Swarm and Evolutionary Computation

Hyper-Heuristics to Customise Metaheuristics for Continuous Optimisation

--Manuscript Draft--

Manuscript Number:	
Article Type:	Full Length Article
Keywords:	Metaheuristic; Hyper-heuristic; Search operators; Evolutionary computation.
Corresponding Author:	José Carlos Ortiz-Bayliss, Ph.D. Instituto Tecnologico y de Estudios Superiores de Monterrey Monterrey, Nuevo León MEXICO
First Author:	Jorge Mario Cruz-Duarte, Ph.D.
Order of Authors:	Jorge Mario Cruz-Duarte, Ph.D.
	Ivan Amaya, Ph.D.
	José Carlos Ortiz-Bayliss, Ph.D.
	Santiago Enrique Conant-Pablos, Ph.D.
	Hugo Terashima-Marín, Ph.D.
	Yong Shi, Ph.D.
Abstract:	Literature is prolific with metaheuristics for solving continuous optimisation problems. But, in practice, it is difficult to choose one appropriately. Moreover, it is necessary to determine a good enough set of parameters for the selected approach. Hence, this work proposes a strategy based on a hyper-heuristic powered by Simulated Annealing for tailoring population-based metaheuristics. Moreover, our approach considers search operators from well-known techniques as building blocks for new ones. We test this strategy through 107 continuous benchmark functions and by varying their dimensions. It is possible to obtain metaheuristics with diverse configurations from several cases of study. We prove the potential of the proposed framework for devising metaheuristics to solve continuous optimization problems with different characteristics, similar to those from practical engineering scenarios.
Suggested Reviewers:	Juan Gabriel Avina-Cervantes, Ph.D. Professor, Universidad de Guanajuato - Campus Irapuato-Salamanca avina@ugto.mx He has expertise in global optimisation, mathematical methods, and search operators.
	Ender Özcan, Ph.D. Professor, University of Nottingham School of Computer Science Ender.Ozcan@nottingham.ac.uk He has expertise in hyper-heuristics, search operators, and combinatorial problems
	Carlos Rodrigo Correa-Cely, Ph.D. Professor, Universidad Industrial de Santander crcorrea@saber.uis.edu.co He has expertise in continuous optimisation problems, engineering optimisation, metaheuristics, and numerical methods
	Gabriela Ochoa, Ph.D. Professor, University of Stirling gabriela.ochoa@cs.stir.ac.uk As an expert in the field of hyper-heuristics and landscape analysis, her expertise would be of great importance to validate the contribution of the manuscript.
Opposed Reviewers:	

Hyper-Heuristics to Customise Metaheuristics for Continuous Optimisation*

Jorge M. Cruz-Duarte[a], Ivan Amaya[a], José C. Ortiz-Bayliss[a,*], Santiago E. Conant-Pablos[a], Hugo Terashima-Marín[a], Yong Shi[b]

[a] *School of Engineering and Sciences, Tecnologico de Monterrey, Av. Eugenio Garza Sada 2501 Sur, Monterrey, NL 64849, México*

[b] *Research Center on Fictitious Economy and Data Science, Chinese Academy of Sciences, Zhongguancun East Road 80, Haidian District, Beijing 100190, China*

Abstract

Literature is prolific with metaheuristics for solving continuous optimisation problems. But, in practice, it is difficult to choose one appropriately. Moreover, it is necessary to determine a good enough set of parameters for the selected approach. Hence, this work proposes a strategy based on a hyper-heuristic powered by Simulated Annealing for tailoring population-based metaheuristics. Moreover, our approach considers search operators from well-known techniques as building blocks for new ones. We test this strategy through 107 continuous benchmark functions and by varying their dimensions. It is possible to obtain metaheuristics with diverse configurations from several cases of study. We prove the potential of the proposed framework for devising metaheuristics to solve continuous optimization problems with different characteristics, similar to those from practical engineering scenarios.

Keywords: Metaheuristic, Hyper-heuristic, Search operators, Evolutionary computation.

1. Introduction

Technological breakthroughs offer us comfort and improve our quality of life. But it also gives birth to new continuous optimisation problems. Metaheuristics (MHs), defined as general-purpose methods, have been proposed to tackle such real-world problems. They are characterised by their flexibility, versatility, and algorithmic simplicity when facing a problem. A vast number of metaheuristics claiming to be the best one for solving engineering problems has been reported [1, 2]. However, they are not so general: the *no-free-lunch* theorem reigns [3]. In practice, researchers and practitioners must know how to properly select a MH for a given problem. Even then, they must also know how to set its parameters. Therefore, the question of deciding which MH is worth implementing to solve a defined problem remains open.

Throughout the last three decades, several MHs with curious metaphors (or "flavours") have been presented to face various problems [4, 5]. However, they are not so different from conventional MHs such as Simulated Annealing (SA), Differential Evolution (DE), and Particle Swarm Optimisation (PSO). If we disassemble them into simple heuristics, say, search operators, it is easy to notice that many of these are variations of some basic ones, *e.g.,* mutation [6], crossover [7], and Lévy flight [8]. Some authors have taken advantage of this fact for setting a procedure by selecting two or more simple heuristics to render MHs with excellent performances in particular problems [9]. The idea of algorithms for combining heuristics is not entirely new, as it goes back to the 1960s. But only recently it has grown into an optimisation sub-area known as hyper-heuristics (HHs) [10]. Of course, some alternative methodologies have also appeared, but they are beyond the scope of this work.

HHs provide a general approach, which have been defined as a high-level heuristic. Such a heuristic selects or modifies low-level heuristics as a mean of finding better solutions for a problem domain [11]. It is interesting

*The research was supported by the Research Group in Intelligent Systems at the Tecnológico de Monterrey (México), the Project TEC-Chinese Academy of Sciences, and by the CONACyT Basic Science Project with grant number 287479.

*Corresponding author

Email addresses: jorge.cruz@tec.mx (Jorge M. Cruz-Duarte), iamaya2@tec.mx (Ivan Amaya), jcobayliss@tec.mx (José C. Ortiz-Bayliss), sconant@tec.mx (Santiago E. Conant-Pablos), terashima@tec.mx (Hugo Terashima-Marín), yshi@ucas.ac.un (Yong Shi)

to see that many HHs have been applied to combinatorial problems [12, 13], whilst only a few have dealt with continuous ones [11]. For example, Miranda *et al.* proposed a Hybrid Hyper-Heuristic for Algorithm Design (H3AD) [14]. The authors used H3AD for optimising (or redesigning) the well-known PSO algorithm to 60 continuous benchmark problems. Their results showed that, in at least 80% of the times, customised algorithms outperformed their counterparts. Moreover, the selection process reduced computational cost. Nevertheless, this strategy employed the PSO grammar as design schema. So, it may prove somewhat difficult to extend this idea to other metaheuristics. In a similar approach, Abell *et al.* generated an algorithm portfolio that included DE and PSO, and which targeted the Black-Box Optimisation Benchmarking problems [15]. They demonstrated that, by incorporating feature extraction it is possible to consider problem structure when selecting the best solution method within the portfolio.

Despite the scarce research dealing with high-level solvers for continuous optimisation, the available ones lack a proper generalisation scheme. Hence, we proposes a HH-based strategy powered by Simulated Annealing (SA) for filling such a knowledge gap. The main idea of this work is based on the Sichuan's popular quote that Deng Xioaping used to say [16]: "*It does not matter whether a cat is yellow or white, as long as it catches mice.*" Rephrasing, we aim to provide a structured and innovative way to find a suitable method for solving a given problem, only using mathematical operations over a population, no matter what name they could possess. Our approach generates custom population-based MHs to solve continuous optimisation problems. Each custom MH is built by cascading one or more search operators collected from well-known MHs. The number of these search operators in the sequence is called cardinality, which can be adjusted to the needs of the implementation. We analysed ten well-known metaheuristics by extracting their search operators (SOs), then generated two collections, one with 205 SOs and another with 1274033 SOs. In addition, we constructed a collection with 66 predefined metaheuristics, derived from those which were early analysed, for comparison purposes. To carry out all the experiments, we selected and categorised 107 continuous benchmark functions using three categorical features and seven dimensionalities. Therefore, we deploy the proposed HH model by utilising both heuristic collections with a maximum cardinality of three and five. Results show the potential of the proposed HH-based framework for devising metaheuristics to solve continuous optimization problems with different dimensionalities and characteristics. Particularly, we demonstrate that the tailored metaheuristics outperform 63% of the basic ones in the worst scenario studied in this work.

This document is organised as follows. Sect. 2 introduces the theoretical foundations, where we detail all concepts employed to develop either the framework and the customised metaheuristics. This section is essential because several of the foundations related with this work are currently matter of controversy by some researchers. Subsequently, Sect. 3 describes the testing methodology and how the proposed model works. Then, Sect. 4 presents and discusses the experiments carried out and their corresponding data. This section is mainly composed by two central parts, the first one concerns the analysis and extraction performed for the search operators as well as the generation of heuristic collections; the second part focuses on the deployment of the proposed methodology. Finally, Sect. 5 highlights the most relevant conclusions and lays out some paths for future works.

2. Theoretical foundations

2.1. Optimisation

Optimisation is somehow implicit in nature. Even so, it mainly concerns mathematical procedures for reaching the best result of a problem in practical engineering scenarios. In this work, we use the problem definition given by a feasible domain and an objective function to minimise, as follows:

Definition 1 (Problem domain)**.** Let $\mathfrak{X}$ be a feasible set or problem domain such as it is simply established by

$$\mathfrak{X} = \{\vec{x} \in \mathbb{R}^D \,:\, (\exists\, \vec{L}, \vec{U} \in \mathbb{R}^D)[\vec{L} \preceq \vec{x} \preceq \vec{U}]\}, \tag{1}$$

with $\vec{L}$ and $\vec{U}$ as the lower and upper boundary vectors. Thus, D represents the dimensionality of the problem.

Definition 2 (Minimisation problem)**.** Let $f(\vec{x})$ be a real-valued function defined on a set $\mathfrak{X} \neq \emptyset$ such as $f(\vec{x}) : \vec{x} \in \mathfrak{X} \subseteq \mathbb{R}^D \to \mathbb{R}$. Thus, a minimisation problem is stated as

$$\vec{x}_* = \underset{\vec{x} \in \mathfrak{X}}{\arg\min} \left\{ f(\vec{x}) \right\}, \tag{2}$$

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

2.2. Problem characteristics

The characteristics of an optimisation problem relate to how difficult it is to solve. They can be classified as *a priori* and *a posteriori*. Characteristics from the former category are obtained by analysing the mathematical formulation of the problem and by visually inspecting its landscape [17]. In turn, the latter ones are obtained via specialised studies such as Exploratory Landscape Analysis (ELA). These two categories are usually nominal and ordinal ones, respectively.

