality. For example, we got differentiable and not differentiable problems neglecting the unimodality feature; a similar case applies for marginalising unimodality.

At the highest level, we utilised the population-based heuristics provided by CUSTOMHyS and also included others based on single-agent operations. Both collections belong to the heuristic space and are employed from a metaheuristic point of view. Although, the former is specialised population-based operations over a continuous problem domain with a fixed dimensionality, say, Population-based Fixed-dimension (PF) heuristics. Otherwise, the latter set corresponds to the single-agent operations, sometimes called actions, applied over a domain with variable dimensionality, say, Single-agent Variable-dimension (SV) heuristics. In this work, this particular domain is discrete, and the dimensionality is aliased to cardinality. These two sets of operators are closely related to the MCOP that we

solved. Table 2 summarises the simple heuristics collected for this work, and further details are provided in Appendix A. We divided them into three groups: SV perturbation, PF perturbation, and selection heuristics as aforementioned. The SV perturbators were designed conforming several works reported in the literature [32,33]. They were extracted from ten well-known metaheuristics: Random Search [34], Simulated Annealing [35], Genetic Algorithm [36], Cuckoo Search [37], Differential Evolution [38], Particle Swarm Optimisation [39], Firefly Algorithm [40], Stochastic Spiral Optimisation Algorithm [41], Central Force Optimisation [42], and Gravitational Search Algorithm [43]. We also included the random sample because it is the most straightforward manner of performing a search in an arbitrary domain. This simple heuristic also serves as the initialiser in all the implemented heuristic sequences. In this table, we present the name, variation parameters, and tuning parameters of these simple heuristics. The variation parameters concern those that drastically alter the nature of the operator. Meanwhile, tuning parameters hone the searching process. Besides, we also obtained the selectors from the metaheuristics mentioned above. It is worth mentioning that we also dedicated some words in Appendix A for documenting the finalisers commonly utilised in MH implementation.

Table 1. Description of the domains utilised in this work according to their level of abstraction and role. PF and SV stand Population-based Fixed-dimension and Single-agent Variable-dimension heuristics.

		Role	
		Solver	Problem
Level of abstraction	High	Search Space: PF Heuristics, $H_{PF} \subseteq \mathfrak{H}^3$ Individual: Metaheuristic, $MH_o \triangleq \langle \eta_i, \eta_o, \eta_f \rangle \in H_{SV}$ Procedure: $(\eta_f(\eta_o) \circ \eta_i)\{(H_{PF}, Q)\}$	Domain: Heuristics, $H_{PF} \subseteq \mathfrak{H}^\omega$ Individual: Sequence, $\vec{h} = (h_1, \dots, h_\omega)^\top \in H_{PF}$ Image: $Q(\vec{h} \mathfrak{X}) : H \times \mathfrak{X} \mapsto \mathbb{R}$ Objective: Maximisation, $\max_{\vec{h}} Q(\vec{h} \mathfrak{X}, f)$
	Low	Search Space: Continuous, $\mathfrak{X} \subseteq \mathbb{R}^D$ Individual: Unfolded metaheuristic, $\vec{h} = (h_1, \dots, h_\omega)^\top \in H_{PF}$ Procedure: $(h_\omega \circ \dots \circ h_1)\{(\mathfrak{X}, f)\}$	Domain: Continuous, $\mathfrak{X} \subseteq \mathbb{R}^D$ Individual: Position, $\vec{x} = (x_1, \dots, x_D)^\top \in \mathfrak{X}$ Image: Cost function, $f(\vec{x}) : \mathfrak{X} \mapsto \mathbb{R}$ Objective: Minimisation, $\min_{\vec{x}} f(\vec{x})$

Since we have described the problem and solution domains, we can now specify how they were employed to study the proposed model. Recall that we are studying ‘unfolded’ metaheuristics (uMHs) represented by heterogeneous heuristic sequences (cf. Figure 2). Thus, the number of iterations t_{max} is given by the sequence length or cardinality ω , i.e., $t_{max} = \omega$. In detail, we planned the experiments using a population size (N) of 30 for each unfolded metaheuristic with two cardinality limits. The first cardinality range is between 1 and 100, and the second one between 1 and 500. We selected these values to analyse the influence over the performance of a tight or a loose imposition in the sequence length. Due to this, we allowed SOs to appear more than once in the sequence. Thus, for controlling the actions that search over the heuristic space, we implemented the well-known Simulated Annealing (SA) using the Single-agent SV heuristics documented in Table 2 and Appendix A.2.1. Particularly, we utilised an initial dimensionless temperature (Θ_0) of 1.0, a minimal dimensionless temperature (Θ_{min}) of 10^{-6} , and a cooling rate (δ) of 10^{-3} . A maximal number of steps (s_{max}) of 200 was set for $\omega_{max} = 100$, and 500 steps were allowed for $\omega_{max} = 500$. We refer henceforth to these two configurations as Experiments 1 and 2, respectively. Subsequently, to assess the performance of each uMH (say, \vec{h}) when solving a given problem, we ran each candidate $N_r = 50$ times and recorded the last fitness values. Then, we estimated its performance using

$$Q(\vec{h}) \approx -(\text{med} + \text{iqr})(\{\forall \vec{x}_{*,r} \in X_* | f(\vec{x}_{*,r})\}), \tag{5}$$

where med and iqr are the median and interquartile range operators, respectively, applied to the fitness values $f(\vec{x}_{*,r})$. The symbol ‘ \approx ’ indicates that the estimation depends on the N_r samples X_* obtained from a stochastic process. Plus, the minus sign in (5) is because HH deals with a maximisation problem (cf. Definition 9), but the objective function $f(\vec{x})$, at the low-level domain, corresponds to a minimisation problem (cf. Definition 1). We chose this formula mainly because the median and the interquartile range are better descriptors in the presence of outliers. We also based this selection on several a priori results from previous works [31,44]. Lastly, and for the sake of clarity, we summarised the procedures performed for each problem in Pseudocode 1. In this, we refer to the SV heuristics as actions when they are described along with the PF heuristics to avoid confusion.

Table 2. Ten widespread metaheuristics segregated into their search operators. Parameter data are selected based on customary values from the literature.

Type	Simple Heuristic	Variation Parameters *	Tuning Parameters
SV Perturbation	Add	–	$i \sim \mathcal{U}\{1, \omega + 1\}$
	Add Many	–	$\theta \sim \mathcal{U}\{1, \omega_u - \omega\}$
	Remove	–	$i \sim \mathcal{U}\{1, \omega\}$
	Remove Many	–	$\theta \sim \mathcal{U}\{1, \omega - \omega_l\}$
	Shift	–	$i \sim \mathcal{U}\{1, \omega\}$
	Shift Locally	–	$\varepsilon \in \mathbb{R}_{++}, \mathcal{D} : \mathfrak{S} \times \mathfrak{S} \mapsto \mathbb{R}_+$
	Swap	–	$i, j \sim \mathcal{U}\{1, \omega\}$
	Restart	–	–
	Mirror	–	–
	Roll	–	–
	Roll Many	–	$k \sim \mathcal{U}\{1, \omega - 1\}$
	PF Perturbation	Random Sample (RX) †	$\vec{r} \ni r_i \sim \mathcal{U}(-1, 1)$
Random Search (RS)		$\vec{r} \ni r_i \sim \mathcal{U}(-1, 1)$	$\alpha \in [0, 1]$
Local Random Walk (RW)		$\vec{r} \ni r_i \sim \mathcal{U}(0, 1)$	$\alpha \in [0, 1], p \in [0, 1]$
Random Flight (RF)		$\vec{r} \ni r_i \sim \mathcal{L}(1.5)$	$\alpha \in [0, 1]$
Genetic Crossover (GC)		Pairing scheme, ‡ Crossover mechanism §	$m_p \in [0, 1]$
Genetic Mutation (GM)		$\vec{r} \ni r_i \sim \mathcal{U}(-1, 1)$	$\alpha \in [0, 1], p_e \in [0, 1], p_m \in [0, 1]$
Differential Mutation (DM)		Mutation scheme	$M \in \{1, 2, 3\}, \alpha_m \in [0, 3] \forall m \in \{0, M\}$
Particle Swarm Dynamic (PS)		Velocity approach, $\vec{r}_i \ni r_{i,j} \sim \mathcal{U}(0, 1) \forall i \in \{1, 2\}$	$\alpha_0 \in [0, 1], \phi_1, \phi_2 \in [0, 4], \kappa \in [0, 1]$
Firefly Dynamic (FD)		$\vec{r} \ni r_i \sim \mathcal{U}(-0.5, 0.5)$	$\alpha_0, \alpha_1 \in [0, 1], \alpha_2 \in [0, 1000]$
Spiral Dynamic (SD)		–	$r_0 \in [0, 1], \theta \in [0^\circ, 360^\circ], \sigma_r \in [0, 1]$
Central Force (CF)		–	$\alpha_0, \alpha_1 \in [0, 0.01], \alpha_2 \in [1, 2]$
Gravitational Search (GS)	–	$\alpha_0 \in [0, 1], \alpha_1 \in [0, 0.1]$	
Selection	Direct	–	–
	Greedy	–	–
	Probabilistic	–	$p_s \in [0, 1]$
	Metropolis	–	$k_B \in \mathbb{R}_+, \Theta(t) : Z_+ \mapsto \mathbb{R}_+$

* An alternative distribution function can be implemented instead of the default one, for example, the uniform $\mathcal{U}(0, 1)$, normal standard $\mathcal{N}(0, 1)$, and symmetric Lévy stable $\mathcal{L}(1.5)$ ones. † This simple heuristic is also used as initialiser. ‡ Available schema: random, rank weighting, roulette wheel, and tournament pairing. Tournament pairing requires two additional parameters such as $M_T \in \{1, 2, 3\}$ and $p_T \in]0, 1]$. § Possible mechanisms: single-point, two-points, uniform, blend, and linear crossover. Linear crossover requires two additional parameters such as $\beta_1, \beta_2 \in \mathbb{R}$.

Pseudocode 1 Hyper-heuristic based on Simulated Annealing (SAHH)

Input: Domain \mathcal{X} , objective function f , heuristic collections $H_{PF}, H_{SV} \subset \mathcal{H}$ (cf. Table 2), initialiser h_i , performance $Q(\vec{h}|\mathcal{X}, f)$, and population size N . Additional parameters: Θ_0, Θ_{\min} , and δ .

Output: Best heuristic sequence \vec{h}_*

```

1:  $\vec{h} \leftarrow \text{CHOOSERANDOMLY}(H_{PF}, P_{H_{PF}})$   $\triangleright P_{H_{PF}}(h)$  is the probability distribution of  $H_{PF}$ 
2: for  $r = \{1, \dots, N_r\}$  do  $\triangleright$  Repeat the subsequent evaluations  $N_r$  times
3:    $X_* \ni \vec{x}_{r,*} \leftarrow \text{EVALUATEMH}(\vec{h})$   $\triangleright$  Evaluate and record the candidate sequence
4: end for
5:  $\text{perf}(\vec{h}) \leftarrow Q(\vec{h}|\mathcal{X}, f)$  via Equation (5)  $\triangleright$  Estimate the performance metric
6:  $\vec{h}_* \leftarrow \vec{h}$   $\triangleright$  Initialise the best heuristic sequence
7:  $s \leftarrow 0$ , and  $\Theta \leftarrow \Theta_0$   $\triangleright$  Initialise the step counter and temperature
8: while  $(\Theta > \Theta_{\min})$  and  $(s \leq s_{\max})$  do
9:    $a \leftarrow \text{CHOOSEACTION}(H_{SV}, a, \omega)$   $\triangleright$  Choose an action at random from the action set
    $a \in H_{SV}$ 
10:   $\vec{h}_c \leftarrow a\{\vec{h}\}$   $\triangleright$  Obtain a neighbour sequence  $\vec{h}_c$  by applying  $a$  to  $\vec{h}$ 
11:  for  $r = \{1, \dots, N_r\}$  do  $\triangleright$  Repeat the subsequent evaluations  $N_r$  times
12:     $X_* \ni \vec{x}_{r,*} \leftarrow \text{EVALUATESEQUENCE}(\vec{h}_c)$   $\triangleright$  Evaluate and record the candidate
    sequence
13:  end for
14:   $\text{perf}(\vec{h}_c) \leftarrow Q(\vec{h}_c|\mathcal{X}, f)$  via Equation (5)  $\triangleright$  Estimate the performance metric
15:  if  $\mathcal{U}(0, 1) \leq \exp(-(\text{perf}(\vec{h}_c) - \text{perf}(\vec{h}))/\Theta)$  then  $\triangleright$  Apply Metropolis selection
16:     $\vec{h}, \text{perf}(\vec{h}) \leftarrow \vec{h}_c, \text{perf}(\vec{h}_c)$ 
17:  end if
18:  if  $\text{perf}(\vec{h}_c) < \text{perf}(\vec{h}_*)$  then  $\triangleright$  Apply Greedy selection
19:     $\vec{h}_*, \text{perf}(\vec{h}_*) \leftarrow \vec{h}_c, \text{perf}(\vec{h}_c)$ , and  $s \leftarrow 0$ 
20:  else
21:     $s \leftarrow s + 1$ 
22:  end if
23:   $\Theta \leftarrow \Theta(1 - \delta)$   $\triangleright$  Decrease the temperature
24: end while

25: procedure  $\text{EVALUATESEQUENCE}(\vec{h})$   $\triangleright$  Apply the composition:  $(h_\omega \circ \dots \circ h_1 \circ h_i)\{\mathcal{X}\}$ 
26:   $t \leftarrow 0$ 
27:   $X(t) \ni \vec{x}_n(t) \leftarrow h_i\{\mathcal{X}\}, \forall n \in \{1, \dots, N\}$   $\triangleright$  Initialise the population
28:   $F(t) \ni f_n(t) \leftarrow f(\vec{x}_n(t)), \forall n \in \{1, \dots, N\}$   $\triangleright$  Evaluate the population
29:   $\vec{x}_*(t) \leftarrow \vec{x}_k(t)$  since  $k = \text{arginf}\{F(t)\}$   $\triangleright$  Find the best global position by Direct
  selection
30:  for  $t = \{1, \dots, \omega\}$  do  $\triangleright t_{\max} = \omega = \#\vec{h}$ 
31:     $h_{p,t}, h_{s,t} \leftarrow h_t \in \vec{h}$   $\triangleright$  Read the perturbator and selector from the  $t$ -th search
    operator
32:     $X(t), F(t) \leftarrow h_{p,t}\{X(t)\}$   $\triangleright$  Apply the  $t$ -th perturbator
33:     $\vec{x}_*(t) \leftarrow h_{s,t}\{X(t)\}$   $\triangleright$  Update the best global position using the  $t$ -th selector
34:  end for
35:  return  $\vec{x}_*(t)$ 
36: end procedure

```

5. Results and Discussion

After carrying out the experiments described in the previous section, we study all the results using different approaches to validate our proposal. These approaches are organised as we now mention. We first discuss some punctual implementations to detail the unfolded metaheuristics designed by the SAHH approach. Then, we study the number of iterations from both low- and high-level approaches. They are the metaheuristic cardinality ω (or the number of search operators) and the hyper-heuristic steps s . Later, we discuss the nature of those operators selected by SAHH to be part of the customised uMHs. Right after analysing these characteristics, we evaluate the performance of the overall approach and the individual unfolded metaheuristics. Finally, we focus our attention on the time complexity of the methodologies that comprise this research.