In this work, we only use three representative binary features from the first class of characteristics [17, 18]. Recall Definitions 1 and 2, then, a function $f(\vec{x}) : \mathfrak{X} \in \mathbb{R}^D \to \mathbb{R}$ is said to be:

- *Differentiable* if its derivative exists at each point in its domain. So, a function is differentiable at a point $\vec{x}_i \in \mathfrak{X}$ if there is a linear function that approximates it in $\vec{x}_i$.
- *Separable* if it can be written as the sum of up to D functions, where at least one of them is a single-variable function, *i.e.*,
$$f(\vec{x}) = f_i(x_i) + g(\vec{x} \backslash x_i)$$
since $\vec{x} \backslash x_i = (x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_D)^\intercal \in \mathbb{R}^{D-1}$.
- *Unimodal* if it only possess one peak or valley in its landscape. In other words, the function only has an optimum.

There are three other common features: Continuity, Scalability, and Convexity. Nonetheless, these somehow depend of those described before. For more information about this interesting topic, we invite the reader to consult [17, 19–21]. As an illustrative example, Figure presents the two-dimensional functions for each combination of the aforementioned features. These features are binary encoded as a triplet DSU, where each letter and position refer to the Differentiability, Separability, and Unimodal features. Hence, Sphere, Rastrigin, Schwefel, Griewank, Step, Stochastic, Type-I Simple Deceptive, and Schaffer N3 belong to DSU categories 111, 110, 101, 100, 011, 010, 001, and 000, respectively. We, hereafter, use this notation to denote this triplet of features.

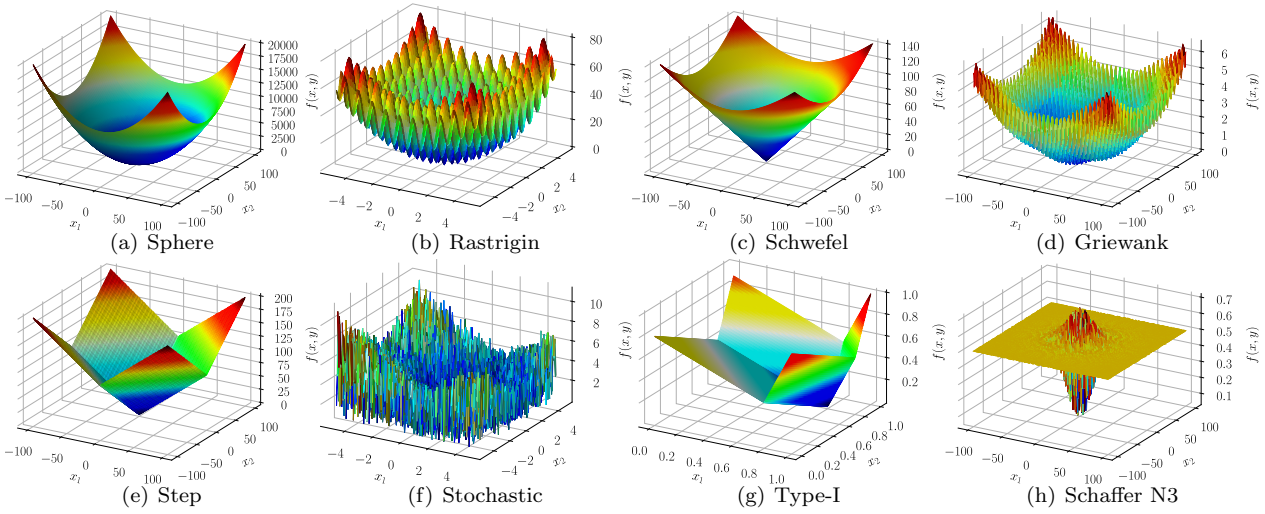


Figure 1. Illustrative example of two-dimensional benchmark functions with different categorical features, which are binary encoded as a triplet DSU (Differentiability, Separability, and Unimodal). Thus, (a) Sphere, (b) Rastrigin, (c) Schwefel, (d) Griewank, (e) Step, (f) Stochastic, (g) Type-I, and (h) Schaffer N3 belong to DSU categories 111, 110, 101, 100, 011, 010, 001, and 000, respectively.

2.3. Heuristics

A heuristic is a procedure that creates or modifies a candidate solution for a given problem instance. There are many classifications of heuristics in the literature. Most of them relate to combinatorial optimisation domains [10], whilst they are rather scarce for continuous ones [11].

In this work, we categorise continuous heuristics in three groups, extending the ideas of [10, 11]: *low-level*, *mid-level*, and *high-level*. These relate to *simple heuristics*, *metaheuristics*, and *hyper-heuristics*, respectively. Certainly, all of them are heuristics but operate under different conditions and domains. Figure 2 shows an illustrative example of the aforementioned, which is also detailed below. However, since most heuristics use a set of search agents, we first need to establish the following concepts:

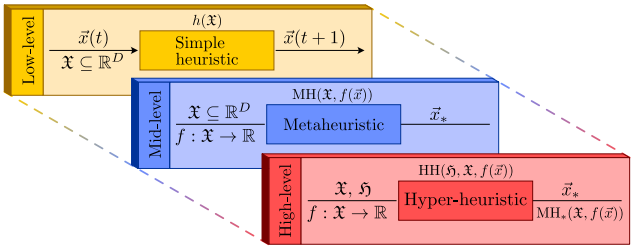


Figure 2. Heuristics of three levels interacting with the problem domain ($\mathfrak{X}$) and the heuristic space ($\mathfrak{H}$). Since $f(\vec{x})$ is the objective function, $\vec{x}(t)$ is a candidate solution at time t, and $\vec{x}_*$ is the best solution found.

Definition 3 (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 2) at time t in an iterative procedure, *i.e.,* $X(t) = \{\vec{x}_1(t), \vec{x}_2(t), \ldots, \vec{x}_N(t)\}$. Then, $\vec{x}_n(t) \in \mathfrak{X}$ denotes the n-th candidate solution or search agent of, let us say, the population $X(t)$ of size N.

Definition 4 (Best solution)**.** Let $Z(t)$ be an arbitrary set of solutions, which can be designated as, *e.g.,* the entire population $Z(t) = X(t)$, the n-th neighbourhood $Z(t) = Y_n(t)$, and the historical evolution of the n-th candidate $Z(t) = \{\vec{x}_n(0), \vec{x}_n(1), \ldots, \vec{x}_n(t)\}$. Therefore, let $\vec{x}_*(t) \in Z(t)$ be the best solution from $Z(t)$, *i.e.,* $\vec{x}_*(t) = \operatorname{argmin}\{f(Z(t))\}$.

2.4. Simple Heuristics (SHs)

They are the atomic unit in terms of search techniques that interact directly with problem domains. SHs are commonly categorised as *constructive* and *perturbative*. As their names suggest, a constructive heuristic renders new solutions from scratch while a perturbative heuristic modifies existing ones [22]. Thus, we adopt these categories adding another one, as Definition 5 describes.

Definition 5 (Simple Heuristic)**.** Let $h \in \mathfrak{H}$ be a simple heuristic from the heuristic space $\mathfrak{H}$ such that either produces, modifies or evaluates a candidate solution $\vec{x} \in \mathfrak{X}$, *cf.* Definition 1, using a fitness metric, *e.g.,* its objective function value $f(\vec{x})$, *cf.* Definition 2. Therefore, three types of simple heuristics can be stated as follows.

Remark 1 (Initialiser)**.** Let $h_i \in \mathfrak{H}_i \subset \mathfrak{H}$ be a simple heuristic that generates a solution within the search space $\vec{x} \in \mathfrak{X}$ from scratch, *i.e.,* $\vec{x} = h_i\{\mathfrak{X}\}$. Then, h_i is called initialiser. The most common one in the literature is to place the agents within the feasible search space randomly, *e.g.,* by using the uniform distribution.

Remark 2 (Search Operator)**.** Let $h_p \in \mathfrak{H}_p \subset \mathfrak{H}$ be a simple heuristic that modifies a candidate position $\vec{y} \in \mathfrak{X}$ from the current position $\vec{x} \in \mathfrak{X}$, *i.e.,* $\vec{y} = h_p\{\vec{x}, \mathfrak{X}\}$. Thus, h_p is called search operator or perturbator. In many implementations, this operator requires additional information to modify an individual, for example, the best current position in the population and positions of neighbouring agents.

Remark 3 (Selector)**.** Let $h_s \in \mathfrak{H}_s \subset \mathfrak{h}$be a simple heuristic that updates the current solution $\vec{x} \in \mathfrak{X}$ by using a candidate solution $\vec{y} \in \mathfrak{X}$, produced by a search operator, and a selection criterion *i.e.,* $\vec{x} = h_s\{\vec{t}\}$. A perturbator always precedes a selector, so one can write $\vec{x}(t+1) = (h_s \circ h_p)\{\vec{x}(t)\}$, where t refers to the step of an iterative procedure. Therefore, we assume that each search operator implicitly implements a selector. Furthermore, there are several standard selection criteria such as, *e.g.,* direct, greedy, and Metropolis update.

Remark 4 (Finaliser)**.** Let $h_f \in \mathfrak{H}_f \subset \mathfrak{H}$ be a simple heuristic that evaluates the quality of a solution, during an iterative procedure, by using information about, *e.g.,* the current solution, its fitness value, the current iteration, the previous candidate solutions, and other measurements. h_f is called Finaliser since it maps $\mathbb{R}^D$, $\mathfrak{X}$, $\mathbb{R}$ and $\mathbb{Z}_+$ to $\mathbb{Z}_2$, which corresponds to a stop flag (true or false)—a convergence check.

2.5. Metaheuristics (MHs)

They are defined as master strategies which control SHs. MHs are trendy in the literature because of their proven performance on different scenarios [2, 5]. Without loss of generality, most of the MHs follow Definition 6 which is exemplified by Figure 3.

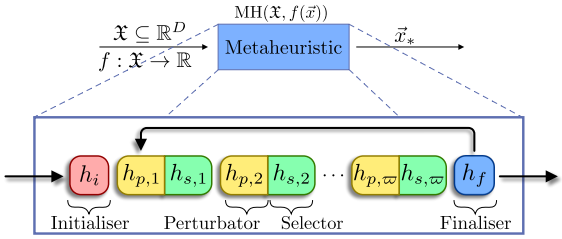


Figure 3. Scheme of a metaheuristic composed by an initialiser h_i, ϖ perturbators $h_{p,j}$ and selectors $h_{s,j}$, $\forall\ j \in \{1, \ldots, \varpi\}$, and a finaliser h_f. It is represented as a sequence of simple heuristics that iterates until a stopping flag (given by the finaliser) is raised.

Definition 6 (Metaheuristic)**.** Let MH be an iterative procedure called metaheuristic that renders an optimal solution $\vec{x}_*$ for a given optimisation problem with an objective function $f(\vec{x})$ (*cf.* Definition 2). This procedure is represented as a finite sequence of simple heuristics (*cf.* Definition 5) to be applied iteratively until a stopping condition is met, *i.e.,* $\text{MH} = (h_i, h_{p,1}, \ldots, h_{p,\varpi}, h_f) \in \mathfrak{H}_i \times \mathfrak{H}_p^{\varpi} \times \mathfrak{H}_f$.

Remark 5. (Cardinality) An inherent property of metaheuristics is the cardinality, which is defined as the number of search operators implemented in it, *i.e.,* disregarding the initialiser and finaliser; ergo $\#\text{MH} = \varpi$ (cf. Remark 2). By way of standardisation, we denote a metaheuristic and its cardinality such as MH^{ϖ}, where $\text{MH}^1 = \text{MH}$.

2.6. Hyper-heuristics (HHs)

Many researchers describe HHs as high-level heuristics controlling simple heuristics in the process of solving an optimisation problem [10]. Therefore, HHs move in the heuristic space to find a heuristic configuration that solves a given problem. With that in mind, a HH can be defined according to [11] as follows:

Definition 7 (Hyper-Heuristic)**.** Let $\boldsymbol{h} \in \mathfrak{H}^{\varpi}$ be a heuristic configuration from the heuristic space $\mathfrak{H}$ (*cf.* Definition 5), and let $F(\boldsymbol{h}|\mathfrak{X}) : \mathfrak{H}^{\varpi} \times \mathfrak{X} \to \mathbb{R}$ be its performance measure function. Recall $\mathfrak{X}$ as the problem domain in an optimisation problem with an objective function $f(\vec{x}) : \mathfrak{X} \to \mathbb{R}$ (*cf.* Definitions 1-2). Then, a solution $\vec{x}_* \in \mathfrak{X}$ and its corresponding fitness value $f(\vec{x}_*)$ are found when a $\boldsymbol{h}$ is applied on $\mathfrak{X}$, so its performance $F(\boldsymbol{h}|\mathfrak{X})$ can also be determined. Therefore, let HH be a technique that solves

$$(\boldsymbol{h}_*; \vec{x}_*) = \underset{\boldsymbol{h} \in \mathfrak{H}^{\varpi}, \vec{x} \in \mathfrak{X}}{\operatorname{argmax}} \{F(\boldsymbol{h}|\mathfrak{X})\}. \tag{3}$$

In other words, a HH searches for the optimal heuristic configuration $\boldsymbol{h}_*$ that produces the optimal solution $\vec{x}_*$ with the maximal performance $F(\boldsymbol{h}_*|\mathfrak{X})$.

Remark 6. (Heuristic Configuration) When performing a hyper-heuristic process, a heuristic configuration $\boldsymbol{h} \in \mathfrak{H}^{\varpi}$ is a way of referring to a metaheuristic MH (*cf.* Definition 6).

2.7. Selected well-known metaheuristics

We now briefly describe some common metaheuristics. We selected them for our study because they are customary and, thus, to facilitate the comparison of resulting data. More importantly, they can be described through the scheme of Figure 3.

- *Genetic Algorithm (GA)* was developed by Holland, De Jong, and others in the early ages of computing machines [23, 24]. This algorithm comprises a set of operators inspired on the evolutionary mechanisms from genes of organisms [6]. GA was initially implemented for solving combinatorial problems by using a chromosome-like encoding, but it was later employed in continuous optimisation [6].

- *Simulated Annealing (SA)* was introduced by Kirkpatrick *et al.* for solving combinatorial optimisation problems [25], but it has been also implemented in other domains [26, 27]. It was the first metaheuristic inspired in a nonlinear physical process, the annealing: thermal treatment of materials to promote its recrystallisation [28].

- *Differential Evolution (DE)* was originally proposed by Price and Storn for solving global optimisation problems [29]. DE is a stochastic population-based direct search algorithm whose performance depends mainly on the chosen mutation and crossover strategies [7, 30].

- *Particle Swarm Optimisation (PSO)* was conceived by Kennedy and Eberhart as a population-based methodology that explodes social and individual interaction between agents [31]. Particles in the swarm have two basic components, position and velocity, which are updated each iteration via a basic kinematic expression [32].

- *Firefly Algorithm (FA)* was proposed by Yang as a swarm intelligence methodology [33]. FA posses a quite simple search procedure inspired in the way fireflies mate [34].

- *Cuckoo Search (CS)* was introduced by [8] as a methodology to exploit some features from metaheuristics, such as swarm intelligence and random walk through Lévy flight. CS mimics brood parasitism behaviour of certain cuckoo species, which hide their eggs inside alien nests [35, 36].

- *Central Force Optimisation (CFO)* was conceived by Formato as a deterministic (gradient-like) optimisation strategy [37]. CFO seeks a solution using Newtonian gravitational motion equations [38–40].

- *Spiral Optimisation Algorithm (SOA)* was proposed by Tamura and Yasuda as a direct-solving metaheuristic procedure based on the logarithmic spiral dynamic [41]. Recently, it was proposed a general version of SOA called Stochastic Spiral Optimisation Algorithm (SSOA) to include random variations to overcome the slow convergence of SOA [42].

- *Gravitational Search Algorithm (GSA)* was presented by Rashedi *et al.* as a technique based on the Newtonian gravity and motion laws [43]. GSA incorporates foundations from PSO and CFO giving place to a metaheuristic that uses the velocity and acceleration of agents to assess their position [44].

3. Methodology

In this work, all the experiments were carried out in Python 3.7.6 running on 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. These experiments followed a three-stage methodology to test our approach for tailoring metaheuristics (MHs).

As a first stage (Figure 4) we extracted the Search Operators (SOs) from the following 10 well-known MHs: Random Search (RS) [45], Simulated Anneling (SA) [25], Genetic Algorithm (GA) [6], Cuckoo Search (CS)[8], Differential Evolution (DE) [30], Particle Swarm Optimisation (PSO)[31, 32], Firefly Algorithm (FA) [33], Stochastic Spiral Optimisation Algorithm (SSOA) [42], Central Force Optimisation (CFO) [37], and Gravitational Search Algorithm (GSA)[44]. We extracted such operators via the MH scheme described in Definition 6. For each one of them, we also identified their controlling parameters as well as their ranges and default values. Besides, we identified diverse selection mechanisms (*i.e.,* selectors) that those metaheuristics implement. Afterward, we generated two collections of SOs that serve as heuristic spaces. The first one

constitutes a database of SOs from the initial 10 MHs where we only vary the most relevant parameters, and the others were set to default values. We considered a parameter as relevant if its value dramatically changes the behaviour of the operator, such that the SO could be regarded as different from the others. For example, DE mutation mechanism can be implemented using an expression of either *rand*, *best*, *rand-to-best*, so forth; thus, each variant is considered as a search operator. The second database comprises an expanded SOs database by employing 21 different value combinations for each parameter of each SO from the first database. Furthermore, by using the first collection, we build a database with predefined MHs, each one with its default selector, for comparison purposes. These MHs are considered as basic ones and their cardinalities are amongst one and two.

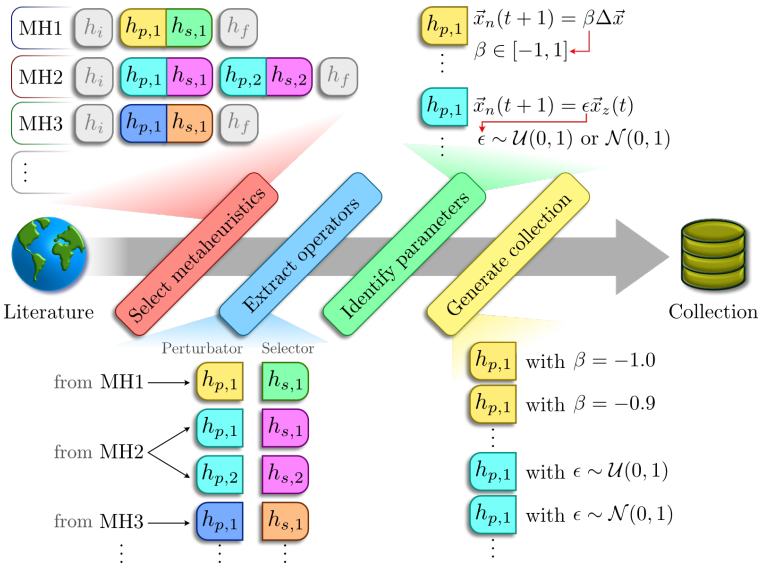


Figure 4. First stage of the Methodology carried out in this work, which consists of the search operator collection.

For the remaining stages, we first selected and implemented a collection of 107 multidimensional benchmark functions commonly used in the literature [17, 21, 46] as problem domains. Moreover, we categorised these problems by using their qualitative characteristics such as Differentiability, Separability, and Unimodality (*cf.* Section 2.2) in eight triads of binary-encoded features, *i.e.,* DSU. Table 1 summarises this collection where function groups are organised in ascending order of hardness. So, those functions considered as the easiest ones belong to the category 111, which means they are differentiable, separable, and unimodal problems, *e.g.,* the well-known, Sphere 8(De Jong). In turn, the category 000 corresponds to the hardest problems which are non-differentiable, non-separable, and multimodal.

In the second stage, we carried out a brute-force experiment utilising the first collection of search operators to solve each one of the benchmark problems. Besides, different number of dimensions were considered such as 2, 5, 10, 20, 30, 40 and 50. Each procedure was repeated 30 times to ensure statistical significance. Plus, each search operator employed a population size of 30 and a fixed number of iterations of 100. Results from this stage were analysed in order to find relevant insights about solvers and problems. Subsequently, a similar procedure was carried out but using the database of basic metaheuristics.

Finally, we implemented a Simulated Annealing (SA) approach to tune the HH to customise MHs for solving the same benchmark problems from the previous stage. This SA-based HH method required the hyper-parameters of SA (*i.e.,* initial temperature Θ_0, cooling rate δ, maximum number of steps $t_{\max}$, and maximum stagnation count $s_{\max}$), the problem information (*i.e.,* domain $\mathfrak{X}$ and objective function $f(\vec{x})$), and the heuristic space $\mathfrak{H}$ (collection of search operators). Moreover, we set a maximum cardinality $n_{\max}$ for the metaheuristics to avoid producing solvers with high computing burden. The heuristic space and maximum cardinality were defined according to three cases of study; *i.e.,* the short collection with $n_{\max}$ of three and five, and the long collection with $n_{\max}$ of three. The former cardinality value was chosen chiefly because MHs available in literature usually have at most three search operations. The latter can represent those methods with complex search mechanics, but without an excessively increased computing burden, such as

Table 1. Set of standard benchmark functions used in this work. Third column corresponds to the encoded for the categorical features of these functions. DSU stands for Differentiable, Separable, and Unimodal. Naturally, '1' and '0' mean True and False.