5.1. Illustrative Selected Problems

First of all, we selected a problem from each category to illustrate the implementation of the proposed approach. Figure 3 shows how the SAHH procedure improves the heuristic sequence that it initially picked at random. To enhance the readability of these data, we plotted fitness order values, which were calculated using logarithm base 10 $\log f(\vec{x}_*)$, instead of the bare fitness values. Note that each violin in this figure corresponds to a heuristic sequence, where light-grey violins stand for the initial candidate sequence. In contrast, coloured ones correspond to those obtained from each experiment. In all the plots, the influence of the dimensionality over the problem domain is quite observable. This is expected, considering the established conditions (same population size and maximal cardinality). Regarding the maximal cardinality, it is straightforward to infer that unfolded metaheuristics (uMH) obtained from Experiment 2, which has a loose cardinality range, generally improve upon the corresponding uMHs from Experiment 1. An exception of this general improvement, which may not be the only one, can be seen when solving the problem Type I (Figure 3) with 20 dimensions. However, when the number of dimensions equals 50 for the same problem, it is easy to notice a remarkable difference between the two resulting metaheuristics. However, this does not imply that one achieved uMH is better than the others. Recall that each experiment has different conditions, and the resulting methodologies have distinctive properties, e.g., the type and the number of search operators (SOs) composing them. Nevertheless, an objective fact noticed at a glance from Figure 3 is the relatively compact shape of uMHs from Experiment 2. This is to be expected, chiefly because they were designed to have more SOs than those from Experiment 1. Thus, in raw words, they had more opportunities to ‘refine’ their search procedure. It is also worth remarking that even considering one design specification or the other, employing a particular approach may not grant a relevant benefit when dealing with some problems with exceptional ‘hardness’. An example of that can be observed in Figure 3d, where the uMHs tailored for the non-differentiable and multimodal (DU = 00) Schaffer N3 problem exhibit similar performances, close to that of the first uMH guessed by SAHH.

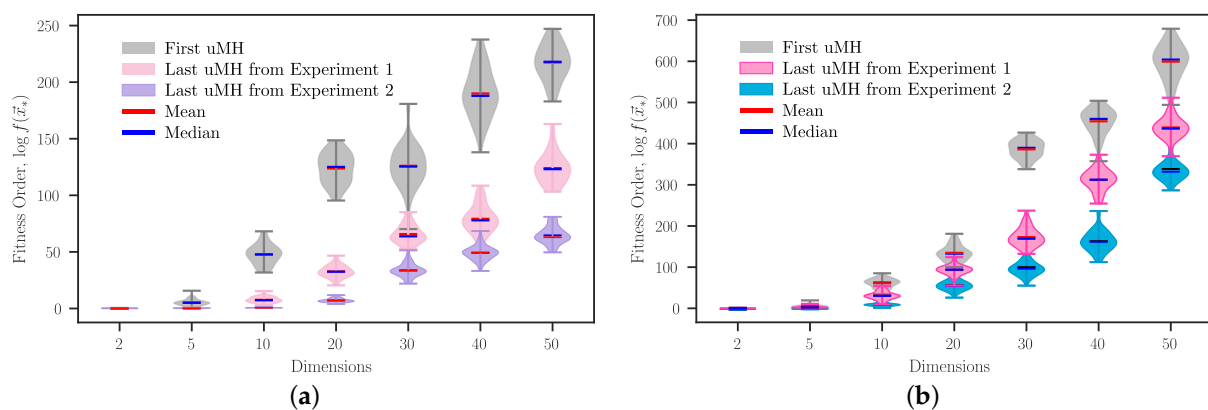


Figure 3. Cont.

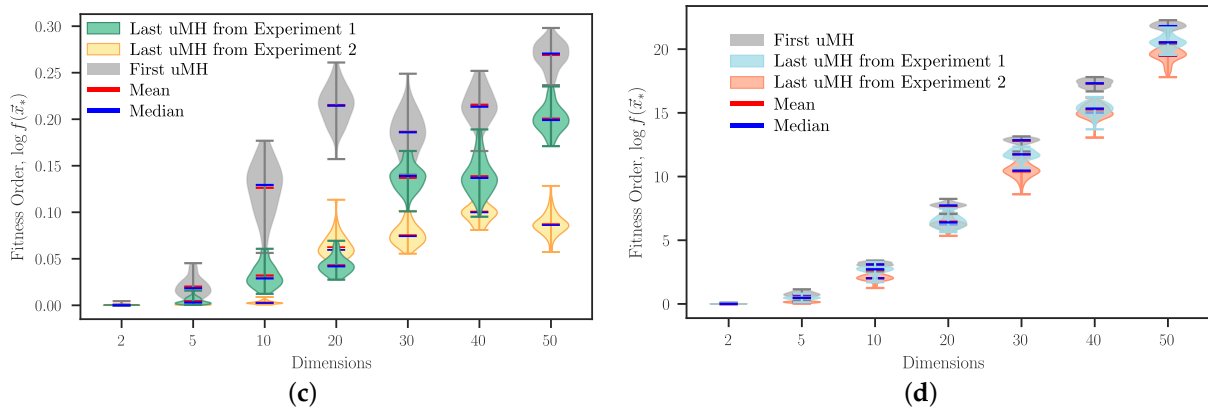


Figure 3. Fitness order variation obtained at the first (light-gray) and last (coloured) SAHH step for an illustrative problem for each category (DU) and considering different dimensionalities. This order is calculated by using the logarithm base 10 to each fitness value achieved per run, $\log f(\vec{x}_*)$. uMH stands for unfolded metaheuristic. (a) Schwefel (11). (b) Rastrigin (10). (c) Type I (01). (d) Schaffer N3 (00).

Table 3 complements the results from Figure 3. Concerning the performance value, it is evident that Experiment 2 promotes reaching better performing unfolded metaheuristics for most cases. However, this is not a general rule. An exception is observed in the performance value of the uMH designed for the 20-dimensional Type I problem (Figure 3). This is self-explained by remembering that SA is just a single-agent metaheuristic implemented as an HH. We have not run several implementations of the same HH problem as we did for evaluating the heuristic sequences because it exhibits a high computing cost. For this work, we employed SAHH as a well-established (stochastic) approach to design uMHs by solving the MCOP, which is considered ill-posed due to the presence of multiple solutions. Bear in mind that if there exists a unique solution, the NFL theorem is violated. This is an interesting discussion that is beyond our scope. Furthermore, it is worth noticing that there are some cases where quite similar performances can be rendered from dramatically different heuristic sequences, for example, those achieved for the Schaffer N3 problem with $D \geq 10$. In that case, it is preferable to use the setup from Experiment 1 or to enhance the HH solver. Plus, we can observe with ease that, on average, Experiment 1 and 2 obtained 92- and 460-cardinality uMHs, in HH procedures with an average of 121 and 283 steps, respectively. In terms of search operators, both implementations achieve methods close to the upper limit, but that was not the case of the required hyper-heuristic steps, at least in this illustrative example. A relevant fact to highlight from these data is the apparent absence of a cardinality (ω) and the increasing trend of heuristic steps (s) related to the dimensionality. It seems that our approach somehow deals with the dimensionality curse when solving the MCOP. However, this occurs at the high level, where this parameter can be considered as a design condition.

Table 3. Illustrative results from the selected problems in Figure 3. Columns correspond to the problem name, its binary-encoded category (DU), the achieved performance metric value (Perf.), unfolded metaheuristic cardinality (ω), and hyper-heuristic steps (s). Better performances are in bold face.

Problem	DU	Dim.	Experiment 1			Experiment 2		
			Perf.	ω	s	Perf.	ω	s
Schwefel	11	2	0.001	67	129	6.2×10^{-4}	494	323
		5	0.402	99	189	0.050	226	421
		10	11.025	100	187	0.669	483	216
		20	38.686	99	158	8.475	498	465
		30	74.324	94	180	37.957	499	273
		40	89.752	99	164	55.137	498	288
		50	137.846	90	99	70.506	499	441

Table 3. Cont.

Problem	DU	Dim.	Experiment 1			Experiment 2		
			Perf.	ω	s	Perf.	ω	s
Rastrigin	10	2	0.001	82	40	3.9×10^{-9}	378	72
		5	3.413	96	118	1.405	498	190
		10	40.683	100	142	14.999	500	180
		20	105.712	92	74	69.849	500	360
		30	196.954	89	192	117.694	266	380
		40	341.164	100	175	194.557	465	345
		50	463.786	94	162	358.056	490	404
Type I	01	2	1.3×10^{-4}	68	24	1.1×10^{-5}	311	127
		5	0.008	100	45	6.1×10^{-4}	500	203
		10	0.046	89	164	3.6×10^{-3}	499	145
		20	0.053	100	187	0.074	473	15
		30	0.155	99	15	0.087	499	491
		40	0.155	100	170	0.109	497	464
		50	0.214	87	20	0.094	497	392
Schaffer N3	00	2	0.004	100	108	1.6×10^{-3}	377	44
		5	0.657	97	78	0.236	483	247
		10	3.005	73	110	2.305	496	318
		20	6.989	87	190	6.578	471	455
		30	12.253	97	43	11.029	495	310
		40	15.862	94	170	15.697	500	202
		50	20.992	77	60	20.093	497	158

5.2. Low- and High-Level Iterations

After taking a smooth dive into the results using some punctual examples, we now analyse them from different perspectives. Figure 4 shows the normalised distribution of steps and cardinality when grouping this information by using the categories (DU) and the number of dimensions. It is pretty evident that for both parameters, ω and s , the maximal concurrence is near the upper limit (see the distributions in the margins of the diagonal plots). In this case, we used all the results obtained from Experiment 1, but one can notice a somewhat similar pattern from Experiment 2. From Figure 4a, we observe that uMHs with ω of about 30 (followed by 70) were more frequently designed for non-differentiable problems. At the same time, for the differentiable one, the cardinalities were more scattered along with the domain. Particularly, it is interesting that our approach found several ‘short’ heuristic sequences with up to 20 search operators for differentiable and multimodal (DU = 10) domains. This is remarkable because short procedures would be preferred in many practical implementations. Now, concerning the steps spent finding these uMHs, the distributions show relatively uniform shapes along with the limits. Using a different lens, Figure 4b, we encountered some inferences about ω and s when grouping the results by problem dimensionality. In general, it is worth noting the presence of two modes for the cardinality, near 30 and 70, when the two-dimensional problems are considered. Increasing dimensionality seems to slightly shift these modes to 20 and 60 when search domains reach 50 dimensions. This is an unexpected behaviour that can only be attributed to the capabilities of the proposed approach for dealing with high-dimensional problems. Plus, some particular facts to mention are as follows: for 5D domains, there are no low cardinalities; for 10D and 20D, uMHs of about 40 search operators are less frequent; and for 30D, the ω distribution is weighted to its extrema. Lastly, by looking at the steps required to design these metaheuristics, we can remark an evident transition from s values loaded about the first half of the studied range (i.e., from 1 to 100) for the 2D problems to a uniform-like distribution for those with 50D. In addition, it is somewhat understandable that for lower-dimensional domains, the hyper-heuristic implementation of Simulated

Annealing required ‘few’ steps for finding an optimal heuristic sequence. In such cases, we know the problem hardness is mainly related to its inherent characteristics, and the influence of the dimensionality *curse* is not so relevant. The following analysis is intended to attain more information about this discussion.

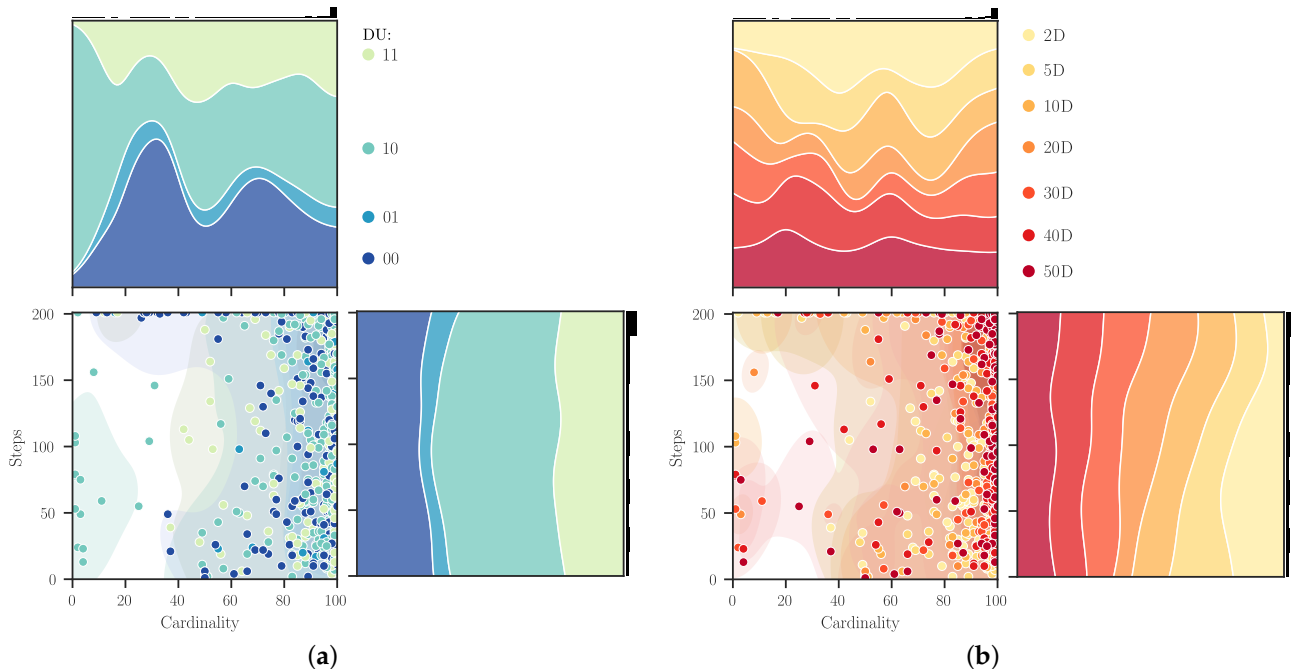


Figure 4. Scatter plot and marginal distribution for SAHH Steps and HH Cardinality grouped by Problem Characteristics (DU) and Dimensionality (Dim). (a) Categories (DU). (b) Dimensionality.