	Id.	Function Name	Ref.
Category 111	10	Cigar	[47]
	22	Ellipsoid	[48]
	33	Hyper-Ellipsoid	[49]
	53	Powel Sum	[17]
	56	Quartic (noiseless)	[17]
	62	Rotated-Hyper-Ellipsoid	[50]
	73	Schumer Steiglitz	[46]
	80	Schwefel 2.23	[17]
	83	Sphere (De Jong)	[17]
	91	Sum Squares	[17]
Category 110	4	Alpine 2	[17]
	5	Bohachevsky	[47]
	16	Deb 1	[17]
	17	Deb 2	[46]
	30	Giunta	[17]
	41	Michalewicz	[50]
	55	Qing	[17]
	57	Quintic	[17]
	59	Rastrigin	[17]
	70	Schubert	[17]
	71	Schubert 3	[17]
	72	Schubert 4	[17]
	81	Schwefel 2.25	[17]
	90	Styblinski-Tank	[51]
	98	W-Wavy	[17]
	99	Weierstrass	[17]
	105	Yao-Liu 09	[47]
Category 101	6	Brent	[17]
	7	Brown	[17]
	9	Chung Reynolds	[46]
	19	Dixon-Price	[17]
	28	Exponential	[17]
	36	K-Tablet	[52]
	43	Mishra 2	[17]
	50	Perm 01	[47]
	51	Perm 02	[50]
	60	Ridge	[17]
	74	Schwefel	[17]
	75	Schwefel 1.2	[17]
	76	Schwefel 2.04	[17]
	92	Trid	[50]
	103	Xin-She Yang 3	[17]
	106	Zakharov	[17]

	Id.	Function Name	
Category 100	1	Ackley 1	[17]
	2	Ackley 4	[17]
	18	Deflected Corrugated Spring	[46]
	20	Drop Wave	[51]
	21	Egg Holder	[17]
	29	F2	[46]
	31	Griewank	[17]
	32	Happy Cat	[51]
	34	Inverted Cosine-Wave	[46]
	35	Jennrich-Sampson	[46]
	40	Lunacek N02	[46]
	42	Mishra 1	[17]
	44	Mishra 7	[17]
	48	Pathological	[17]
	49	Periodic	[17]
	52	Pinter	[17]
	58	Rana	[17]
	61	Rosenbrock	[17]
	63	Salomon	[17]
	64	Sargan	[17]
	65	Schaffer N1	[51]
	66	Schaffer N2	[51]
	69	Schaffer N6	[17]
	89	Streched V Sine Wave	[17]
	93	Trigonometric 1	[17]
	94	Trigonometric 2	[17]
	97	Vincent	[47]
	100	Whitley	[17]
Category 011	77	Schwefel 2.20	[17]
	78	Schwefel 2.21	[17]
	84	Step	[17]
	85	Step 2	[17]
	86	Step 3	[17]
	87	Step Int	[17]

	Id.	Function Name	
Category 010	3	Alpine 1	[17]
	11	Cosine Mixture	[46]
	15	Csendes	[17]
	25	Expanded Five-Uneven-Peak Trap	[21]
	26	Expanded Two-Peak Trap	[21]
	54	Price 01	[17]
	82	Schwefel 2.26	[17]
	88	Stochastic	[46]
	101	Xin-She Yang 1	[17]
001	79	Schwefel 2.22	[17]
	95	Type-I Simple Deceptive Problem	[53]
Category 000	8	Carrom Table	[47]
	12	Cross-in-Tray	[47]
	13	Cross-Leg Table	[46]
	14	Crowned Cross	[47]
	23	Expanded Decreasing Minima	[21]
	24	Expanded Equal Minima	[21]
	27	Expanded Uneven Minima	[21]
	37	Katsuura	[47]
	38	Levy	[50]
	39	Lunacek N01	[46]
	45	Mishra 11	[17]
	46	Modified Vincent	[21]
	47	Needle-Eye	[46]
	67	Schaffer N3	[51]
	68	Schaffer N4	[51]
	96	Type-II Medium-Complex Deceptive	[53]
	102	Xin-She Yang 2	[17]
	104	Xin-She Yang 4	[17]
	107	Zero Sum	[47]

hybrid methods. Furthermore, we carried out similar experiments as in the brute-force study but employing the basic metaheuristic collection for comparison purposes. Hyper-parameters of SA-based HH were fixed as $\Theta_0 = 200$, $\delta = 0.05$, $t_{\max} = 100$, and $s_{\max} = 30$ for all the experiments performed in this stage. In this work, all metaheuristics were built considering the same population size of 30, the uniform random initialisation, and a maximum number of iterations of 100 as stopping criterion. Also, each MH internally works in a normalised domain, between -1 and 1, of the original one $\mathfrak{X} \in \mathbb{R}^D$ with arbitrary boundaries, *i.e.,* $\mathfrak{X} \to [-1, 1]^D$.

Pseudocode 1 summarises the Hyper-heuristic approach based on Simulated Annealing. We dedicated the following lines of this section to detail the most relevant parts of such an approach. So, SA-based HH starts by generating from scratch a candidate solution $\boldsymbol{h}$ of $\varpi = \#\boldsymbol{h} = 1$ by randomly choosing a search operator from the heuristic space $\mathfrak{H}$. Random selection is weighted by an *a posteriori* probability distribution $P_{\mathfrak{H}}(h)$. In the simplest implementation, this distribution is just the uniform one. Then, the sequence of search operators is used to build a metaheuristic MH^{ϖ} according to Definition 6. Subsequently, to assess the performance of each metaheuristic MH^{ϖ} when solving a given problem, we ran each candidate $N_r = 30$ times and recorded all fitness value. Then, we determined its performance, such as

$$F(\mathrm{MH}^{\varpi}|\mathfrak{X}) = (\mathrm{med} + \mathrm{iqr}) \left(\bigcup_{r=1}^{N_r} f(\vec{x}_{r,*}) \right), \tag{4}$$

where med and iqr are the median and interquartile range operators applied to the fitness values $f(\vec{x}_{r,*})$.

Afterwards, the SA-based HH uses a procedure for generating neighbour heuristic sequences described in Pseudocode 3 based on [54, 55]. There, a neighbour $\boldsymbol{h}_c$ of a heuristic sequence $\boldsymbol{h}$ can be obtained by adding, deleting, or perturbing a heuristic at random. When addition or perturbation actions are selected, candidate heuristics are extracted from the collection of search operators. Later, the neighbour or candidate metaheuristic is built MH_c^n and evaluated $F(\mathrm{MH}_c^n|\mathfrak{X})$ as before. Subsequent steps in Pseudocode 1 correspond to the common procedure of the Simulated Annealing method.

Pseudocode 1. Hyper-heuristic based on Simulated Annealing for tailoring metaheuristics

Input: Initial temperature $\Theta_0 \in \mathbb{R}_+$, cooling rate $\delta \in]0,1[$, maximum number of steps $t_{\max}$, problem domain $\mathfrak{X} \subseteq \mathbb{R}^D$, objective function $f(\vec{x})$, heuristic space $\mathfrak{H}$, initialiser h_i, finaliser h_f, and maximum cardinality $\varpi_{\max} \geq 1$
Output: Best metaheuristic MH_*^{ϖ} found

$\boldsymbol{h} \leftarrow$ ChooseRandomly($\mathfrak{H}$, $P_{\mathfrak{H}}(h)$) ▷ $P_{\mathfrak{H}}(h)$ is the probability distribution of $\mathfrak{H}$
$\mathrm{MH}^{\varpi} \leftarrow \{h_i, \boldsymbol{h}, h_f\}$ with $\varpi = \#\boldsymbol{h} = 1$
Evaluate MH^{ϖ} via $F(\mathrm{MH}^{\varpi}|\mathfrak{X})$ in (4) and using Pseudocode 2
Initialise $t \leftarrow 0$ and $\mathrm{MH}_*^{\varpi} \leftarrow \mathrm{MH}^{\varpi}$
while $(t \leq t_{\max})$ and $(s \leq s_{\max})$ **do**
    $\boldsymbol{h}_c \leftarrow$ GetNeighbour($\boldsymbol{h}$, $\mathfrak{H}$, $\varpi_{\max}$) and $u_r \sim \mathcal{U}(0,1)$ ▷ Pseudocode 3
    $\mathrm{MH}_c^{\varpi} \leftarrow \{h_i, \boldsymbol{h}_c, h_f\}$ with $\varpi = \#\boldsymbol{h}_c$
    Evaluate MH_c^{ϖ} via $F(\mathrm{MH}_c^{\varpi}|\mathfrak{X})$ in (4) and using Pseudocode 2
    $\Delta E \leftarrow F(\mathrm{MH}_c^{\varpi}|\mathfrak{X}) - F(\mathrm{MH}^{\varpi}|\mathfrak{X})$ and $\Theta \leftarrow \Theta(1-\delta)$
    if $u_r \leq \exp(-\Delta E/\Theta)$, **then** $\mathrm{MH}^{\varpi} \leftarrow \mathrm{MH}_c^{\varpi}$, **end if**
    if $F(\mathrm{MH}_c^{\varpi}|\mathfrak{X}) < F(\mathrm{MH}_*^{\varpi}|\mathfrak{X})$, **then** $\mathrm{MH}_*^{\varpi} \leftarrow \mathrm{MH}^{\varpi}$ and $s \leftarrow 0$, **else** $s \leftarrow s+1$ **end if**
    $t \leftarrow t+1$
end while

Pseudocode 2. Metaheuristic scheme

Input: Problem domain $\mathfrak{X} \subseteq \mathbb{R}^D$, objective function $f(\vec{x})$, search operators $\boldsymbol{h}_p \in \mathfrak{H}_p^{\varpi}$, selector configuration $\boldsymbol{h}_s \in \mathfrak{H}_s^{\varpi}$ initialiser $h_i \in \mathfrak{H}_i$, finaliser $h_f \in \mathfrak{H}_f$, and population size N.
Output: Optimal solution $\vec{x}_*$

Initialise the population positions $X(0) \ni \vec{x}_n(0) \leftarrow h_i\{\mathfrak{X}\}, \forall\ n \in \{1,\ldots,N\}$
Initialise additional features for each individual in the population
Evaluate the population $f_n(0) \leftarrow f(\vec{x}_n(0)),\ \forall\ n \in \{1,\ldots,N\}$
Find $\vec{x}_*(0) \leftarrow \vec{x}_k(0)$ since $k = \mathrm{argmin}\{f_1,\ldots,f_N\}$, and set $t \leftarrow 0$
repeat
    for $m = \{1,\ldots,\varpi\}$ **do** ▷ $\varpi = \#\boldsymbol{h}_p = \#\boldsymbol{h}_s$
        Apply the m-th search operator and selector, *i.e.,* $X(t) = (h_{s,m} \circ h_{p,m})\{X(t)\}$
        Update $f_n(t),\ \forall\ n \in \{1,\ldots,N\}$, $\vec{x}_*(t)$, and the additional population features
    end for
    $t \leftarrow t+1$
until $h_f\{t, \mathfrak{X}, \vec{x}_*(t), f(\vec{x}_*(t)), \ldots\} ==$ `True`

4. Results

4.1. Search operators

Table 2 summarises the search operators collected in this stage from the aforementioned MHs: RS, SA, GA, CS, DE, PSO, FA, SSOA, CFO, and GSA. We also included the random sample as is the most straightforward manner of performing a search in an arbitrary domain. In this table, we present the operators' name, the corresponding definitions where they are detailed, their control parameters and default selector. We classified

Pseudocode 3. Procedure for generating a neighbour hyper-heuristic

procedure GETNEIGHBOUR($\boldsymbol{h}$, $\mathfrak{H}$, $\varpi_{\max}$)
  if $\varpi \geq \varpi_{\max}$, **then** `ActionSet` $\leftarrow \{\texttt{Del}, \texttt{Per}\}$ ▷ $\varpi = \#\boldsymbol{h}$
  else if $\varpi \leq 1$, **then** `ActionSet` $\leftarrow \{\texttt{Add}, \texttt{Per}\}$
  else `ActionSet` $\leftarrow \{\texttt{Add}, \texttt{Del}, \texttt{Per}\}$, **end if**
  `Action` $\leftarrow$ CHOOSERANDOMLY(`ActionSet`)
  if `Action` == `Add` **then**
    $h_a \leftarrow$ CHOOSERANDOMLY($\mathfrak{H} \backslash \boldsymbol{h}$)
    $k \sim \mathcal{U}\{1, \varpi + 1\}$
    $\boldsymbol{h}' \leftarrow \left(\cup_{i=1}^{k-1} h_i\right) \cup h_a \cup \left(\cup_{j=k}^{\varpi} h_j\right)$, $\forall\, h_i, h_j \in \boldsymbol{h}$
  else if `Action` == `Del` **then**
    $h_d \leftarrow$ CHOOSERANDOMLY($\mathfrak{H}$)
    $\boldsymbol{h}' \leftarrow \boldsymbol{h} \backslash h_d$
  else
    $h_p \leftarrow$ CHOOSERANDOMLY($\mathfrak{H} \backslash \boldsymbol{h}$)
    $k \sim \mathcal{U}\{1, \varpi\}$ and $\boldsymbol{h}' \leftarrow \boldsymbol{h}$
    $h'_k \leftarrow h_p : h'_k \in \boldsymbol{h}'$
  end if
  return $\boldsymbol{h}'$
end procedure

Table 2. Search operators extracted from 10 well-known metaheuristics in the literature. Values or ranges for variation and tuning parameters, as well as selectors, are chosen from those commonly employed in the literature.

Def.	Operator name	Variation parameters	Tuning parameters	Selector[a]
8	Random sample	$\vec{r} \ni r_i \sim \mathcal{U}(-1, 1)$	$\alpha \in]0, 1]$	Direct
9	Random search	$\vec{r} \ni r_i \sim \mathcal{U}(-1, 1)$	$\alpha \in]0, 1]$	Greedy
10	Local random walk	$\vec{r} \ni r_i \sim \mathcal{U}(0, 1)$	$\alpha \in]0, 1]$, $p \in]0, 1]$	Greedy
11	Random flight	$\vec{r} \ni r_i \sim \mathcal{L}(1.5)$	$\alpha \in]0, 1]$	Greedy
12	Genetic crossover	Pairing scheme[b], Crossover mechanism[c]	$m_p \in]0, 1]$	Direct
13	Genetic mutation	$\vec{r} \ni r_i \sim \mathcal{U}(-1, 1)$	$\alpha \in]0, 1]$, $p_e \in [0, 1]$, $p_m \in]0, 1]$	Greedy
14	Differential mutation	Mutation scheme	$M \in \{1, 2, 3\}$, $\alpha_m \in]0, 3[\ \forall\, m \in \{0, M\}$	Greedy
15	Swarm dynamic	Velocity approach, $\vec{r}_i \ni r_{i,j} \sim \mathcal{U}(0, 1)\, \forall i \in \{1, 2\}$	$\alpha_0 \in]0, 1[$, $\phi_1, \phi_2 \in]0, 4]$, $\kappa \in [0, 1]$	Direct
16	Firefly dynamic	$\vec{r} \ni r_i \sim \mathcal{U}(-0.5, 0.5)$	$\alpha_0, \alpha_1 \in [0, 1]$, $\alpha_2 \in]0, 1000]$	Direct
17	Spiral dynamic	–	$r_0 \in]0, 1[$, $\theta \in]0°, 360°[$, $\sigma_r \in [0, 1[$	Direct
18	Central force dynamic	–	$\alpha_0, \alpha_1 \in]0, 0.01]$, $\alpha_2 \in]1, 2[$	Direct
19	Gravitational search	–	$\alpha_0 \in]0, 1]$, $\alpha_1 \in]0, 0.1]$	Direct

[a] Feasible selectors: direct, greedy, probabilistic, and Metropolis.
[b] 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 $p_T \in]0, 1]$.
[c] Possible mechanisms: single-point, two-points, uniform, blend, and linear crossover. Linear crossover requires two additional parameters such as $\beta_1, \beta_2 \in \mathbb{R}$.

these parameters as variation and tuning ones. The first one concerns those parameters that can dramatically change the behaviour of the operator. The second one those that refine the search procedure.

The remaining of this section details the search operators as well as the selectors in Table 2, using the same nomenclature and terminology of Sect. 2. These operators were grouped in four categories according to their source of inspiration or the behaviour that they model. Moreover, they are organised according to the sophistication level of their mathematical formulation, which is not necessarily related to the numerical complexity. For example, random-based operators such as random sample and random walk are straightforward since they represent a change in position of the agent. But, as we go deeper into this section, we find dynamics-based operators that imply several and more elaborate operations. For example, in the gravitational search dynamic a change in position (for the agent) depends on a velocity calculation that relies on

the estimate of an acceleration value.

First of all, 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. Bearing this in mind, the collected search operators are presented for determining the n-th candidate solution $\vec{y}_n \in \mathfrak{X}$ in the problem domain $\mathfrak{X} \in \mathbb{R}^D$. Such operators are expressed in vector form. So, most of them require the Hadamard–Schur's product $\odot$ and the element-wise Heaviside step function $H : \mathbb{R}^D \to \mathbb{Z}_2^D$ with $H(0) = 1$.

This first group comprises the simplest search operators 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 [56]. Many of them are used by more sophisticated algorithms, as we detail next. Nonetheless, we start by defining the simplest one, *i.e.,* random sample [45]. Then, we continue with one a bit more sophisticated, though also simple, which leads countless implementations in numerical methods, *i.e.,* random search. Lastly, we conclude this group with a variant of the aforementioned operator, which has the essential structure of many modern search operators.

Definition 8 (Random sample)**.** A candidate solution is obtained through a random sample operation such as

$$\vec{y}_n = \vec{r}, \tag{5}$$

where $\vec{r} \in \mathbb{R}^D$ is a vector of i.i.d. random numbers with $\vec{r} \ni r_i \sim \mathcal{U}(-1, 1)$.

Remark 7 (Random samples with different distributions)**.** Instead of the uniform one, another probability distribution can be used, but the domain boundaries must be preserved. However, we only use the uniform distribution function for this work.

Definition 9 (Random search)**.** A candidate solution is determined by a random search operation as shown

$$\vec{y}_n = \vec{x}_n(t) + \alpha\vec{r}, \tag{6}$$

where $\alpha \in]0, 1]$ is the step size factor and $\vec{r} \in \mathbb{R}^D$ is a vector of i.i.d. random numbers with either $\mathcal{U}(-1, 1)$ or other probability distribution, such as the normal standard $\mathcal{N}(0, 1)$ and symmetric Lévy stable $\mathcal{L}(1.5)$ ones.

Definition 10 (Local random walk)**.** A candidate solution is achieved via a local random walk such as

$$\vec{y}_n = \vec{x}_n(t) + \alpha\vec{r} \odot H(p - \vec{q}) \odot (\vec{x}_{z_1}(t) - \vec{x}_{z_2}(t)), \tag{7}$$

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

Remark 8 (Considerations about the local random walk)**.** In the strict-sense, local search implies looking for a candidate solution close to the current one, *i.e.,* a neighbour. So, for continuous domains, it is meaningful to use $\vec{r} \ni r_i \sim \mathcal{N}(0, 1)$ instead of $\mathcal{U}(0, 1)$. However, in the wide-sense, the uniform distribution function is popular for this task as we used it in this work. The range of this vicinity is given by the individuals $\vec{x}_{z_1}(t)$ and $\vec{x}_{z_2}(t)$. It is common to randomly choose them from the population, *i.e.,* $z_1, z_2 \sim \mathcal{U}\{1, N\}$ and $z_1 \neq z_2$. Although, it is not infrequent that some authors set them as the best current individual and the current one to perform a guided search. This variant is explained in the Definition 11. Withal, when the probability value is $p = 1.0$, this search operator corresponds to that one used by Simulated Annealing (SA) [27]. Otherwise, when $p \in]0, 1[$, it can be found in the Cuckoo Search Algorithm (CSA) implementations as the eggs' detection mechanism [8].

Definition 11 (Random flight)**.** A candidate solution found by employing a random flight operation as given

$$\vec{y}_n = \vec{x}_n(t) + \alpha\vec{r} \odot (\vec{x}_n(t) - \vec{x}_*(t)), \tag{8}$$

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. The latter can be found by implementing the Mantegna-Stanley's algorithm [57].

The second category of operators obeys those inspired on evolutionary processes, which conform two well-known metaheuristics such as Genetic Algorithm (GA) [23] and Differential Evolution (DE) [29].