Figure 5 extends the results from Figure 4 by combining the dimensionalities and categories for distributions of both metaheuristic and hyper-heuristic iterations (i.e., cardinality and steps, respectively). Figure 5a shows that uMHs with more than 25 search operators are more prevalent when solving problems with extreme combinations of DU, i.e., 11 and 00. Although, there is an exception for each category. For the differentiable and unimodal problems, it appears when their dimensionality equals two, which is somewhat expected due to their inherent ‘simplicity’. For the non-differentiable and multimodal problems, one can notice low cardinality uMHs can be attributed to the SAHH solving procedure and other parameters worth studying in future works. Besides, ω distributions found for uMHs solving differentiable and multimodal functions are diverse and range almost every value for all dimensionalities. It is crucial to notice that for this category (DU = 10), our approach managed to find unfolded metaheuristics with up to 50 search operators with a remarkable frequency. The remaining category, non-differentiable and unimodal problems, renders quite interesting patterns, most of them showing compact distributions about the 100-cardinality uMHs. We may remark that this behaviour is chiefly led by the fact that such a characteristic combination is a bit unusual, so there are few problems within this category (DU = 10). In particular, our collection comprises 26, 45, 8, and 28 problems for the DU categories 11, 10, 01, and 00. Furthermore, Figure 5b exhibits the steps required by the hyper-heuristic to find a proper uMH for solving a problem from a given category and dimensionality. These distributions are, by far, smoother than those for the cardinality, which is somehow understandable because they belong to a higher level of abstraction (indirectly related to the problem domain). Except for the two-dimensional case, the steps required by SAHH to find the uMHs vary from a distribution that is highly concentrated in the lower limit into one akin to a uniform distribution. Again, we can explain the particular behaviour for 2D because most of the candidate heuristic sequences during the HH procedure were worth considering. Using an analogy with the Law of Supply and

Demand, we could state that there is more supply than demand. For the intermediate categories (say, 10 and 01), a slight transition from left- to right-hand loaded distributions is observable. Thus, dimensionality seems to directly influence the number of steps required for designing a solver that deals with problems from these categorical features. Lastly, for the non-differentiable and multimodal problems, we notice a reversed pattern with respect to distributions from the differentiable and unimodal problems. It is quite interesting to see how our approach manages to find uMHs for these functions using few steps with more frequency.

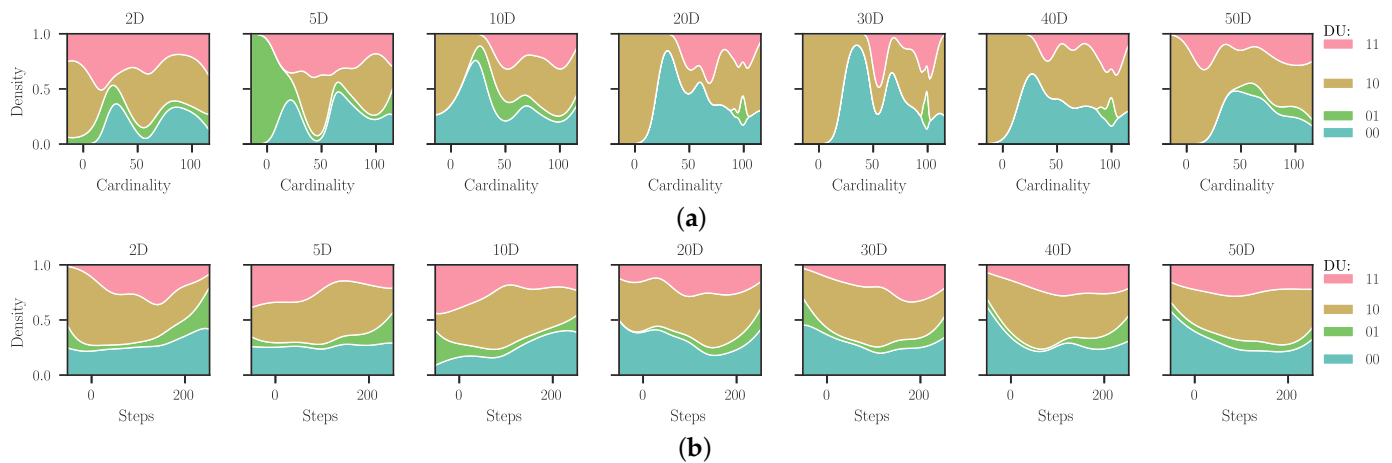


Figure 5. Cardinality and steps distributions varying the DU (differentiability and unimodality) characteristics and dimensionality for the unfolded metaheuristics designed in Experiment 1. (a) Cardinality. (b) Steps.

5.3. Nature of Search Operators

By this point, we have analysed the influence of categories and dimensionality in the unfolded metaheuristic cardinality and the required hyper-heuristic steps. Now, we delve into the nature of the search operators, particularly the perturbators, conforming these uMHs achieved via Experiment 1. We also disregard a detailed analysis from Experiment 2 due to their similarities with this case. Figure 6 presents bidimensional histograms involving the operator families employed in the metaheuristics for solving problem domains with different dimensionalities. We believe that Figure 6 is quite illustrative about the manner we perform this classification (see the thumbnail blocks in the upper centre). In the first block (upper-left corner), we observe the family distributions for ‘All’ the problems without considering their categorical features. We split all these results using the categorical characteristics, unimodality and differentiability, as we did above. The purple and magenta plots in the first row and column, respectively, correspond to the marginal cases where only one feature is considered, i.e., only differentiability (right blocks) or unimodality (bottom blocks). Greenish plots show the results for particular combinations of the differentiability and unimodality characteristics in a matrix-shape form. In general, we notice that our approach manages to deal with a given problem with certain conditions (characteristics and dimensionality) by arranging operators from one family or another to build its solver. There are some situations where a family is often preferred from others, as we discuss below. Nevertheless, it does not mean that a particular set of operators is the best for such a situation. We are just showing the nature of those operators selected for conforming an unfolded metaheuristic found throughout a hyper-heuristic search.

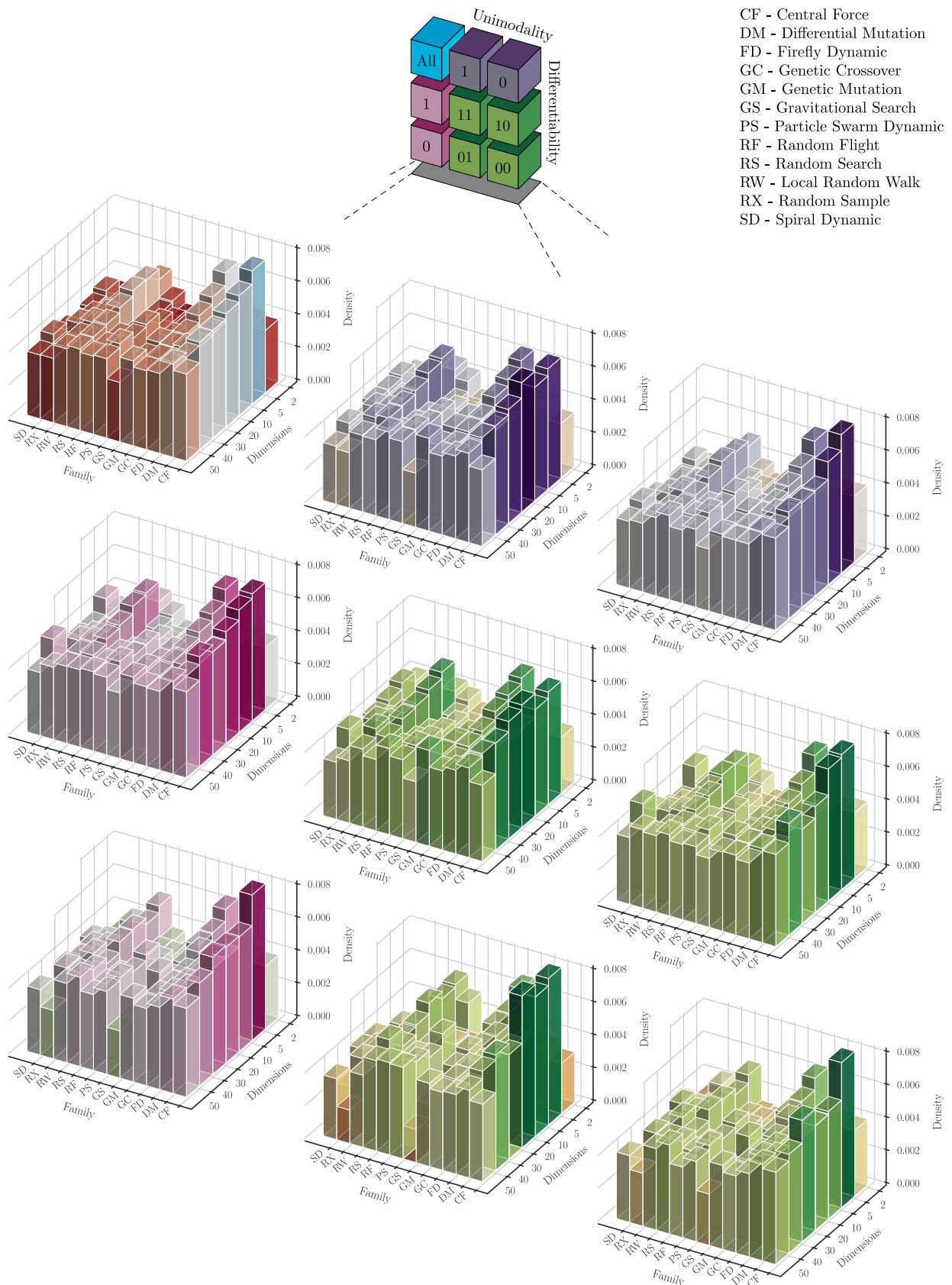


Figure 6. Frequency of the operator families used for each case category (differentiability and unimodality) and by varying the number of dimensions.

From the histogram for all the functions in Figure 6, it is easy to notice that operators from the Genetic Crossover (GC) and Random Search (RS) families were more frequently selected for dealing with two-dimensional problems. This fact is somehow expected due to the ‘simplicity’ of these problems where, in general, a simple stochastic approach would be enough to solve them. Obviously, GC is not such a simple approach, but we believe that its unstructured outcomes with respect to the other operators explain this behaviour. For greater dimensionalities, perturbators from the Central Force (CF) family became more relevant, but it seems to be attenuated when augmenting the number of dimensions. A similar effect is noticed with the GC perturbators and, slightly different, with the RS ones that show an increase in popularity around five and ten dimensions. On the contrary, we found that simple heuristics from Gravitational Search (GS) and Particle Swarm (PS) families seem, in a more notorious way, to be boosted up when dimensionality rises. One may explain such a fact with the ‘swarm’ interaction of the probes representing such operators, making them more suitable for higher-dimensional problems. Regarding the family distribution per dimension, we must remark that this becomes more uniform when increasing the problem dimensionality. Nevertheless, we note that mutation-based operators (from Genetic and Differential Mutation families) are oft-chosen to build heuristic sequences for solving 50-dimensional problems.

In general, the shape observed for the overall case, without categories, is preserved when grouping the results as mentioned. To better scrutinise these family distributions, the following lines are dedicated to remark those interesting situations that appear when grouping the results. For non-differentiable problems, operators based on random walks (e.g., RS) seem to be invariant in popularity, with a high-density value, concerning the dimensionality. This effect can be observed in detail at the blocks corresponding to categories 10 and 00. Likewise, the Gravitational Search (GS) family appears to be invariant to dimensionality and exhibits low-density values. This is also noticed in a reduced proportion for the Spiral Dynamic (SD) family. We can notice that GS heuristics start as the less popular ones when dealing with 2D domains for the marginally differentiable problems. Still, they gradually increase in density to a moderate value when the 50D problems are considered. This behaviour is also regarded in the block of marginally multimodal problems. It is crucial to remark that we observe a more uniform density distribution with respect to dimensions and families for this marginal classification. Nevertheless, it is quite obvious that this is mostly inherited from the differentiable and multimodal problems (DU = 10).

5.4. Performance Analysis

Once we have boarded the cardinality, required steps, and nature of the resulting unfolded metaheuristics, which are somehow similar for the two experimental setups, we proceed to study the most crucial feature of them: performance. Figure 7 presents the performance values for the resulting uMHs from the two experimental setups, when solving the problem domains with different characteristics and dimensionalities. In this figure, rows correspond to the combination of problem categories (differentiability and unimodality), and columns are the number of dimensions. It is pretty easy to notice that the conditions of Experiment 2 promote better performing uMHs than those of Experiment 1. Nevertheless, this is not a general rule. For example, when dealing with multimodal problem domains (say, DU = 10 and 00), the performance distribution for both experimental setups is virtually the same in the context of boxplots. One can explain such a fact with the dimensionality of these problems, making them ‘easy’ to solve. Naturally, if we have to choose a procedure to find an adequate solver, we would prefer to implement Experiment 1 because it requires fewer resources. In the other cases, it is evident that those additional requirements render notably better performing metaheuristics. Another point to emphasise is the precision of these performance values, with Experiment 2 allowing more concentrated values than the other.