Definition 12 (Genetic crossover)**.** This operator works on a ranked version of the population $\hat{X}(t)$ w.r.t. the cost function value *i.e.,* $\hat{X}(t) = \{\hat{\vec{x}}_1, \ldots, \hat{\vec{x}}_N\}$ with $f(\hat{\vec{x}}_1) < \cdots < f(\hat{\vec{x}}_N)$. Such preliminary adjustment is called by many authors as natural selection [6]. Subsequently, a candidate position is rendered by a genetic crossover operation through

$$\vec{y}_n = \vec{m} \odot \hat{\vec{x}}_{z_1} + (1 - \vec{m}) \odot \hat{\vec{x}}_{z_2}, \quad \forall\, z_1, z_2 \in \{1, \ldots, M\}, n \in \{M+1, \ldots, N\}, \tag{9}$$

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, $M = \lfloor m_p N \rceil$ 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, p_1 and p_2 refer to the parent indices chosen by using a pairing scheme from the mating pool. Consider that $p \sim \mathcal{U}_P\{a, b\}$ indicates that p is a positive integer randomly chosen between a and b, with $0 < a < b$, by sampling a given probability distribution $P(q) : \mathbb{Z}_{++} \ni [a, b] \to [0, 1] \in \mathbb{R}_+$; when $P(q)$ is unspecified, it is assumed the uniform distribution, *i.e.,* $P(q) = 1/(b - a + 1)$. Therefore, the most common pairing schema found in the literature are listed below [6, 23].

- Random pairing: $z_i \sim \mathcal{U}\{1, M\}$, $\forall\, i \in \{1, 2\}$.
- Rank weighting pairing: $z_i \sim \mathcal{U}_{P_R}\{1, M\}$ with $P_R(m) = 2(M + 1 - m)/(M + 1)M$ as the rank-based weight distribution, $\forall\, i \in \{1, 2\}$ and $m \in \{1, \ldots, M\}$.
- Roulette wheel pairing: $z_i \sim \mathcal{U}_{P_C}\{1, M\}$ with $P_C(m) = |f(\hat{\vec{x}}_m) - f(\hat{\vec{x}}_{M+1})|/|\sum_{k=1}^{M} f(\hat{\vec{x}}_k)|$ as the cost-based weight distribution, $\forall\, i \in \{1, 2\}$ and $m \in \{1, \ldots, M\}$.
- Tournament pairing: $z_i = w_k \in W_T$ with $k \sim \mathcal{U}_{P_T}\{1, M_T\}$, $P_T(k) = p_T(1 - p_T)^k$ as the probability of victory, $W_T = \{w_j \sim \mathcal{U}\{1, M\} | (\forall\, j = 1, \ldots, M_T) \wedge (w_1 < \cdots < w_{M_T})\}$, $\forall\, i \in \{1, 2\}$, $k \in \{1, \ldots, M_T\}$, and $p_T \in]0, 1]$.

On the other hand, the mask vector is commonly assumed in the range $[0, 1]^D$, as we do in this work. But, some researchers incorporate values beyond such a range. This vector can be determined via diverse crossover mechanisms [6]. The most common ones are:

- Single-point crossover: $\vec{m} = H(d_1 - \vec{s})$ with $d_1 \sim \mathcal{U}\{1, D\}$ and $\vec{s} = (1, 2, \ldots, D)^\intercal$.
- Two-points crossover: $\vec{m} = H(d_1 - \vec{s}) - H(d_2 - \vec{s})$ with $d_1, d_2 \sim \mathcal{U}\{1, D\}$ and $d_1 < d_2$, and $\vec{s} = (1, 2, \ldots, D)^\intercal$.
- Uniform crossover: $\vec{m} = H(\vec{r} - 0.5)$ with $\vec{r} \ni r_i \sim \mathcal{U}(0, 1)$.
- Blend crossover: $\vec{m} = \vec{r}$ with $\vec{r} \ni r_i \sim \mathcal{U}(0, 1)$.
- Linear crossover: $\vec{m} = \beta_1 \vec{1}$ and $1 - \vec{m} = \beta_2 \vec{1}$ with $\beta_1, \beta_2 \in \mathbb{R}$ and $\vec{1} = (1, 1, \ldots, 1) \in \mathbb{Z}_1^D$.

Definition 13 (Genetic mutation)**.** This operator also works on a ranked version of the population $\hat{X}(t)$ w.r.t. the cost function value *i.e.,* $\hat{X}(t) = \{\hat{\vec{x}}_1, \ldots, \hat{\vec{x}}_N\}$ with $f(\hat{\vec{x}}_1) < \cdots < f(\hat{\vec{x}}_N)$. Hence, the candidate position is found by implementing a genetic mutation such as

$$\vec{y}_n = \vec{m} \odot \vec{x}_n(t) + \alpha(1 - \vec{m}) \odot \vec{r}, \quad \forall\, n \in \{\lceil p_e N \rceil, \ldots, N\}, \tag{11}$$

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 $\mathcal{U}(-1, 1)$ 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 mutating agents in continuous optimisation [6]. Some feasible options are the normal standard distribution $\vec{r} \ni r_i \sim \mathcal{N}(0, 1)$ and the symmetric Lévy stable distribution $\vec{r} \ni r_i \sim \mathcal{L}(1.5)$.

Definition 14 (Differential mutation)**.** The candidate position is determined by using the differential mutation operator as displayed

$$\vec{y}_n = \vec{m} + \sum_{m=1}^{M} \alpha_m \left(\vec{x}_{z_{2m-1}}(t) - \vec{x}_{z_{2m}}(t) \right), \tag{12}$$

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 $\forall\, m \in \{0, \ldots, M\}$. 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 representative schema are listed as follows [30]:

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

Remark 9 (Differential crossover)**.** Differential mutation is always succeeded by a crossover operation. However, such an operation showed a poor performance when implemented apart from the mutation operator. For that reason, we opted for disregarding it from the selected search operators. However, for comparison purposes, we required this crossover mechanism when implementing the basic version of DE (see Sect. 3). So, we implemented the crossover operator described as follows. Let $\vec{x}_n(t) \in X(t)$ and $\vec{y}_n \in \mathfrak{X}$ be the current and candidate positions for the n-th individual, respectively. Then, the candidate position is updated by performing the crossover operation given by

$$\vec{y}_n = \vec{m} \odot \vec{y}_n + (1 - \vec{m}) \odot \vec{x}_n(t), \tag{13}$$

where $\odot$ is the Hadamard-Schur's product, and $\vec{m} \in \mathbb{Z}_2^D$ is the crossover mask vector calculated such as $\vec{m} = H(p_{\mathrm{CR}} - \vec{r})$ by employing $H : \mathbb{R}^D \to \mathbb{Z}_2^D$ as the element-wise Heaviside step function with $H(0) = 1$, $\vec{r} \in \mathbb{R}^D$ is a vector of i.i.d. uniformly random variables in $[0, 1]$, $\vec{r} \ni r_i \sim \mathcal{U}(0, 1)$, and p_{CR} is the crossover probability that depends of the constant $\mathrm{CR} \in]0, 1]$. Such a probability indicates if an element mutates, and its formula varies according to the crossover type [58]. For binomial crossover, this probability is found as

$$p_{\mathrm{CR}} = \mathrm{CR}\left(1 - \frac{1}{D}\right) + \frac{1}{D}, \tag{14a}$$

and for exponential crossover, it is

$$p_{\mathrm{CR}} = \frac{1 - \mathrm{CR}^D}{D(1 - \mathrm{CR})}. \tag{14b}$$

In this work, we employed the binomial crossover operator for building the DE basic version.

Subsequently, the third group of operators are those inspired on the idealised social behaviour of many living creatures [59]. Therefore, we extracted the operators of two well-known metaheuristics as Particle Swarm Optimisation (PSO) [31, 32] and Firefly Algorithm (FA) [33, 34].

Definition 15 (Swarm dynamic)**.** The candidate position for the n-th agent is rendered via the swarm dynamic operation, such as

$$\vec{y}_n = \vec{x}_n(t) + \vec{v}_n(t), \tag{15}$$

where $\vec{v}_n(t)$ corresponds to the velocity of an individual. This can be determined through different approaches. Literature is prolific with PSO variants that use slight or radical changes of the velocity calculation. However, the most common and simplest ones, widely implemented in several practical problems, are the inertial and constricted approaches, which follow the expression:

$$\vec{v}_n(t) = \alpha_0 \vec{v}_n(t-1) + \alpha_1 \phi_1 \vec{r}_1 \odot (\vec{x}_{n,*}(t) - \vec{x}_n(t)) + \alpha_2 \phi_2 \vec{r}_2 \odot (\vec{x}_*(t) - \vec{x}_n(t)) \tag{16}$$

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 $\mathcal{U}(0, 1)$, $\forall\ j \in \{1, 2\}$ [60]. Moreover, $\vec{v}_n(t-1) \in V(t-1)$ 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 operator 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 $n \in \{1, \ldots, N\}$.

- Inertial approach: It employs $\alpha_0 \in]0,1[$, $\alpha_2 = \alpha_3 = 1$. In this approach α_0 is called inertial weight and can be used as a fixed parameter, a time-dependent function, or a uniform random variable [61, 62].
- Constrained approach: $\alpha_0 = \alpha_1 = \alpha_2 = \chi$, where χ is known as the constriction factor determined by

$$\chi = \frac{2\kappa H(\phi - 4)}{\phi - 2 - \sqrt{\phi(\phi - 4)}} + \sqrt{\kappa}\,(1 - H(\phi - 4)) \tag{17}$$

since $H : \mathbb{R} \to \mathbb{Z}_2$ is the Heaviside step function with $H(0) = 0$, $\kappa \in]0,1]$, and $\phi = \phi_1 + \phi_2$ [32, 61].

Definition 16 (Firefly dynamic)**.** The candidate solution obtained via the firefly dynamic operator is shown as

$$\vec{y}_n = \vec{x}_n(t) + \alpha_0 \vec{r} + \alpha_1 \sum_{\substack{k=1 \\ k \neq n}}^{N} H(-(I_k - I_n))(\vec{x}_k(t) - \vec{x}_n(t)) e^{-\alpha_2 ||\vec{x}_k(t) - \vec{x}_n(t)||_2^2}, \tag{18}$$

where $\alpha_0, \alpha_1 \in]0,1]$ and $\alpha_2 \in]0,1000]$ are constant factors, $||\cdot||_2$ is the ℓ^2-norm, $\vec{r}$ is a vector of i.i.d. random variables with $\mathcal{U}(-0.5, 0.5)$ or another distribution, and $I_n = f(\vec{x}_n)$, $\forall\, n \in \{1, \ldots, N\}$, is the light intensity of agent n.

Lastly, the last group concerns those operators from metaheuristics that model trajectories and dynamics common in the classical mechanics, such as Spiral Optimisation Algorithm (SOA) [41, 42], Central Force Optimisation (CFO) [37], and Gravitational Search Algorithm (GSA) reported by [43, 44].