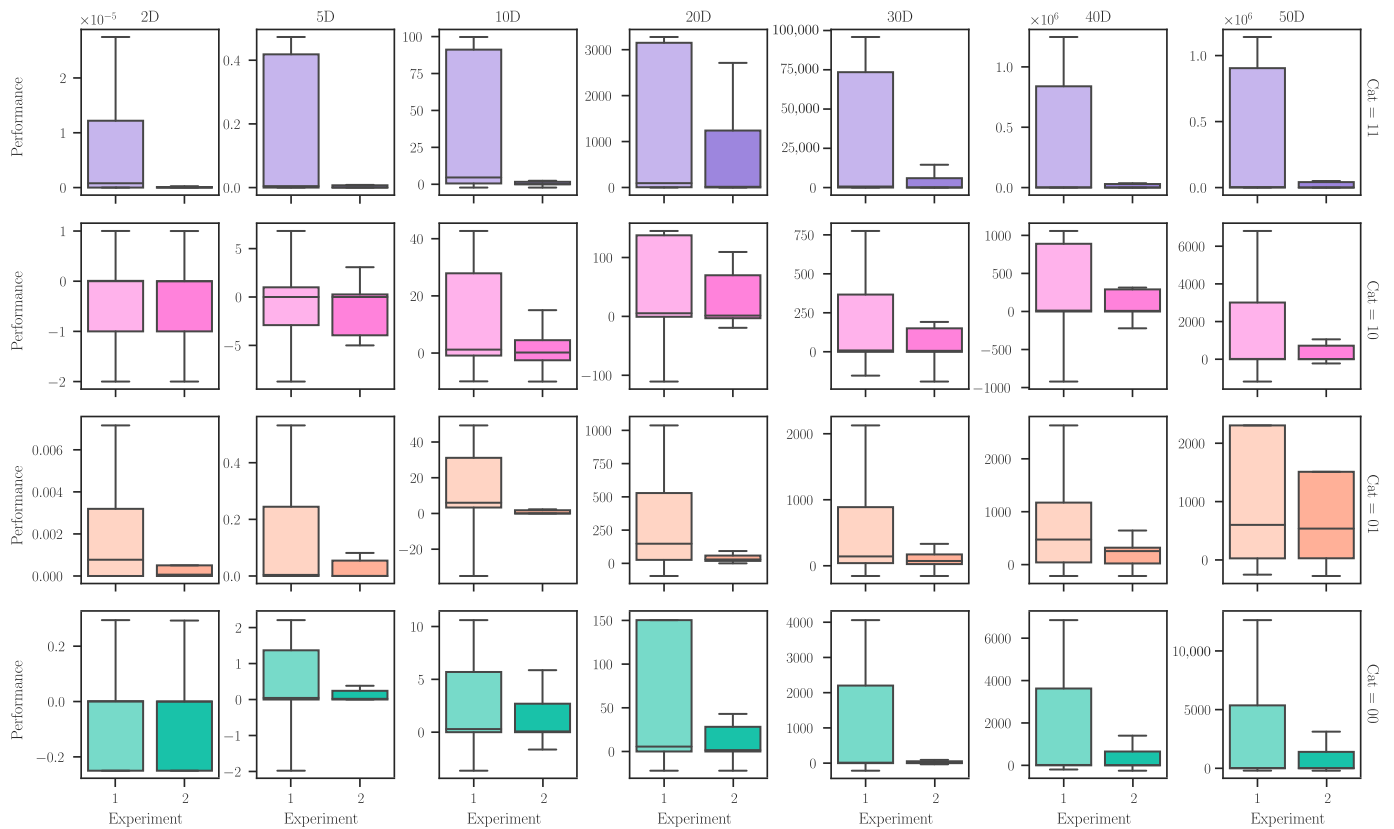


Figure 7. Performance values achieved by the unfolded metaheuristics when solving optimisation problems, with different characteristics and dimensionalities, via the proposed approach and using two experimental setups.

For our subsequent discussion, we compare the achieved unfolded metaheuristics against the collection of basic metaheuristics, which are also provided by the CUSTOMHyS framework [31], using the same performance metric. These basic methods are those used for extracting the simple heuristics but implemented with different default hyper-parameters. This gives us a total of 66 MHs. The first comparison is also the most straightforward: performance-to-performance. Thus, we calculated the Success Rate (SR) for an achieved unfolded metaheuristic (uMH) for a given problem domain (\mathcal{X}, f) compared against $(M = 66)$ basic MHs implemented in the same domain, such as,

$$SR(uMH) \triangleq \frac{1}{M} \sum_{i=1}^M H(Q(uMH|\mathcal{X}, f) - Q(MH_i|\mathcal{X}, f)), \quad (6)$$

where $H : \mathbb{R}^D \rightarrow \mathbb{Z}_2^D$ is the Heaviside step function and $Q(\vec{h}|\mathcal{X}, f)$ is the performance metric from (5). SR means a weighted counting of how many basic MHs were beaten (outperformed) by the designed uMH per problem and dimensionality. Although analysing the individual SR for each problem domain cannot be insightful, we aggregated the SR values from problems sharing the same category and dimensionality employing the average operation. Thus, we obtained an Average Success Rate (ASR) measurement that Figure 8 shows. With that in mind, we must notice that our approach, whichever experimental configuration, surmounts the performance of at least one basic metaheuristic in almost all problems (i.e., $ASR > 0$). This is crucial because even for an excellent solver for a given problem, we must know its limitations and advantages. However, employing the approach we are proposing in this manuscript, we as practitioners are able to find automatically a heuristic-based solver for any problem. It may not be the best one, but it is a good starting point for further enhancements. These words are supported with the non-zero values of ASR that Figure 8 depicts; it means that the uMH achieved by SAHH has beaten at least one basic MH. Moreover, we must emphasise that ASRs from

Experiment 1 were, in general, improved when we implemented the second setup. The most notorious scenarios where our approach surmounts the basic metaheuristics are the multimodal (DU = 10 and 00) problems. On the contrary, unimodal problem domains seem to be difficult rims, paradoxically, due to their ‘easiness’, for tackling these basic metaheuristics. In other words, the relatively low success rates do not imply that our approach has trouble solving these problems. It just means that many basic MHs can achieve good performance values.

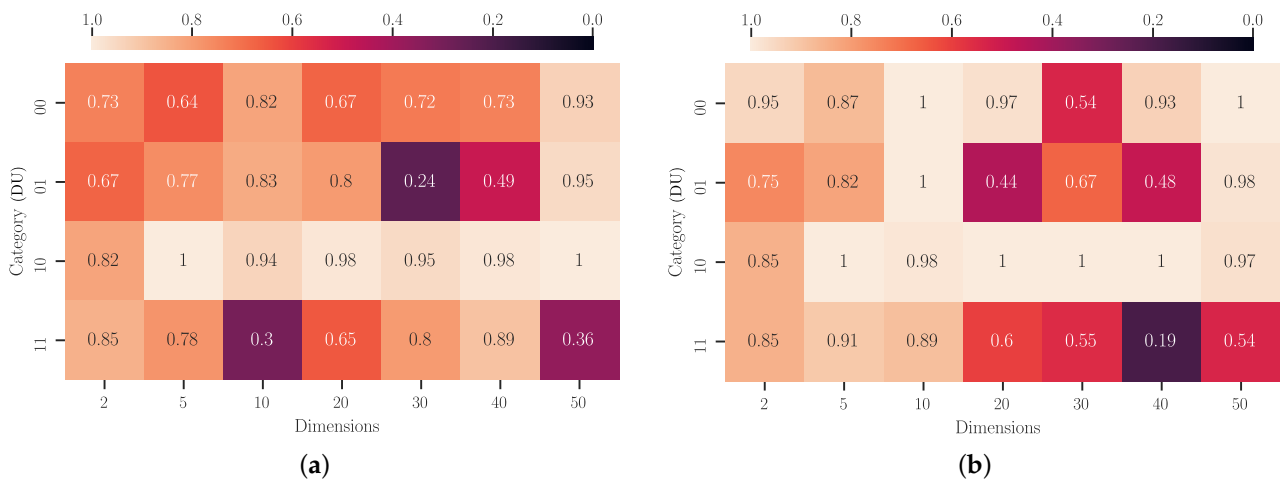


Figure 8. Average Success Rate achieved for the unfolded metaheuristic tailored using the proposed methodology compared against the basic metaheuristics. (a) Experiment 1. (b) Experiment 2.

Subsequently, we analyse the results by carrying out multiple comparisons using the pairwise (one-sided) Wilcoxon signed-rank test. Thus, we compute the true statistical significance for multiple pairwise comparisons such as $p = 1 - \prod_i (1 - p_i)$ by following the procedure and conditions given in [45,46]. We employed the performance value reached by the metaheuristics, for each problem and dimension via (5), as the measure of one run of the algorithm. Thus, we set the unfolded metaheuristic generated by SAHH as the control parameter and the basic metaheuristics as the other parameters. We look for the statistical significance of uMHs outperforming at least one of the basic MHs in a given problem. In doing so, we justify the fact that we do not require prior knowledge of which problems a particular MH is an excellent choice. Keeping this in mind, we state the null and alternative hypotheses as:

Hypothesis 0. (Null) *The unfolded metaheuristic, generated by the SAHH approach, performs equal to or worse than at least one basic metaheuristic with a significance level 0.05.*

Hypothesis 1. (Alternative) *The unfolded metaheuristic, generated by the SAHH approach, outperforms at least one basic metaheuristic with a significance level of 0.05.*

Figure 9 shows the p -values from this Wilcoxon’s analysis, and also grouped by category and dimensionality. At a general glance, we notice that it is possible ($p < 0.05$) to reject the null hypothesis H_0 for several scenarios in both experimental setups. Particularly, for those liaised with the multimodal domains, numerous basic metaheuristics are not suitable for dealing with them. It is also worth mentioning that the number of dimensions seems to increment the p -values determined for these scenarios. Nevertheless, it does not mean that SAHH generates poor-performing unfolded metaheuristics. We attribute this tendency to fail to reject the null hypothesis because these problems become harder to solve with any technique. It is also valuable to comment that Experiment 2 represents a better configuration than Experiment 1 for most problems with a dimensionality beyond five. On the opposite side, we can find those problem domains with a unimodal landscape where it

is hard to find a method incapable of solving them. Likewise, increasing the dimensionality has a similar effect as the one we just commented. However, we only fail to reject H_0 when regarding the nondifferentiable and unimodal problems in 2D and Experiment 2. This result is quite understandable because this setup proffers to generate longer heuristic sequences from longer HH searching procedures. It may also be considered a drawback of using a memoryless metaheuristic approach to deal with a hyper-heuristic problem. Naturally, if the practitioner decides to apply our approach for low-dimensional problems, we highly recommended a configuration similar to Experiment 1. Otherwise, Experiment 2 would suffice as a good starting point for dealing with more than twenty dimensions.

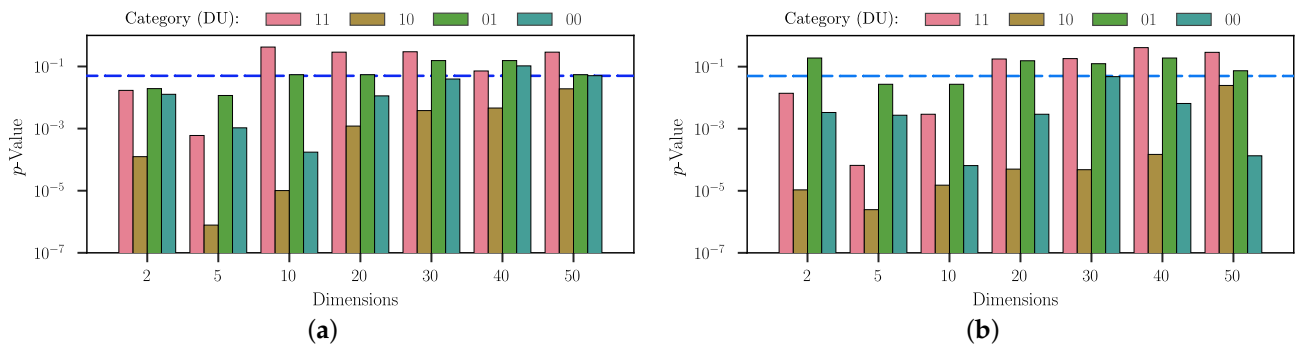


Figure 9. p -Values obtained from the pairwise (one-sided) Wilcoxon signed-rank test to the fitness values achieved for each problem of the categories (DU) and by considering different dimensions. Blue dashed stroke stands $p = 0.05$. (a) Experiment 1. (b) Experiment 2.

5.5. Time Complexity Analysis

All the terms mentioned below follow the nomenclature described in Sections 2 and 3. After analysing the nature of the resulting heterogeneous heuristic sequences, we are interested in studying the computing cost of our proposal. Keep in mind that once a uMH is designed for any problem, its implementation or further tuning for a relatively similar problem requires a low computing burden; it could be even lower than that required for a standard metaheuristic. Thus, we only focus our attention on the hyper-heuristic process for designing these uMHs. In that sense, SAHH has a time complexity similar to that of a regular metaheuristic, given by

$$\begin{aligned}
 T_{SAHH} &= T_{\bar{p}} + T_{\bar{f}} + T_{\bar{s}} + s_{\max} * (T_{\bar{p}} + T_{\bar{f}} + T_{\bar{s}}) \\
 &= \mathcal{O}\left(s_{\max} * \sup\left\{T_{\bar{p}}, T_{\bar{f}}, T_{\bar{s}}\right\}\right),
 \end{aligned}
 \tag{7}$$

since $T_{\bar{p}}$, $T_{\bar{f}}$, and $T_{\bar{s}}$ are also the computational costs of evaluating the perturbator, objective function, and the selector, respectively; and s_{\max} is the maximal number of steps carried out in this high-level search. Plus, SAHH is a single-agent algorithm, so selection, perturbation, and acceptance are actions considered to have constant time, i.e., $T_{\bar{p}} = T_{\bar{s}} = \mathcal{O}(1)$. Otherwise, the objective function at this high level stands for the performance metric described in (5), so its computing cost is given by $T_{\bar{f}} = \mathcal{O}(N_r * T_{uMH})$, where N_r is the number of repetitions, and T_{uMH} is the cost of implementing a candidate uMH. Naturally, the time for calculating the median and interquartile range is disregarded due to its magnitude. In summary, we can rewrite the SAHH time complexity as

$$T_{SAHH} = \mathcal{O}(N_r * s_{\max} * T_{uMH}).
 \tag{8}$$

Now, regarding the low level, the time complexity for an unfolded metaheuristic composed of up to ω_{\max} search operators yields

$$T_{uMH} = \mathcal{O}(\omega_{\max} * N * D * T_p),
 \tag{9}$$

since N is the population size, D is the dimensionality, and T_p is the computational cost of evaluating the perturbator. According to the well-known Big- \mathcal{O} definition, this value can be determined such as $T_p = \sup\{T_{p,1}, \dots, T_{p,\omega}\}$ for the uMH. It is easy to notice that the initialiser, finaliser, and selectors have an irrelevant complexity contribution. Besides, we focus on analysing the procedure and not the problem to solve, so we can adopt without loss of generality that the time complexity of evaluating the fitness is constant. Equation (9) can be employed to find the theoretical time complexity of any unfolded metaheuristic for some given conditions, for example, Experiments 1 or 2. In that case, we must use the worst case for T_p ; i.e., $T_p = \mathcal{O}(N^2)$ for the Genetic Crossover perturbator with Cost-based Roulette Wheel Pairing [47]. Thus, uMHs have a cubic time complexity with respect to the population size and a linear one with respect to the dimensionality, which is relatively high but affordable considering the context of designing solvers.

Keeping this in mind, we can express the time complexity of the overall approach T_{SAHH} by plugging (9) in (8), as shown,

$$T_{\text{SAHH}} = \mathcal{O}(N_r * s_{\max} * \omega_{\max} * D * N^3). \quad (10)$$

This expression gives us information about which components influence the time complexity of the overall approach. Notice that we considered the maximal number of search operators ω_{\max} instead of ω , which can be different from one uMH to another. From the above analysis, it is exciting to surmise that the running time scales linearly with the problem dimensionality and cubically with the population size; which is somehow manageable. The main reason is the high-level metaheuristic (i.e., SAHH) controlling the process, which performs one modification per step in a single element of the unfolded metaheuristic. For the scope of this research, SAHH constitutes a good prospect with a low computational burden.

Furthermore, we can use (9) to assess the order of operations required by the designed uMHs to tackle problems with different characteristics and dimensionalities. Thus, instead of the overall worst case of $\mathcal{O}(\omega_{\max} * T_p)$, we approximated this by accumulating the number of operations done for each simple heuristic from the uMH achieved in the SAHH procedure, i.e., $\mathcal{O}(\omega_{\max} * T_p) \approx \sum_{k=1}^{\omega} T_{p,k}$. Figure 10 displays this information as boxplots, where the effect of the problem dimensionality is evident for both experimental setups. First, it is interesting to note that Experiment 2 allows for more search steps, promoting a more refined heuristic sequence that solves a given problem. Such a fact translates into operation values more concentrated around a specific amount. This has meaningful implications in providing more confidence to the practitioner when implementing the approach proposed in this work. Although, it is not a secret that using this setup instead of the first one would have an additional computing burden. Plus, it is worth mentioning that the order of operations does not increment too dramatically, as we foresaw with the brief theoretical analysis. Likewise, as we expected, the difference between Experiments 1 and 2 (Figure 10a,b, respectively) does not constitute an abyss. Indeed, the overall value is 0.74 ± 0.41 with a mean value of 0.70. This is due to the low computational burden that SA imposes on the HH approach. The second setup has 300 and 400 units more in s_{\max} and ω_{\max} than those for the first setup. In general, we must notice that results for non-differentiable and unimodal functions represent the most expensive procedure for both experimental setups. It is also remarkable that many of the outliers from Figure 10 are mostly situated at data liaised with differentiable (DU = 11 and 10) problems. This is an unusual combination of characteristics since problems from this category exhibit diverse behaviours. Such a fact must be a matter to be considered in further works.

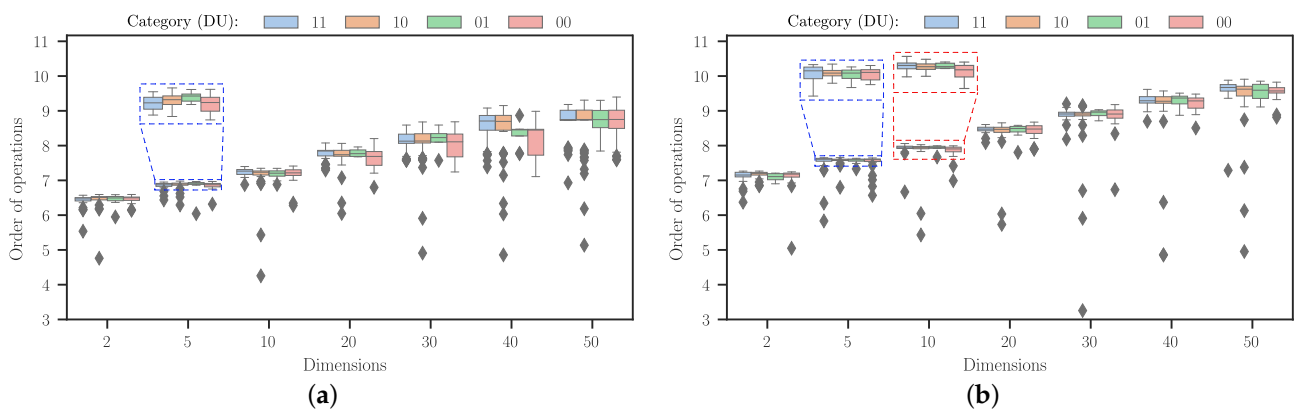


Figure 10. Order of the numerical operations carried out for designing unfolded metaheuristics when solving problem domains with different characteristics and dimensionalities. (a) Experiment 1. (b) Experiment 2.

6. Conclusions

This work proposed a new heuristic-based solver model for continuous optimisation problems. However, our proposal can be easily extended to other domains. Such a model, which we named ‘unfolded’ metaheuristic (uMH), comprises a heterogeneous sequence of simple heuristics. This is achieved by delegating the control mechanism in the standard metaheuristic scheme to a high-level strategy. Hence, this implies solving the Metaheuristic Composition Optimisation Problem (MCOP) to tailor a particular uMH that tackles a specific optimisation problem. We carried out a proof-of-concept by utilising the CUSTOMHyS framework, which is publicly available at <https://github.com/jcrvz/customhys>, accessed on 10 May 2021. Through CUSTOMHyS we analyse 107 continuous optimisation problems and a collection of 12 population-based fixed-dimension heuristics, extracted from ten well-known metaheuristics in the literature, to search in the problem domain. Thus, we specified the unfolded metaheuristic model and implemented a Hyper-Heuristic (HH) based on Simulated Annealing (SA), SAHH in short, to solve the MCOPs. For the SAHH, we also incorporated 11 single-agent variable-dimension heuristics to search in the heuristic domain. We considered two main experimental setups. The first one allowed uMHs of up to 100 search operators, with SAHH performing up to 200 steps. The second one considered up to 500 operators and steps. In the manuscript, we refer to the number of search operators in a uMH as its cardinality, ω .

From the results, we found that our proposed approach represents a very reliable alternative, with low computing cost, for designing optimisation population-based metaheuristics that solve continuous problems. To complement this claim, we shortly summarise several insights extracted from the experiments we carried out. First, we noticed that SAHH manages to find an unfolded metaheuristic, which deals with the optimisation problem, no matter the imposed design conditions. Naturally, some conditions affect the performance of the outcoming solvers. For example, the cardinality range does not directly affect the uMH performance, but when this range is relaxed, the uMH commonly renders more precise performance values. However, this has the drawback of increasing time complexity (or the number of operations) as a trade-off for a greater chance of balanced exploration and exploitation features. Besides, we found some leads indicating that the dimensionality affects the metaheuristic and hyper-heuristic iterations (i.e., cardinality and steps, respectively) on a scale related to the problem characteristics. The influence of dimensionality over problem hardness acts like a catalyst that amplifies the inherent difficulty given by its characteristics. In such cases, we noted that uMHs with greater cardinality only slightly outperform those with lower ω values, so we must recommend using the low-cost ones in practical implementations. Nevertheless, we have not detected an abysmal difference between the unfolded metaheuristics obtained from the studied setups in the number of operations, i.e., about 0.70 of the median value. This is a great feature due to the heuristic nature of the solvers that our approach generates.

We also studied the nature of the search operators that are automatically selected by SAHH when building uMHs. We remark that SAHH often chooses operators from the Genetic Crossover and Random Search families for two-dimensional problems. However, their frequency seems to be attenuated when increasing the number of dimensions. A similar case was observed with simple heuristics of the Central Force nature, but starting with 5D problems and an evident greater dominance. Conversely, we regard operators from the Gravitational Search and Particle Swarm families correspond to the less frequent ones at the 2D mark, although they become relevant during the dimensionality increment. We attribute this effect to the nature of these families, related to multi-probes intelligent interactions, which performs better on broader search domains. In the end, for the greatest dimensionality, we observed a pretty uniform distribution from almost all the families of search operators. Therefore, we infer that operators slightly sophisticated to the random ones are enough to fulfil the solver requirement for few dimensions. More nature-diverse operators are needed for many variables.

Lastly, we carried out a similar experiment using the metaheuristics selected for extracting the search operators used to construct the heuristic space employed by SAHH; we called them basic MHs. We found that our uMHs surmount the average performance of at least one basic metaheuristic for the same problem domain. Such a fact has a tremendous repercussion in the context of the No-Free-Lunch (NFL) theorem, mainly because the practitioner does not have to know which is the excellent metaheuristic for a given problem. Instead, s/he only has to use our approach, and it will indeed find a good one with some particular requirements. Bear in mind that this work does not pretend to violate the NFL theorem. Instead, we are using it in our favour to propose suitable solvers.

To the best of our knowledge, this unfolded metaheuristic model has not been previously proposed in the literature. According to our results, it is a model worth considering in practical implementations because it seems to be a very reliable and low-cost computing approach for solving optimisation problems. We know that our model is not a panacea. There are several issues to deal with for further understanding and improvement; some are now commented. To alleviate the influence of dimensionality and problem characteristics, we plan to loosen the constraints imposed on the cardinality and steps, and include other relevant parameters, such as the population size. In the same way, we believe that improving the hyper-heuristic algorithm (i.e., SAHH) must increase the overall performance of the resulting uMHs. An alternative and faster approach requires considering a non-heuristic-based algorithm such as a Machine Learning methodology. Plus, we are going to consider additional benchmark problems, including those utilised in algorithm competitions, e.g., CEC and GECCO. Lastly, we plan to implement this approach in a practical engineering scenario to test its feasibility to achieve low-budget heuristic sequences. We considered it a relevant topic for future works.

Author Contributions: Conceptualisation, J.M.C.-D., I.A., J.C.O.-B., and N.P.; methodology and software, J.M.C.-D., I.A., J.C.O.-B., and N.P.; validation, J.M.C.-D., I.A., J.C.O.-B., and N.P.; formal analysis and investigation, J.M.C.-D., I.A., J.C.O.-B., and N.P.; resources, J.M.C.-D., I.A., J.C.O.-B., and N.P.; data curation, J.M.C.-D., I.A., J.C.O.-B., and N.P.; writing—original draft preparation, J.M.C.-D., I.A., J.C.O.-B., and N.P.; writing—review and editing, J.M.C.-D., I.A., J.C.O.-B., and N.P.; visualisation, J.M.C.-D., I.A., J.C.O.-B., and N.P.; supervision, J.M.C.-D., I.A., J.C.O.-B., and N.P.; project administration, J.M.C.-D., I.A., J.C.O.-B., and N.P.; funding acquisition, I.A. All authors have read and agreed to the published version of the manuscript.

Funding: The research was supported by the CONACyT Basic Science Project with grant number 287479.

Conflicts of Interest: The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results.

Abbreviations

The following abbreviations are used in this manuscript:

CF	Central Force
CFO	Central Force Optimisation
CS	Cuckoo Search
DC	Differential Crossover
DE	Differential Evolution
DM	Differential Mutation
FA	Firefly Algorithm
FD	Firefly Dynamic
GA	Genetic Algorithm
GC	Genetic Crossover
GM	Genetic Mutation
GS	Gravitational Search
GSA	Gravitational Search Algorithm
HH	Hyper-Heuristic
NFL	No-Free-Lunch
MCOP	Metaheuristic Composition Optimisation Problem
MH	Metaheuristic
PS	Particle Swarm Dynamic
PSO	Particle Swarm Optimisation
RF	Random Flight
RS	Random Search
RW	Local Random Walk
RX	Random Sample
SA	Simulated Annealing
SAHH	Hyper-Heuristic based on Simulated Annealing
SD	Spiral Dynamic
SH	Simple Heuristic
SO	Search Operator
SSOA	Stochastic Spiral Optimisation Algorithm
uMH	Unfolded Metaheuristic

Appendix A. Simple Heuristics

In this section, we detail the simple heuristics extracted from the following 10 well-known MHs: Random Search (RS) [48], Simulated Annealing (SA) [3], Genetic Algorithm (GA) [49], Cuckoo Search (CS) [50], Differential Evolution (DE) [38], Particle Swarm Optimisation (PSO) [39,51], Firefly Algorithm (FA) [52], Stochastic Spiral Optimisation Algorithm (SSOA) [41], Central Force Optimisation (CFO) [53], and Gravitational Search Algorithm (GSA) [43]. These heuristics are presented and organised using the same nomenclature and terminology of Section 2. Furthermore, they also correspond to those listed in Table 2.

Appendix A.1. Initialisers

This simple heuristics are pretty simple in their implementation because they are in charge of drawing the initial solution(s) for a subsequent searching process. In many practical implementations, the initialisers are just random samplers using a well-defined feasible domain. In others words, this work is performed by a practitioner with certain degree of expertise on the problem under solution. Notwithstanding, we shortly describe the widest utilised initialiser in several (almost all) heuristic-based applications, which employ either one or multiple search agents, as follows.

Random Sample (RX): An initial solution $\vec{x}_n(t)$ is obtained through a random sample operation such as $\vec{x}_n(t) = \vec{r}$, where $\vec{r} \in \mathfrak{S}$ is a vector of i.i.d. random numbers with uniform distribution between the boundaries of the feasible domain. If it is a continuous feasible domain between -1 and 1 , $\mathfrak{X} = [-1, 1]^D \subset \mathfrak{S} = \mathbb{R}^D$, then $\vec{r} \ni r_i \sim \mathcal{U}(-1, 1)$. If it is a

discrete feasible domain between 1 and $m (\gg 1)$, $\mathfrak{X} = \{\forall y_i \in \bar{y}, \bar{y} \in \mathfrak{S} | 1 \leq y_i \leq m\} \subset \mathfrak{S} = \mathbb{Z}^D$, hence $\bar{r} \ni r_i \sim \mathcal{U}\{1, m\}$. Other feasible domain can also be considered, for example, the mixed continuous-discrete ones.

Notice that this initialiser, as well as others, can be implemented for either a single-agent or population-based search procedure.

Appendix A.2. Perturbators

In this section, we describe several perturbators found in the selected metaheuristics described at the beginning of Appendix A. Somewhat similar to initialisers, many perturbators can be generalised as population-based procedures, but one may encounter some interesting exceptions. One of these, closely related to this work, are the single-agent perturbators with variable length or dimension. These are found with ease in several ‘classic’ heuristics dealing with combinatorial problems, where the candidate solution is represented by a set of elements (say, item, values, weights, and so forth) which can vary in length while solving the problem. Therefore, we grouped the extracted perturbators into two main groups: single-agent variable-dimension and population-based fixed-dimension perturbators. Consider that these perturbators can be implemented in both integer and continuous domains. Naturally, for the latter, one must take some additional precautions into account. Other groups from the possible combinations of these characteristics would be worth investigating, but they are out of our scope.