Definition 17 (Spiral dynamic)**.** The candidate position is determined by the spiral dynamic operator given by,

$$\vec{y}_n = \vec{x}_*(t) - \vec{r} \odot \boldsymbol{R}_D(\theta) \cdot (\vec{x}_n(t) - \vec{x}_*(t)), \tag{19}$$

where $\vec{x}_*(t)$ is the rotation centre, $\theta \in]0°, 360°[$ 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, $\boldsymbol{R}_D(\theta) \in \mathbb{R}^{D \times D}$ is the rotation matrix determined by

$$\boldsymbol{R}_D(\theta) = \prod_{i=1}^{D} \prod_{j=1}^{i-1} \boldsymbol{R}_{i\text{-}j}(\theta). \tag{20}$$

since $\boldsymbol{R}_{i\text{-}j}(\theta) \in \mathbb{R}^{D \times D}$ is the two-dimensional rotation matrix for the plane i-j from all the $D(D-1)/2$ combinations of planes in $\mathbb{R}^D$. Subsequently, each $\boldsymbol{R}_{i\text{-}j}(\theta)$ is defined by its elements $\rho_{p,q} \in \boldsymbol{R}_{i\text{-}j}(\theta)$, with $p, q \in \{1, 2, \ldots, D\}$, as shown,

$$\rho_{p,q} = \begin{cases} 1, & \text{if } p = q \wedge q \neq i, \\ \cos(\theta), & \text{if } p = i \wedge q = i, \\ \sin(\theta), & \text{if } p = j \wedge q = i, \\ -\sin(\theta), & \text{if } p = i \wedge q = j, \\ \cos(\theta), & \text{if } p = j \wedge q = j, \\ 0, & \text{otherwise.} \end{cases}$$

An alternative procedure for determining $\boldsymbol{R}_D(\theta)$ is by utilizing the well-known Euler-Rodrigues's rotation formula [63].

Definition 18 (Central force dynamic)**.** The candidate position calculation through the central force dynamic operator is given by

$$\vec{y}_n = \vec{x}_n(t) + \frac{1}{2}\vec{a}_n(t)\Delta t^2, \tag{21a}$$

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.,* $\Delta t = 1$. The former parameter is obtained such as,

$$\vec{a}_n(t) = \alpha_0 \sum_{\substack{k=1 \\ k \neq n}}^{N} H(M_k(t) - M_n(t))(M_k(t) - M_n(t))^{\alpha_1} \frac{\vec{x}_k(t) - \vec{x}_n(t)}{||\vec{x}_k(t) - \vec{x}_n(t)||_2^{\alpha_2}}, \tag{21b}$$

where $M_n = f(\vec{x}_n(t))$, $\forall\, n \in \{1, \ldots, N\}$, is the mass of agent n, $\alpha_0 \in]0, 0.01]$ is the gravitational constant, $\alpha_1 \in]0, 0.01], \alpha_2 \in]1, 2[$ are two positive factors, and $||\cdot||_2$ is the ℓ^2-norm.

Definition 19 (Gravitational search)**.** The candidate position is achieved by the gravitational search operator as given

$$\vec{y}_n = \vec{x}_n(t) + \vec{r}_0 \odot \vec{v}_n(t) + \vec{a}_n(t), \tag{22a}$$

since $\vec{v}_n(t) \in V(t)$ is the new velocity of agent n. This characteristic is attained such as,

$$\vec{v}_n(t) = \vec{r} \odot \vec{v}_n(t-1) + \vec{a}_n(t), \tag{22b}$$

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

$$\vec{a}_n(t) = G(t) \sum_{\substack{k=1 \\ k \neq n}}^{N} M_k(t) \vec{r}_k \odot \frac{\vec{x}_k(t) - \vec{x}_n(t)}{||\vec{x}_k(t) - \vec{x}_n(t)||_2 + \varepsilon_g}, \tag{22c}$$

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 $\mathcal{U}(0,1)$, $||\cdot||_2$ is the ℓ^2-norm, and $\varepsilon_g \in \mathbb{R}_+$ is a small constant. In particular, $G(t)$ and $M_n(t)$ are determined with

$$G(t) = \alpha_0 e^{-\alpha_1 t} \quad \text{and} \quad M_n(t) = \frac{f(\vec{x}_\circ(t)) - f(\vec{x}_n(t))}{N f(\vec{x}_\circ(t)) - \sum_{k=1}^{N} f(\vec{x}_k(t))}, \tag{22d}$$

where $\alpha_0 \in]0,1]$ is the initial gravitational value, $\alpha_1 \in]0,0.1]$ is the damping ratio, and $f(\vec{x}_\circ(t))$ is the worst fitness value at $\vec{x}_\circ(t)$, such that $\vec{x}_\circ(t) = \arg\max\{\bigcup_{n=1}^{N} f(\vec{x}_n(t))\}$.

4.2. 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 $\vec{y}_n(t) \in \mathfrak{X}$ are the n-th agent position from the current population and its candidate new position, respectively. Also, their corresponding fitness values are $f(\vec{x}_n)$ and $f(\vec{y}_n)$, since $f : \mathfrak{X} \to \mathbb{R}$ 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.

Definition 20 (Direct selection)**.** The new position of the n-th agent is straightforwardly designated equal to $\vec{y}_n$, *i.e.,* $\vec{x}_n(t+1) = \vec{y}_n$.

Definition 21 (Greedy selection)**.** The new position for the n-th agent is assigned equal to $\vec{y}_n$ iff it improves the current position (*i.e.,* $\Delta f_n \leq 0$), thus

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

where the sign "$\leq$" is preferred to escape or overcome large plateaux [30].

Definition 22 (Probabilistic selection)**.** The new position for the n-th agent is set equal to $\vec{y}_n$ if it improves the current position (*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

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

In this work, we employed $p_s = 0.5$.

Definition 23 (Metropolis selection)**.** The new position for the n-th agent is set equal to $\vec{y}_n$ if it improves the current position (*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

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

since $k_B \in \mathbb{R}_+$ is the Boltzmann's constant and $\Theta(t) : \mathbb{Z}_+ \to \mathbb{R}_+$ is the temperature which decreases slowly such as $\Theta(t) = \Theta_0(1-c)^t$, with $\Theta_0 \gg 1$ as the initial temperature, $c \in]0, 1[$ as the decreasing (cooling) ratio, and $t \in \mathbb{Z}_+$ as the current iteration [27]. Some default values used for these parameters are $k_B = 1.0$, $\Theta_0 = 1000$, and $c = 0.01$. It is worth noting that this is only a flavour of this selection scheme, and there are several variations of it in the literature [26].

4.3. Collections of search operators

As mentioned in the methodology, we achieved three databases of operators. They are summarised in Table 3. The first and second collections comprise the perturbators and their associated selectors with predefined values for their tuning parameters. These search operators (SOs) are based on the description given in Sect. 4.1. The third database consists of a list of predefined metaheuristics (MHs) using the previously mentioned SOs. Naturally, these MHs are instances of the 10 MHs selected for extracting their search operators.

Table 3. Collections of operators built from metaheuristics available in the literature

Id.	Name	Description
1	Short Collection	205 search operators
2	Long Collection	1274033 search operators
3	Basic Metaheuristics	66 predefined metaheuristics

Table 4 shows the first (short) collection of SOs utilised in this work. It comprises a total of 205 SOs obtained by considered different variation parameters and predefined values for tuning parameters, according to the information in Table 2. Besides, each row groups four versions of the same operator varying systematically the selector, such as direct, greedy, Metropolis, and probabilistic; except for the first row that corresponds to random search with a defined greedy selector.

The second (long) collection is generated in the same way as the first one but considering 21 different values for each tuning parameter, which gave place to 1,274,033 operators. Unfortunately, we do not show them due to space limitations.

Furthermore, Table 5 is the collection of 66 basic MHs implemented for comparison purposes. These MHs were built by using the search operators from Sect. 4.1. It is easy to notice that rows correspond to MHs such as Random Search (RS), Central Force Optimisation (CFO), Differential Evolution (DE), Firefly Algorithm (FA), Genetic Algorithm (GA), Gravitational Search Algorithm (GSA), Cuckoo Search (CSA), Simulated Annealing (SA), Stochastic Spiral Optimisation Algorithm (SSOA), and Particle Swarm Optimisation (PSO). Rows in Table 5 represent predefined metaheuristics, mostly with cardinalities of one, except for GA, DE, and CS, whose cardinality equals two. A range of identifiers (*e.g.,* 1–9) means that the same MH is used but with different configurations of tuning parameters, which are obtained from the possible value combinations. For example, for the MH based on central force dynamic (*i.e.,* CFO), $\alpha_0 = \{0.001, 0.010, 0.100\}$ and $\alpha_1 = \{0.01, 0.05, 0.10\}$ and $\alpha_2 = 1.5$. So, when id = 1, $\alpha_0 = 0.001$ and $\alpha_1 = 0.01$ are chosen. If id = 2, $\alpha_0 = 0.010$ and $\alpha_1 = 0.01$ are used instead. Consequently, $\alpha_0 = 0.100$ and $\alpha_1 = 0.10$ are used when id = 9.

4.4. Preliminary experiments

Figure 5 shows the normalised average ranking for each search operator, implemented as a metaheuristic, solving the benchmark functions (Table 1) from each category and varying their dimensions (*i.e.,* 2, 5, 10, 20, 30, 40, and 50). All the MHs employed the same initialiser and finaliser as well as the conditions specified in Sect. 3. Values in Figure 5 were determined by the following procedure: First, all the performance values, determined with (4), were ranked for each search operator solving each problem with a given dimension. Then, ranks were grouped according to the categories of each problem by averaging them. Lastly, the resulting information was normalised by dividing on the total sum of results for the same category. So, lower values (purplish colours) indicate better performing operators. It is essential to clarify that search operators with high values (reddish colours) are not necessarily bad for solving problems of a given category; it just means that there are more suitable methods to do so.

First of all, a very particular operator (commonly associated with the No-Free-Lunch theorem) is the random search, which was located as the first one in the collection (cf. Table 2). From results, we can notice

Table 4. Collection of 205 search operators employed in this work. Each row groups four versions of the same operator varying systematically the selector, such as direct, greedy, Metropolis, and probabilistic, except for the first row that corresponds to random search with a defined greedy selector.