Appendix A.2.1. Single-Agent Variable-Dimension Perturbators

For these simple heuristics, also named as *rules* or *actions* in the literature, we consider an arbitrary problem domain \mathfrak{S} . Nevertheless, bear in mind that we use these heuristics in the context of a combinatorial domain such as the heuristic space, $\mathfrak{S} = \mathfrak{H}$. Recall that $\bar{x}(t)$ and \bar{y} correspond to the current and candidate solutions, respectively, and ω is the cardinality of $\bar{x}(t)$, $\omega = \#\bar{x}$. This cardinality or dimensionality is finite and delimited by a lower and upper value such as $\omega_l \leq \omega \leq \omega_u$. In the following lines, we briefly describe the actions regarded for this work.

Add: It inserts randomly chosen element $y_i \in \mathfrak{S}$ at a random l.h.s. position $i \sim \mathcal{U}\{1, \omega + 1\}$ into $\bar{x}(t)$. Thus, the candidate sequence will be $\bar{y} = (x_1, \dots, x_{i-1}, y_i, x_{i+1}, \dots, x_\omega)^\top$ with $\#\bar{y} = \omega + 1$.

Add Many: It performs the **Add** operation ϑ times since this value is chosen at random between 1 and the remainder of ω to ω_u , $\vartheta \sim \mathcal{U}\{1, \omega_u - \omega\}$, where ω_u is the maximal cardinality. After doing so, the cardinality of the candidate solution will be $\#\bar{y} = \omega + \vartheta$.

Remove: It discards an element x_i from a random position $i \sim \mathcal{U}\{1, \omega\}$ of $\bar{x}(t)$. Thus, the candidate sequence will be $\bar{y} = (x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_\omega)^\top$ with $\#\bar{y} = \omega - 1$.

Remove Many: It performs the **Remove** operation ϑ times since this value is chosen at random between 1 and the remainder of ω_l to ω , $\vartheta \sim \mathcal{U}\{1, \omega - \omega_l\}$, where ω_l is the minimal cardinality. After doing so, the cardinality of the candidate solution will be $\#\bar{y} = \omega - \vartheta$.

Shift: It selects a random position $i \sim \mathcal{U}\{1, \omega\}$ and changes the corresponding element $x_i \in \bar{x}(t)$ with another chosen at random from the feasible domain \mathfrak{S} , say $y_j \in \mathfrak{S}$. Therefore, the candidate sequence will be $\bar{y} = (x_1, \dots, x_{i-1}, y_j, x_{i+1}, \dots, x_\omega)^\top$ with $\#\bar{y} = \omega$.

Shift Locally: It is similar to the **Shift** action, but the element $y_j \in \mathfrak{S}$ is chosen, also in the feasible domain \mathfrak{S} , but around the original one $x_i \in \bar{x}(t)$, i.e., $y_j \in \{\forall y_k \in \mathfrak{S} | \mathcal{D}(x_i, y_k) < \varepsilon\}$ where $\mathcal{D} : \mathfrak{S} \times \mathfrak{S} \mapsto \mathbb{R}_+$ is a closeness metric and $\varepsilon \in \mathbb{R}_+$ is a small value.

Swap: It interchanges elements $x_i, x_j \in \bar{x}(t)$ at two randomly selected locations, $i, j \sim \mathcal{U}\{1, \omega\}$. Hence, the candidate sequence is given by $\bar{y} = (x_1, \dots, x_{i-1}, x_j, \dots, x_{j-1}, x_i, \dots, x_\omega)^\top$ with $\#\bar{y} = \omega$.

Restart: It ignores the current sequence $\vec{x}(t)$ and randomly selects a new one \vec{y} with the same cardinality $\omega = \#\vec{x}(t) = \#\vec{y}$. This action is a particular case of the **Random sample** initialiser, but utilised as a perturbator.

Mirror: It reorganises $\vec{x}(t)$ in reverse order. Thus, the new sequence will be $\vec{y} = (x_\omega, x_{\omega-1}, \dots, x_1)^\top$, also preserving the cardinality.

Roll: It moves all the elements in $\vec{x}(t)$ one position back or forward at random (i.e., Bernoulli trial with $p = 0.5$) and with cyclic indexing. Thus, the candidate sequence will be $\vec{y} = (x_{1\pm 1}, x_{2\pm 1}, \dots, x_{\omega\pm 1})^\top$, since $x_0 \triangleq x_\omega$ and $x_{\omega+1} \triangleq x_1$.

Roll Many: It is similar to the **Roll** action but $k \geq 1$ positions are randomly chosen to displace, i.e., $k \sim \mathcal{U}\{1, \omega - 1\}$.

It is worth mentioning that $\vec{x}(t)$ and \vec{y} could be expressed as sets (x and y), but the vector notation is preferred for notation standardisation. Besides, notice that we use ω instead of D to make the distinction between a heuristic sequence and a position within a continuous problem domain, respectively. This clarification is useful to avoid misunderstandings in Section 3.

Appendix A.2.2. Population-Based Fixed-Dimension Perturbators

These simple heuristics correspond to those procedures that can deal with multiple search points but in a problem domain with fixed dimensionality. In our particular interest, these perturbators are closely related to the population-based metaheuristics widely utilised in continuous optimisation problems. Thus, they are described mostly in terms of an arbitrary domain \mathfrak{S} , but without loss of generality some expressions are written assuming $\mathfrak{S} = \mathbb{R}^D$, which is the domain we used for them. We then consider that the feasible domain \mathfrak{X} is given by a hyper-cube with boundaries from -1 to 1 per dimension, $\mathfrak{X} = [-1, 1]^D$. Bearing this in mind, the collected perturbators are presented for determining the n -th candidate solution $\vec{y}_n \in \mathfrak{X}$ in the problem domain $\mathfrak{X} \in \mathfrak{S}$ (cf. Section 2.1). Because operators are expressed in vector form, some of them require the (element-wise) Hadamard–Schur’s product \odot and the element-wise Heaviside step function $H : \mathbb{R}^D \rightarrow \mathbb{Z}_2^D$ with $H(0) = 1$. Besides, consider that $\|\cdot\|_2$ refers to the ℓ^2 -norm. Recall that $\vec{x}_n(t) \in X(t)$ stands as the current position of the n -th agent in a population $X(t)$ of size N , and $\vec{x}_*(t)$ corresponds to the best solution found at the current time t . Furthermore, we organised the perturbators according to their sophistication level in terms of mathematical formulation, which is not necessarily related to the numerical complexity.

The first four heuristics comprise the simplest ones implemented in the literature. They use basic ideas such as samples and walks guided by probability distributions. These operators are widespread with multiple variants and schemes [54]. Many of them are embedded in more sophisticated algorithms. Subsequently, the next four perturbators obey those inspired on evolutionary processes, which conform two well-known MHs such as GA [2] and DE [55]. The ninth and tenth simple heuristics correspond to those inspired on the idealised social behaviour of many living creatures [56]. We extracted them from two well-known metaheuristics as PSO [39] and fa [40]. Lastly, the final three perturbators concern those from MHs that model trajectories and dynamics common in the classical mechanics, such as SOA [41], CFO [53], and GSA [43].

Random Sample (RX): It neglects the n -th current position $\vec{x}_n(t)$ and replaces it with a candidate solution \vec{y}_n obtained via a random sample operation such as $\vec{y}_n = \vec{r}$, where $\vec{r} \in \mathfrak{S}$ is a vector of i.i.d. random numbers with $\vec{r} \ni r_i \sim \mathcal{U}(x_l, x_u)$. Likewise the **Restart** action but more general, this perturbation is an implementation of the **RX** initialiser as a perturbator.

Random Search (RS): It determines the candidate solution \vec{y}_n with a random search operation over the current position $\vec{x}_n(t)$, such as $\vec{y}_n = \vec{x}_n(t) + \alpha \vec{r}$, where $\alpha \in]0, 1]$ is the

step size factor and $\vec{r} \in \mathcal{G}$ is a vector of i.i.d. random numbers with $\mathcal{U}(x_l, x_u)$ or other probability distribution.

Random Flight (RF): It calculates the candidate solution \vec{y}_n by employing a random flight operation with respect to the current position \vec{x}_n as given

$$\vec{y}_n = \vec{x}_n(t) + \alpha \vec{r} \odot (\vec{x}_n(t) - \vec{x}_*(t)), \tag{A1}$$

where $\alpha \in \mathbb{R}_+$ is the spatial step size, $\vec{r} \in \mathbb{R}_+^D$ is a vector of i.i.d. random numbers with either uniform $\mathcal{U}(0, 1)$, normal standard $\mathcal{N}(0, 1)$, and symmetric Lévy stable $\mathcal{L}(1.5)$ distributions.

Local Random Walk (RW): It achieves the candidate solution \vec{y}_n by implementing a local random walk on the current position \vec{x}_n , such as

$$\vec{y}_n = \vec{x}_n + \alpha \vec{r} \odot H(p - \vec{q}) \odot (\vec{x}_{z_1}(t) - \vec{x}_{z_2}), \tag{A2}$$

where $\alpha, p \in]0, 1]$ are the step size factor and the probability of change. \vec{r} and \vec{q} are both vectors of i.i.d. random variables with $\mathcal{U}(0, 1)$ or $\mathcal{N}(0, 1)$. \vec{x}_{z_1} and \vec{x}_{z_2} , $z_1 \neq z_2$, are individuals selected from the population by following a given scheme. It is common to randomly choose them from the population, i.e., $z_1, z_2 \sim \mathcal{U}\{1, N\}$ and $z_1 \neq z_2$.

Genetic Crossover (GC): It obtains the candidate solution \vec{y}_n with a somehow complex procedure over the current population $X(t)$. This procedure first determines a ranked version of the population $\hat{X}(t)$ with respect to the cost function value i.e., $\hat{X}(t) = \{\hat{x}_1, \dots, \hat{x}_N\}$ with $f(\hat{x}_1) < \dots < f(\hat{x}_N)$. Such preliminary adjustment is known by several authors as natural selection [49]. Subsequently, \vec{y}_n is determined such as

$$\vec{y}_n = \vec{m} \odot \hat{x}_{z_1} + (1 - \vec{m}) \odot \hat{x}_{z_2}, \quad \forall z_1, z_2 \in \{1, \dots, M\}, n \in \{M + 1, \dots, N\}, \tag{A3}$$

where z_1 and z_2 are mutually exclusive indices from the population, i.e., $z_1 \neq z_2$, and $\vec{m} \in \mathbb{R}_+^D$ is the mask vector. Besides, $M = \lfloor m_p N \rfloor$ is the mating pool size since $m_p \in]0, 1]$ is the portion of the best ranked agents from the population of size N . These parameters are detailed as follows.

On the one hand, z_1 and z_2 refer to the parent indices chosen by using a pairing scheme from the mating pool [2,49]. The most common pairing schemes are *Random*, *Rank weighting*, *Roulette wheel*, and *Tournament* pairing. On the other hand, the mask vector is commonly assumed in the range $[0, 1]^D$, as we do in this work. However, some researchers incorporate values beyond such a range. This vector can be determined via diverse crossover mechanisms [49]. The most common ones are *Single-point*, *Two-points*, *Uniform*, *Blend*, and *Linear* crossover. Further information about the expressions corresponding to either pairing and crossover mechanisms can be found in [7,31].

Genetic Mutation (GM): This operator also works on a ranked version of the population $\hat{X}(t)$ with respect to the cost function value i.e., $\hat{X}(t) = \{\hat{x}_1, \dots, \hat{x}_N\}$ with $f(\hat{x}_1) < \dots < f(\hat{x}_N)$. Hence, the candidate position \vec{y}_n is calculated such as

$$\vec{y}_n = \vec{m} \odot \vec{x}_n(t) + \alpha(1 - \vec{m}) \odot \vec{r}, \quad \forall n \in \{\lfloor p_e N \rfloor, \dots, N\}, \tag{A4}$$

where $\vec{m} = H(p_m - \vec{q})$ is a mask vector, $\alpha \in [0, 1]$ is the spatial step size, \vec{r} and \vec{q} are vectors of i.i.d. random numbers with distribution $\mathcal{U}(x_l, x_u)$ and $\mathcal{U}(0, 1)$, respectively, $p_e \in [0, 1]$ is the elite portion of the population (i.e., the top $\lfloor p_e N \rfloor$ ranked individuals), and $p_m \in [0, 1]$ is the mutation probability. Furthermore, most researchers implement different probability distributions for drawing random disturbances along the problem domain [49].

Differential Mutation (DM): The candidate position \vec{y}_n is determined by using the differential mutation operator, which employs current positions of different agents in the population, as shown

$$\vec{y}_n = \vec{m} + \sum_{m=1}^M \alpha_m (\vec{x}_{z_{2m-1}}(t) - \vec{x}_{z_{2m}}(t)), \tag{A5}$$

where \vec{m} is the target vector which depends of the mutation scheme expressed such that $DE/\cdot/M$, $M \in \{1, 2, 3\}$ stands the number of random differences or perturbations, and $\alpha_m \in [0, 3]$ is a constant factor $\forall m \in \{0, \dots, M\}$. Plus, $z_i \forall i \in \mathbb{Z}_+$ is an integer with uniform random distribution between 1 and N , where N is the population size, then $z_i \sim \mathcal{U}\{1, N\}$ with $z_i \notin \bigcup_{j \in \mathbb{Z}_+ - \{i\}} z_j$. The most oft utilised schemes are listed as follows [38]:

- *rand*: $\vec{m} = \vec{x}_{z_0}(t)$,
- *best*: $\vec{m} = \vec{x}_*(t)$,
- *current*: $\vec{m} = \vec{x}_n(t)$,
- *current-to-best*: $\vec{m} = \vec{x}_n(t) + \alpha_0(\vec{x}_*(t) - \vec{x}_{z_0}(t))$,
- *rand-to-best*: $\vec{m} = \vec{x}_{z_0}(t) + \alpha_0(\vec{x}_*(t) - \vec{x}_{z_{M+1}}(t))$, and
- *rand-to-best-and-current*: $\vec{m} = \vec{x}_{z_0}(t) + \alpha_0(\vec{x}_*(t) - \vec{x}_n(t))$.

Differential Crossover (DC): This procedure always succeeds the **DM** operation in the basic DE implementation. Therefore, the candidate position $\vec{y}_n(t)$ is determined by ‘balancing’ (crossover operation) the components of the n -th current position $\vec{x}_n(t)$ and its immediately preceding position $\vec{x}_n(t - 1)$, such as

$$\vec{y}_n(t) = \vec{m} \odot \vec{x}_n(t - 1) + (1 - \vec{m}) \odot \vec{x}_n(t), \tag{A6}$$

where $\vec{m} \in \mathbb{Z}_2^D$ is the crossover mask vector calculated such as $\vec{m} = H(p_{CR} - \vec{r})$ by employing $H : \mathbb{R}^D \rightarrow \mathbb{Z}_2^D$ as the element-wise Heaviside step function with $H(0) = 1$, and $\vec{r} \in \mathbb{R}^D$ is a vector of i.i.d. uniformly random variables in $[0, 1]$, $\vec{r} \ni r_i \sim \mathcal{U}(0, 1)$. p_{CR} is the crossover probability that depends of the constant $CR \in [0, 1]$ and indicates if an element mutates. Its formula varies according to the crossover type [57], i.e., *Binomial crossover* or *Exponential crossover*.

It is worth mentioning that **DC** has shown poor performance when it is utilised apart from **DM**. For that reason, we disregarded this perturbator from the selected ones in our implementation. Notwithstanding, and for comparison purposes, we employed binomial variant of this heuristic when implementing the basic version of DE.

Particle Swarm Dynamic (PS): The candidate position $\vec{y}_n(t)$ for the n -th agent is rendered via the swarm dynamic operation using its current position $\vec{x}_n(t)$ and velocity $\vec{v}_n(t)$, such as

$$\vec{y}_n = \vec{x}_n(t) + \vec{v}_n(t). \tag{A7}$$

This velocity can be determined through different approaches from the literature. The most common and simplest ones, widely implemented in several practical problems, are the *inertial* and *constrained* approaches, which follow the expression

$$\vec{v}_n(t) = \alpha_0 \vec{v}_n(t - 1) + \alpha_1 \phi_1 \vec{r}_1 \odot (\vec{x}_{n,*}(t) - \vec{x}_n(t)) + \alpha_2 \phi_2 \vec{r}_2 \odot (\vec{x}_*(t) - \vec{x}_n(t)). \tag{A8}$$

In this formula, $\alpha_i \in [0, 1]$ are velocity scale coefficients, $\forall i \in \{0, 1, 2\}$, $\phi_1, \phi_2 \in [0, 4]$, are known as the self and swarm confidence coefficients, respectively, and $\vec{r}_j \in \mathbb{R}^D$ are i.i.d. random vectors with $\mathcal{U}(0, 1)$, $\forall j \in \{1, 2\}$ [58]. Now, depending of the α_i values set, the velocity is determined using whether the *inertial* or *constrained* approach, as detail below.

- *Inertial approach* employs $\alpha_0 \in [0, 1]$, $\alpha_2 = \alpha_3 = 1$. In this case, α_0 is called inertial weight and can be used as a fixed value, a time-dependent function, or a random number [59,60].
- *Constrained approach* utilises $\alpha_0 = \alpha_1 = \alpha_2 = \chi$, since χ is known as the constriction factor determined by $\chi = 2\kappa H(\phi - 4) / (\phi - 2 - \sqrt{\phi(\phi - 4)}) + \sqrt{\kappa}(1 - H(\phi - 4))$, since $\phi = \phi_1 + \phi_2$ and $\kappa \in [0, 1]$ [39,59].

Moreover, $\vec{v}_n(t - 1)$ corresponds to the velocity of the n -th agent, whilst $\vec{x}_{n,*}(t)$ and $\vec{x}_*(t)$ are the best position found by each individual and by the entire population, respectively. The main trade-off of this heuristic for its mathematical simplicity is the requirement of memory to preserve the previous velocity $\vec{v}_n(t - 1)$ and the particular best solution $\vec{x}_{n,*}(t)$ for each agent $n \in \{1, \dots, N\}$.

Firefly Dynamic (FD): The candidate solution $\vec{y}_n(t)$ obtained via the firefly dynamic operator, which compares the current agent position $\vec{x}_n(t)$ with that of the other agents, such as

$$\vec{y}_n = \vec{x}_n(t) + \alpha_0 \vec{r} + \alpha_1 \sum_{k=1}^N H(-(I_k - I_n)) (\vec{x}_k(t) - \vec{x}_n(t)) e^{-\alpha_2 \|\vec{x}_k(t) - \vec{x}_n(t)\|_2^2}, \quad (A9)$$

where $\alpha_0, \alpha_1 \in [0, 1]$ and $\alpha_2 \in [0, 1000]$ are constant factors, \vec{r} is a vector of i.i.d. random variables with $\mathcal{U}(-0.5, 0.5)$ or another distribution, and $I_n = f(\vec{x}_n)$, $\forall n \in \{1, \dots, N\}$, is the light intensity of agent n .

Spiral Dynamic (SD): The candidate position $\vec{y}_n(t)$ is determined from the logarithmic spiral trajectory given from the current position $\vec{x}_n(t)$ and the the rotation centre $\vec{x}_*(t)$, such as,

$$\vec{y}_n = \vec{x}_*(t) - \vec{r} \odot \mathbf{R}_D(\theta) \cdot (\vec{x}_n(t) - \vec{x}_*(t)), \quad (A10)$$

where $\theta \in [0^\circ, 360^\circ]$ is the rotation angle, and \vec{r} is a vector of i.i.d. random variables with $\mathcal{U}(r_0 - \sigma_r, r_0 + \sigma_r)$, since $r_0 \in [0, 1]$ is the convergence rate or central radius and $\sigma_r \in [0, \min\{r_0, 1 - r_0\}]$. Plus, $\mathbf{R}_D(\theta) \in \mathbb{R}^{D \times D}$ is the rotation matrix determined as the product of all the $D(D - 1)/2$ combinations of two-dimensional rotation matrices in \mathbb{R}^D . Consider that this rotation centre is given by the best current position found by the population.

Central Force (CF): This perturbator determines the candidate position $\vec{y}_n(t)$ by using a Newtonian gravitational attraction approach given by

$$\vec{y}_n = \vec{x}_n(t) + \frac{1}{2} \vec{a}_n(t) \Delta t^2, \quad (A11a)$$

where $\vec{a}_n(t) \in \mathbb{R}^D$ is the acceleration vector for the n -th agent and $\Delta t \in \mathbb{R}_+$ is the time interval between steps, i.e., $\Delta t = 1$. This acceleration component is obtained such as,

$$\vec{a}_n(t) = \alpha_0 \sum_{\substack{k=1 \\ k \neq n}}^N H(M_k(t) - M_n(t)) (M_k(t) - M_n(t))^{\alpha_1} \frac{\vec{x}_k(t) - \vec{x}_n(t)}{\|\vec{x}_k(t) - \vec{x}_n(t)\|_2^{\alpha_2}}, \quad (A11b)$$

where $M_n = f(\vec{x}_n(t))$, $\forall n \in \{1, \dots, N\}$, is the mass of agent n , $\alpha_0 \in [0, 0.01]$ is the gravitational constant, and $\alpha_1 \in [0, 0.01]$, $\alpha_2 \in [1, 2]$ are two positive factors.

Gravitational Search (GS): This operator is founded in kinematic and gravitational interactions. So, the candidate position $\vec{y}_n(t)$ is achieved as given

$$\vec{y}_n = \vec{x}_n(t) + \vec{r}_0 \odot \vec{v}_n(t), \quad (A12a)$$

since $\vec{v}_n(t)$ is the velocity of the agent n . This parameter is attained as

$$\vec{v}_n(t) = \vec{r} \odot \vec{v}_n(t - 1) + \vec{a}_n(t), \tag{A12b}$$

where \vec{r}_0 is a vector of i.i.d. random variables with $\mathcal{U}(0, 1)$. Plus, $\vec{a}_n(t)$ is the acceleration component calculated by

$$\vec{a}_n(t) = G(t) \sum_{\substack{k=1 \\ k \neq n}}^N M_k(t) \vec{r}_k \odot \frac{\vec{x}_k(t) - \vec{x}_n(t)}{\|\vec{x}_k(t) - \vec{x}_n(t)\|_2 + \varepsilon_g}, \tag{A12c}$$

since $G(t) \in \mathbb{R}_+$ is the gravitational factor that can be time dependent, $M_n(t)$ is the mass of agent n , \vec{r}_k is a vector of i.i.d. random variables with $\mathcal{U}(0, 1)$, and $\varepsilon_g \in \mathbb{R}_+$ is a small constant. In particular, $G(t)$ and $M_n(t)$ are determined with

$$G(t) = \alpha_0 e^{-\alpha_1 t} \quad \text{and} \quad M_n(t) = \frac{f(\vec{x}_o(t)) - f(\vec{x}_n(t))}{Nf(\vec{x}_o(t)) - \sum_{k=1}^N f(\vec{x}_k(t))}, \tag{A12d}$$

where $\alpha_0 \in [0, 1]$ is the initial gravitational value, $\alpha_1 \in [0, 0.1]$ is the damping ratio, and $f(\vec{x}_o(t))$ is the worst fitness value at $\vec{x}_o(t)$, such that $\vec{x}_o(t) = \text{argsup} \cup_{n=1}^N \{f(\vec{x}_n(t))\}$.

Appendix A.3. Selectors

From the prior mentioned metaheuristics, we chose the four most representative selection operators, i.e., **Direct**, **Greedy**, **Probabilistic**, and **Metropolis selectors**. Before describing them, recall that $\vec{x}_n(t) \in X(t)$ and $\vec{y}_n \in \mathfrak{X}$ are the n -th agent position from the current population and its candidate position, respectively. In addition, their corresponding fitness values are $f(\vec{x}_n)$ and $f(\vec{y}_n)$, since $f : \mathfrak{X} \rightarrow \mathbb{R}$ is the objective function to minimise, and the difference between them is given by $\Delta f_n(t) = f(\vec{y}_n) - f(\vec{x}_n(t))$. Therefore, the new position $\vec{x}_n(t + 1)$ of the n -th agent is assigned by using the previous information and any of the selectors listed below.

Direct: The new position $\vec{x}_n(t + 1)$ of the n -th agent is straightforwardly designated equal to the candidate one \vec{y}_n , i.e., $\vec{x}_n(t + 1) = \vec{y}_n$.

Greedy: The new position $\vec{x}_n(t + 1)$ for the n -th agent is assigned equal to \vec{y}_n if it improves the current position $\vec{x}_n(t)$, i.e., $\Delta f_n \leq 0$, thus

$$\vec{x}_n(t + 1) = \begin{cases} \vec{y}_n, & \text{if } \Delta f_n \leq 0, \\ \vec{x}_n(t), & \text{otherwise.} \end{cases} \tag{A13}$$

Probabilistic: The new position $\vec{x}_n(t + 1)$ for the n -th agent is set equal to \vec{y}_n if it improves the current position $\vec{x}_n(t)$, i.e., $\Delta f_n \leq 0$. Should it be worsened, \vec{y}_n is selected with a given probability $p_s \in [0, 1]$, such that

$$\vec{x}_n(t + 1) = \begin{cases} \vec{y}_n, & \text{if } (\Delta f_n \leq 0) \vee (p_s < r), \\ \vec{x}_n(t), & \text{otherwise.} \end{cases} \tag{A14}$$

Metropolis: The new position $\vec{x}_n(t + 1)$ for the n -th agent is set equal to \vec{y}_n if it improves the current position $\vec{x}_n(t)$, i.e., $\Delta f_n \leq 0$. Should it be worsened ($\Delta f_n > 0$), \vec{y}_n is selected with a given probability of acceptance, as shown

$$\vec{x}_n(t + 1) = \begin{cases} \vec{y}_n, & \text{if } (\Delta f_n \leq 0) \vee \left(r < e^{-\frac{\Delta f_n(t)}{k_B \Theta(t)}} \right), \\ \vec{x}_n(t), & \text{otherwise,} \end{cases} \tag{A15}$$

since $k_B \in \mathbb{R}_+$ is the Boltzmann's constant and $\Theta(t) : \mathbb{Z}_+ \rightarrow \mathbb{R}_+$ is the temperature which decreases slowly with each iteration. There are many flavours for the temperature expression $\Theta(t)$ in the literature [35].

Appendix A.4. Finalisers

Finally, but certainly not least important, we present the most common simple heuristics in charge of deciding when the search procedure must end. According to Definition 7, the differentiating point from one finaliser to another is the criteria function c_f , which maps a set of measurements or features to a Boolean quantity. For that reason, we use the well-known Heaviside function $H : \mathbb{R} \mapsto \mathbb{Z}_2$ in the criteria evaluation. These criteria can be found in the literature as *stopping conditions* [61]. It is worth noticing that these criteria are not exclusive, so one or more can be employed conjunctively as a finaliser, i.e.,

$$c_f(t, f, X, \dots) \triangleq \bigwedge_{\forall i} c_{f,i}(t, f, X, \dots), \quad (\text{A16})$$

where $c_{f,i}$ is a particular criterion that depends of at least one parameter: the current iteration t , objective function f , agents positions X , so forth. The most common criteria found in several numerical methods as briefly described below.

Iteration number: In a searching procedure with t as the current iteration and t_{\max} as the maximal number of iterations allowed, the criteria function is defined as $c_{f,in}(t) \triangleq H(t - t_{\max})$.

Function tolerance: After finding the current best position $\vec{x}_*(t)$ in an iterative procedure, its fitness is compared against the previous one $\vec{x}_*(t-1)$, with the objective function $f(\vec{x})$, employing the following criterion,

$$c_{f,ft}(t, f, X) \triangleq H\left(\varepsilon_{ft} - \frac{|f(\vec{x}_*(t)) - f(\vec{x}_*(t-1))|}{|\sup\{1, f(\vec{x}_*(t))\}|}\right). \quad (\text{A17})$$

The function tolerance criterion $c_{f,ft}(f)$ depends of a threshold value ε_{ft} established to limit the closeness of these two function values.

Domain tolerance: Similar to the **Function tolerance** finalisation, but this one only compares the positions using a distance metric, e.g., the ℓ^2 -norm, such as

$$c_{f,dt}(t, f, X) \triangleq H(\varepsilon_{dt} - \|\vec{x}_*(t) - \vec{x}_*(t-1)\|_2), \quad (\text{A18})$$

since ε_{dt} is the threshold value to evaluate the closeness of these positions.

In this work, we dismiss the finalisers found from the analysed metaheuristics because we delegate this function to a superior technique. Further information is provided in Section 3, where we describe the proposed model as well as its main motivation. Notwithstanding, we implemented only the **Iteration number** finaliser in the basic metaheuristics used for comparison purposes (cf. Section 4).

References

1. Sörensen, K.; Sevaux, M.; Glover, F. A history of metaheuristics. *Handb. Heuristics* **2018**, *2*, 791–808. [CrossRef]
2. Goldberg, D.; Holland, J. Genetic algorithms and machine learning. *Mach. Learn.* **1988**, *3*, 95–99. [CrossRef]
3. Kirkpatrick, S.; Gelatt, C.D., Jr.; Vecchi, M.P. Optimization by Simulated Annealing. *Science* **1983**, *220*, 671–680. [CrossRef] [PubMed]
4. Cruz-Duarte, J.M.; Amaya, I.; Ortíz-Bayliss, J.C.; Correa, R. Solving microelectronic thermal management problems using a generalized spiral optimization algorithm. *Appl. Intell.* **2021**. [CrossRef]
5. Hashim, F.A.; Hussain, K.; Houssein, E.H.; Mabrouk, M.S.; Al-Atabany, W. Archimedes optimization algorithm: A new metaheuristic algorithm for solving optimization problems. *Appl. Intell.* **2021**, *51*, 1531–1551. [CrossRef]
6. Sörensen, K. Metaheuristics—The metaphor exposed. *Int. Trans. Oper. Res.* **2015**, *22*, 3–18. [CrossRef]

7. Cruz-Duarte, J.M.; Ortiz-Bayliss, J.C.; Amaya, I.; Shi, Y.; Terashima-Marín, H.; Pillay, N. Towards a Generalised Metaheuristic Model for Continuous Optimisation Problems. *Mathematics* **2020**, *8*, 2046. [\[CrossRef\]](#)
8. Adam, S.P.; Alexandropoulos, S.A.N.; Pardalos, P.M.; Vrahatis, M.N. No Free Lunch Theorem: A Review. In *Approximation and Optimization*; Demetriou, I., Pardalos, P., Eds.; Springer: Cham, Switzerland, 2019; pp. 57–82. [\[CrossRef\]](#)
9. Parsopoulos, K.E.; Vrahatis, M.N. UPSO: A unified particle swarm optimization scheme. *Lect. Ser. Comput. Comput. Sci.* **2004**, *1*, 868–873.
10. Dao, S.D.; Abhary, K.; Marian, R. A bibliometric analysis of Genetic Algorithms throughout the history. *Comput. Ind. Eng.* **2017**, *110*, 395–403. [\[CrossRef\]](#)
11. Huang, C.; Li, Y.; Yao, X. A Survey of Automatic Parameter Tuning Methods for Metaheuristics. *IEEE Trans. Evol. Comput.* **2020**, *24*, 201–216. [\[CrossRef\]](#)
12. Durgut, R.; Aydin, M.E. Adaptive binary artificial bee colony algorithm. *Appl. Soft Comput.* **2021**, *101*, 107054. [\[CrossRef\]](#)
13. Santos, C.E.d.S.; Sampaio, R.C.; Coelho, L.d.S.; Bestarsd, G.A.; Llanos, C.H. Multi-objective adaptive differential evolution for SVM/SVR hyperparameters selection. *Pattern Recognit.* **2021**, *110*, 107649. [\[CrossRef\]](#)
14. Sevaux, M.; Sörensen, K.; Pillay, N. Adaptive and Multilevel Metaheuristics. In *Handbook of Heuristics*; Springer International Publishing: Cham, Switzerland, 2018; pp. 3–21. [\[CrossRef\]](#)
15. Miranda, P.B.; Prudêncio, R.B.; Pappa, G.L. H3AD: A hybrid hyper-heuristic for algorithm design. *Inf. Sci.* **2017**, *414*, 340–354. [\[CrossRef\]](#)
16. Raidl, G.R. A unified view on hybrid metaheuristics. In *International Workshop on Hybrid Metaheuristics*; Springer: Berlin/Heidelberg, Germany, 2006; pp. 1–12.
17. Talbi, E.G. A taxonomy of hybrid metaheuristics. *J. Heuristics* **2002**, *8*, 541–564. [\[CrossRef\]](#)
18. Barzinpour, F.; Noorossana, R.; Niaki, S.T.A.; Ershadi, M.J. A hybrid Nelder–Mead simplex and PSO approach on economic and economic-statistical designs of MEWMA control charts. *Int. J. Adv. Manuf. Technol.* **2012**. [\[CrossRef\]](#)
19. Hassan, A.; Pillay, N. Hybrid metaheuristics: An automated approach. *Expert Syst. Appl.* **2019**, *130*, 132–144. [\[CrossRef\]](#)
20. Krawiec, K.; Simons, C.; Swan, J.; Woodward, J. Metaheuristic design patterns: New perspectives for larger-scale search architectures. In *Handbook of Research on Emergent Applications of Optimization Algorithms*; IGI Global: Hershey, PA, USA, 2018; pp. 1–36.
21. Stützle, T.; López-Ibáñez, M. Automated design of metaheuristic algorithms. In *Handbook of Metaheuristics*; Springer: Cham, Switzerland, 2019; pp. 541–579.
22. Burke, E.K.; Hyde, M.R.; Kendall, G.; Ochoa, G.; Özcan, E.; Woodward, J.R. A classification of hyper-heuristic approaches: Revisited. In *Handbook of Metaheuristics*; Springer: Cham, Switzerland, 2019; pp. 453–477.
23. Pillay, N.; Qu, R. *Hyper-Heuristics: Theory and Applications*; Springer International Publishing: Berlin, Germany, 2018.
24. Del Ser, J.; Osaba, E.; Molina, D.; Yang, X.S.; Salcedo-Sanz, S.; Camacho, D.; Das, S.; Suganthan, P.N.; Coello, C.A.C.; Herrera, F. Bio-inspired computation: Where we stand and what's next. *Swarm Evol. Comput.* **2019**, *48*, 220–250. [\[CrossRef\]](#)
25. Woumans, G.; De Boeck, L.; Beliën, J.; Creemers, S. A column generation approach for solving the examination-timetabling problem. *Eur. J. Oper. Res.* **2016**, *253*, 178–194. [\[CrossRef\]](#)
26. M. Almufti, S. Historical survey on metaheuristics algorithms. *Int. J. Sci. World* **2019**, *7*, 1. [\[CrossRef\]](#)
27. Hussain, K.; Salleh, M.N.M.; Cheng, S.; Shi, Y. Metaheuristic research: A comprehensive survey. *Artif. Intell. Rev.* **2019**, *52*, 2191–2233. [\[CrossRef\]](#)
28. Dokeroglu, T.; Sevinc, E.; Kucukyilmaz, T.; Cosar, A. A survey on new generation metaheuristic algorithms. *Comput. Ind. Eng.* **2019**, *137*, 106040. [\[CrossRef\]](#)
29. Drake, J.H.; Kheiri, A.; Özcan, E.; Burke, E.K. Recent advances in selection hyper-heuristics. *Eur. J. Oper. Res.* **2020**, *285*, 405–428. [\[CrossRef\]](#)
30. Qu, R.; Kendall, G.; Pillay, N. The General Combinatorial Optimisation Problem: Towards Automated Algorithm Design. *IEEE Comput. Intell. Mag.* **2020**, *15*, 14–23. [\[CrossRef\]](#)
31. Cruz-Duarte, J.M.; Amaya, I.; Ortiz-Bayliss, J.C.; Terashima-Marín, H.; Shi, Y. CUSTOMHyS: Customising Optimisation Metaheuristics via Hyper-heuristic Search. *SoftwareX* **2020**, *12*, 100628. [\[CrossRef\]](#)
32. Sanchez-Diaz, X.F.C.; Ortiz-Bayliss, J.C.; Amaya, I.; Cruz-Duarte, J.M.; Conant-Pablos, S.E.; Terashima-Marín, H. A Preliminary Study on Feature-independent Hyper-heuristics for the 0/1 Knapsack Problem. In Proceedings of the 2020 IEEE Congress on Evolutionary Computation (CEC), Glasgow, UK, 19–24 July 2020; pp. 1–8. [\[CrossRef\]](#)
33. Han, L.; Kendall, G. Guided operators for a hyper-heuristic genetic algorithm. In *Australasian Joint Conference on Artificial Intelligence*; Springer: Berlin/Heidelberg, Germany, 2003; pp. 807–820.
34. Archetti, F.; Candelieri, A., From Global Optimization to Optimal Learning. In *Bayesian Optimization and Data Science*; Springer International Publishing: Cham, Switzerland, 2019; Chapter 2, pp. 19–35. [\[CrossRef\]](#)
35. Franzin, A.; Stützle, T. Revisiting simulated annealing: A component-based analysis. *Comput. Oper. Res.* **2019**, *104*, 191–206. [\[CrossRef\]](#)
36. Dianati, M.; Song, I.; Treiber, M. An introduction to genetic algorithms and evolution strategies. *Sadhana* **2002**, *24*, 293–315.
37. Shehab, M.; Khader, A.T.; Al-Betar, M.A. A survey on applications and variants of the cuckoo search algorithm. *Appl. Soft Comput. J.* **2017**, *61*, 1041–1059. [\[CrossRef\]](#)

38. Das, S.; Mullick, S.S.; Suganthan, P.N. Recent advances in differential evolution-An updated survey. *Swarm Evol. Comput.* **2016**, *27*, 1–30. [\[CrossRef\]](#)
39. Clerc, M.; Kennedy, J. The particle swarm-explosion, stability, and convergence in a multidimensional complex space. *Evol. Comput. IEEE Trans.* **2002**, *6*, 58–73. [\[CrossRef\]](#)
40. Fister, I.; Yang, X.S.; Brest, J. A comprehensive review of firefly algorithms. *Swarm Evol. Comput.* **2013**, *13*, 34–46. [\[CrossRef\]](#)
41. Cruz-Duarte, J.M.; Martin-Diaz, I.; Munoz-Minjares, J.U.; Sanchez-Galindo, L.A.; Avina-Cervantes, J.G.; Garcia-Perez, A.; Correa-Cely, C.R. Primary study on the stochastic spiral optimization algorithm. In Proceedings of the 2017 IEEE International Autumn Meeting on Power, Electronics and Computing (ROPEC), Ixtapa, Mexico, 8–10 November 2017; Volume 1, pp. 1–6.
42. Behniya, M.; Ayati, A.H.; Derakhshani, A.; Haghghi, A. Application of the central force optimization (CFO) method to the soil slope stability analysis. In Proceedings of the International Conference on Progress in Science and Technology, London, UK, 20 November 2016; p. 11.
43. Biswas, A.; Mishra, K.K.; Tiwari, S.; Misra, A.K. Physics-Inspired Optimization Algorithms: A Survey. *J. Optim.* **2013**, *2013*, 1–16. [\[CrossRef\]](#)
44. Cruz-Duarte, J.M.; Amaya, I.; Ortiz-Bayliss, J.C.; Conant-Pablos, S.E.; Terashima-Marín, H. A primary study on hyper-heuristics to customise metaheuristics for continuous optimisation. In Proceedings of the 2020 IEEE Congress on Evolutionary Computation (CEC), Glasgow, UK, 19–24 July 2020; pp. 1–8. [\[CrossRef\]](#)
45. Carrasco, J.; García, S.; Rueda, M.; Das, S.; Herrera, F. Recent trends in the use of statistical tests for comparing swarm and evolutionary computing algorithms: Practical guidelines and a critical review. *Swarm Evol. Comput.* **2020**, *54*, 100665. [\[CrossRef\]](#)
46. García, S.; Molina, D.; Lozano, M.; Herrera, F. A study on the use of non-parametric tests for analyzing the evolutionary algorithms' behaviour: A case study on the CEC'2005 Special Session on Real Parameter Optimization. *J. Heuristics* **2009**, *15*, 617–644. [\[CrossRef\]](#)
47. Goldberg, D.E.; Deb, K. A comparative analysis of selection schemes used in genetic algorithms. In *Foundations of Genetic Algorithms*; Morgan Kaufmann Publishers, Inc.: San Mateo, CA, USA, 1991; Volume 1, pp. 69–93.
48. Bergstra, J.; Bengio, Y. Random search for hyper-parameter optimization. *J. Mach. Learn. Res.* **2012**, *13*, 281–305.
49. Ahn, C.W. *Practical Genetic Algorithms*; Springer, Berlin/Heidelberg, Germany, 2006; Volume 18, pp. 7–22. [\[CrossRef\]](#)
50. Yang, X.S.; Deb, S. Cuckoo search via Lévy flights. In Proceedings of the 2009 World Congress on Nature & Biologically Inspired Computing (NaBIC), Coimbatore, India, 9–11 December 2009; pp. 210–214.
51. Kennedy, J.; Eberhart, R. Particle swarm optimization (PSO). In Proceedings of the IEEE International Conference on Neural Networks, Perth, Australia, 27 November–1 December 1995; pp. 1942–1948.
52. Yang, X.S. Firefly algorithm. In *Nature-Inspired Metaheuristic Algorithms*; Luniver Press: Cambridge, UK, 2008; Volume 20, pp. 79–90.
53. Formato, R.A. Central force optimization: A new deterministic gradient-like optimization metaheuristic. *Opsearch* **2009**, *46*, 25–51. [\[CrossRef\]](#)
54. Schumer, M.A.; Steiglitz, K. Adaptive Step Size Random Search. *IEEE Trans. Autom. Control.* **1968**, *13*, 270–276. [\[CrossRef\]](#)
55. Price, K.; Storn, R. *Differential Evolution—A Simple and Efficient Adaptive Scheme for Global Optimization Over Continuous Space*; Technical Report; International Computer Science Institute: Berkeley, CA, USA, 1995.
56. Kar, A.K. Bio inspired computing—A review of algorithms and scope of applications. *Expert Syst. Appl.* **2016**, *59*, 20–32. [\[CrossRef\]](#)
57. Zaharie, D. A Comparative Analysis of Crossover Variants in Differential Evolution. In Proceedings of the International Multiconference on Computer Science and Information Technology, IMCSIT 2007, Wisła, Poland, 15–17 October 2007; pp. 171–181.
58. Bonyadi, M.R.; Michalewicz, Z. Particle swarm optimization for single objective continuous space problems: A review. *Evol. Comput.* **2017**, *25*, 1–54. [\[CrossRef\]](#)
59. Imran, M.; Hashim, R.; Khalid, N.E.A. An overview of particle swarm optimization variants. *Procedia Eng.* **2013**, *53*, 491–496. [\[CrossRef\]](#)
60. Zhang, Y.; Wang, S.; Ji, G. A comprehensive survey on particle swarm optimization algorithm and its applications. *Math. Probl. Eng.* **2015**, *2015*. [\[CrossRef\]](#)
61. Rao, S.S. *Engineering Optimization: Theory and Practice*; John Wiley & Sons: Hoboken, NJ, USA, 2009.