Ids.	Operator name	Configuration
0	Random search	$\alpha = 1.0$
1–4	Central force dynamic	$\alpha_0 = 0.001$, $\alpha_1 = 0.01$, $\alpha_2 = 1.5$
5–8	Differential mutation *rand* scheme	$M = 1$, $\alpha_0 = \alpha_1 = 1.0$
9–12	Differential mutation *best* scheme	
13–16	Differential mutation *current* scheme	
17–20	Differential mutation *current-to-best* scheme	
21–24	Differential Mutation *rand-to-best* scheme	
25–28	Differential mutation *rand-to-best-and-current* scheme	
29–32	Firefly dynamic with Uniform distribution	$\alpha_0 = 1.0$, $\alpha_1 = 1.0$, $\alpha_2 = 100$
33–36	Firefly dynamic with Normal distribution	
37–40	Firefly dynamic with Lévy distribution	
41–44	Genetic single-point crossover with rank pairing	$m_p = 0.4$
45–48	Genetic single-point crossover with roulette wheel pairing	
49–52	Genetic single-point crossover with random pairing	
53–56	Genetic single-point crossover with tournament pairing	$m_p = 0.4$, $M_T = 2$, $p_T = 1.0$
57–60	Genetic two-points crossover with rank pairing	$m_p = 0.4$
61–64	Genetic two-points crossover with roulette wheel pairing	
65–68	Genetic two-points crossover with random pairing	
69–72	Genetic two-points crossover with tournament pairing	$m_p = 0.4$, $M_T = 2$, $p_T = 1.0$
73–76	Genetic uniform crossover with rank pairing	$m_p = 0.4$
77–80	Genetic uniform crossover with roulette wheel pairing	
81–84	Genetic uniform crossover with random pairing	
85–88	Genetic uniform crossover with tournament pairing	$m_p = 0.4$, $M_T = 2$, $p_T = 1.0$
89–92	Genetic blend crossover with rank pairing	$m_p = 0.4$
93–96	Genetic blend crossover with roulette wheel pairing	
97–100	Genetic blend crossover with random pairing	
101–104	Genetic blend crossover with tournament pairing	$m_p = 0.4$, $M_T = 2$, $p_T = 1.0$
105–108	Genetic linear crossover with rank pairing	$p_m = 0.4$, $\beta_1 = 0.5$, $\beta_2 = 0.5$
109–112	Genetic linear crossover with roulette wheel pairing	
113–116	Genetic linear crossover with random pairing	
117–120	Genetic linear crossover with tournament pairing	$m_p = 0.4$, $M_T = 2$, $p_T = 1.0$ $\beta_1 = \beta_2 = 0.5$
121–124	Genetic mutation with Uniform distribution	$\alpha = 1.0$, $p_e = 0.1$, $p_m = 0.25$
125–128	Genetic mutation with Normal distribution	
129–132	Genetic mutation with Lévy distribution	
133–136	Gravitational search	$\alpha_0 = 1.0$, $\alpha_1 = 0.02$
137–140	Random flight with Lévy distribution	$\alpha = 1.0$, $\beta = 1.5$
141–144	Random flight with Uniform distribution	
145–148	Random flight with Normal distribution	
149–152	Local random walk with Uniform distribution	$\alpha = 1.0$, $p = 0.75$
153–156	Local random walk with Normal distribution	
157–160	Local random walk with Lévy distribution	
161–164	Random sample	
165–168	Random search with Uniform distribution	$\alpha = 0.01$
169–172	Random search with Normal distribution	
173–176	Random search with Lévy distribution	
177–180	Spiral dynamic	$r_0 = 0.9$, $\theta = 22.5°$, $\sigma_r = 0.1$
181–184	Inertial swarm dynamic with Uniform distribution	$\alpha_0 = 0.7$, $\phi_1 = 2.54$, $\phi_2 = 2.56$
185–188	Inertial swarm dynamic with Normal distribution	
189–192	Inertial swarm dynamic with Lévy distribution	
193–196	Constricted swarm dynamic with Uniform distribution	$\kappa = 1.0$, $\phi_1 = 2.54$, $\phi_2 = 2.56$
197–200	Constricted swarm dynamic with Normal distribution	
201–204	Constricted swarm dynamic with Lévy distribution	

that its ranking was mediocre in the lowest dimensions, and it worsens as dimensionality increases. Thence, we can infer that there is always a metaheuristic based on at least a search operator that can surmount this operator (Figure 5).

Furthermore, we can notice at a first glance that the constricted swarm dynamic with uniform distribution family (*i.e.,* 193–196) prevails as good ranked operators for almost all the categories and dimensions. Similarly, metaheuristics based on the spiral dynamic with direct and probabilistic selectors (*i.e.,* 177 and 180, respectively) present remarkable performances for all cases. In two dimensions (Figure 5), we find many search operators capable of solving problems almost independently of their category. A first group of them are the differential mutations using *rand*, *current*, *current-to-best*, and *rand-to-best* schema (*i.e.,* 6, 14, 18, and

Table 5. Collection of 66 basic metaheuristics (MHs), composed by one or two search operators and the corresponding selectors, employed in this work. Each row represents a metaheuristic implemented for a given configuration. A range of identifiers means that the same MH is used but with different configurations of tuning parameters.

Ids.	Search operators	Configurations	Selectors
0	Random search	$\alpha = 1.0$	Greedy
1–9	Central force dynamic	$\alpha_0 = \{0.001, 0.01, 0.1\}$, $\alpha_1 = \{0.01, 0.05, 0.1\}$, $\alpha_2 = 1.5$	Direct
10–13	Differential mutation *rand-to-best-and-current* scheme	$M = \{1, 2\}$, $\alpha_0 = \alpha_1 = \{1.0, 2.0\}$	Greedy
	Differential crossover *bin* mechanism	CR $= 0.2$	Greedy
14–17	Differential mutation *current-to-best* scheme	$M = \{1, 2\}$, $\alpha_0 = \alpha_1 = \{1.0, 2.0\}$	Greedy
	Differential crossover *bin* mechanism	CR $= 0.2$	Greedy
18–23	Firefly dynamic with Uniform distribution	$\alpha_0 = \{0.1, 0.5, 1.0\}$, $\alpha_1 = 1.0$, $\alpha_2 = \{100, 500\}$	Direct
24	Genetic single-point crossover with tournament pairing	$m_p = 0.4$, $M_T = 2$, $p_T = 1.0$	Direct
	Genetic mutation with Uniform distribution	$\alpha = 1.0$, $p_e = 0.1$, $p_m = 0.25$	Direct
25	Genetic single-point crossover with roulette wheel pairing	$m_p = 0.4$	Direct
	Genetic mutation with Uniform distribution	$\alpha = 1.0$, $p_e = 0.1$, $p_m = 0.25$	Direct
26	Genetic single-point crossover with random pairing	$m_p = 0.4$	Direct
	Genetic mutation with Uniform distribution	$\alpha = 1.0$, $p_e = 0.1$, $p_m = 0.25$	Direct
27	Genetic single-point crossover with tournament pairing	$m_p = 0.4$, $M_T = 2$, $p_T = 1.0$	Direct
	Genetic mutation with Uniform distribution	$\alpha = 1.0$, $p_e = 0.1$, $p_m = 0.25$	Direct
28	Genetic single-point crossover with roulette wheel pairing	$m_p = 0.4$	Direct
	Genetic mutation with Uniform distribution	$\alpha = 1.0$, $p_e = 0.1$, $p_m = 0.25$	Direct
29	Genetic single-point crossover with random pairing	$m_p = 0.4$	Direct
	Genetic mutation with Uniform distribution	$\alpha = 1.0$, $p_e = 0.1$, $p_m = 0.25$	Direct
30–35	Gravitational search	$\alpha_0 = \{1.0, 0.5\}$, $\alpha_1 = \{0.020.04, 0.05\}$	Direct
36–41	Random flight with Lévy distribution	$\alpha = \{1.0, 0.1\}$	Greedy
	Local random walk with Uniform distribution	$\alpha = 1.0$, $p = \{0.75, 0.50, 0.25\}$	Greedy
42	Local random walk with Uniform distribution	$\alpha = 1.0$, $p = 1.0$	Metropolis
43	Local random walk with Normal distribution	$\alpha = 1.0$, $p = 1.0$	Metropolis
44	Local random walk with Lévy distribution	$\alpha = 1.0$, $p = 1.0$	Metropolis
45	Local random walk with Uniform distribution	$\alpha = 0.1$, $p = 1.0$	Metropolis
46	Local random walk with Normal distribution	$\alpha = 0.1$, $p = 1.0$	Metropolis
47	Local random walk with Lévy distribution	$\alpha = 0.1$, $p = 1.0$	Metropolis
48–53	Spiral dynamic	$r_0 = 0.9$, $\theta = \{22.5°, 45°\}$, $\sigma_r = \{0.00, 0.05, 0.10\}$	Direct
54–60	Inertial swarm dynamic with Uniform distribution	$\alpha_0 = \{0.7, 0.9\}$, $\phi_1 = \{2.54, 1.54\}$, $\phi_2 = \{2.56, 1.56\}$	Direct
61–65	Constricted swarm dynamic with Uniform distribution	$\kappa = \{1.0, 1.5\}$, $\phi_1 = \{2.54, 1.54\}$, $\phi_2 = \{2.56, 1.56\}$	Direct

22), both implementing the greedy selector. This is followed by all the constricted swarm dynamic operators (193–204), and the random flight with Lévy (137–140) and Normal (145–148) distributions. By visual inspection, it is easy to notice that the latter rendered excellent performances for up to 20 dimensions. Conversely, almost all the genetic crossover families seem to be the worst performing ones in 2D. It is interesting to see that some operators that reached poor positions for low dimensional problems, obtained great ranks for high dimensional ones. For example, the genetic crossover operators and, particularly, the genetic uniform, blend, and linear crossover with random and tournament pairing for differentiable and separable problems (those encoded as 111 and 110). The counter-case is observed for the first 41 SOs: random search, central force dynamic, differential mutation, and firefly dynamic.

It is evident that each SO can be a great option for solving some problems but a not-quite-good option for others. The aforementioned remarks about results from the brute force experiments are just an overview about how search operators perform in problem with different features and dimensionality. Our intention in this stage consists of identifying the capabilities of SOs for certain types of continuous optimisation problems. Therefore, aiming to achieve a general insight, we use the normalised average ranking values determined in Figure 5 as probability weighting function for the search operators in the short collection. Figure 6 illustrates such weights for each category and dimension of the benchmark problems. With this representation as stacked bars, it is easy to recognise which operators dominate the others. This dataset of weights is employed in the next stage as a naive guiding strategy for choosing the SOs from the heuristic space. Besides, we were able to notice several interesting behaviours of the operators while dealing with diverse kinds of problems.

Likewise, we carried out the same experiments but using a different database of operators. In this case, those search operators have predefined configuration corresponding to basic (well-known) metaheuristics from

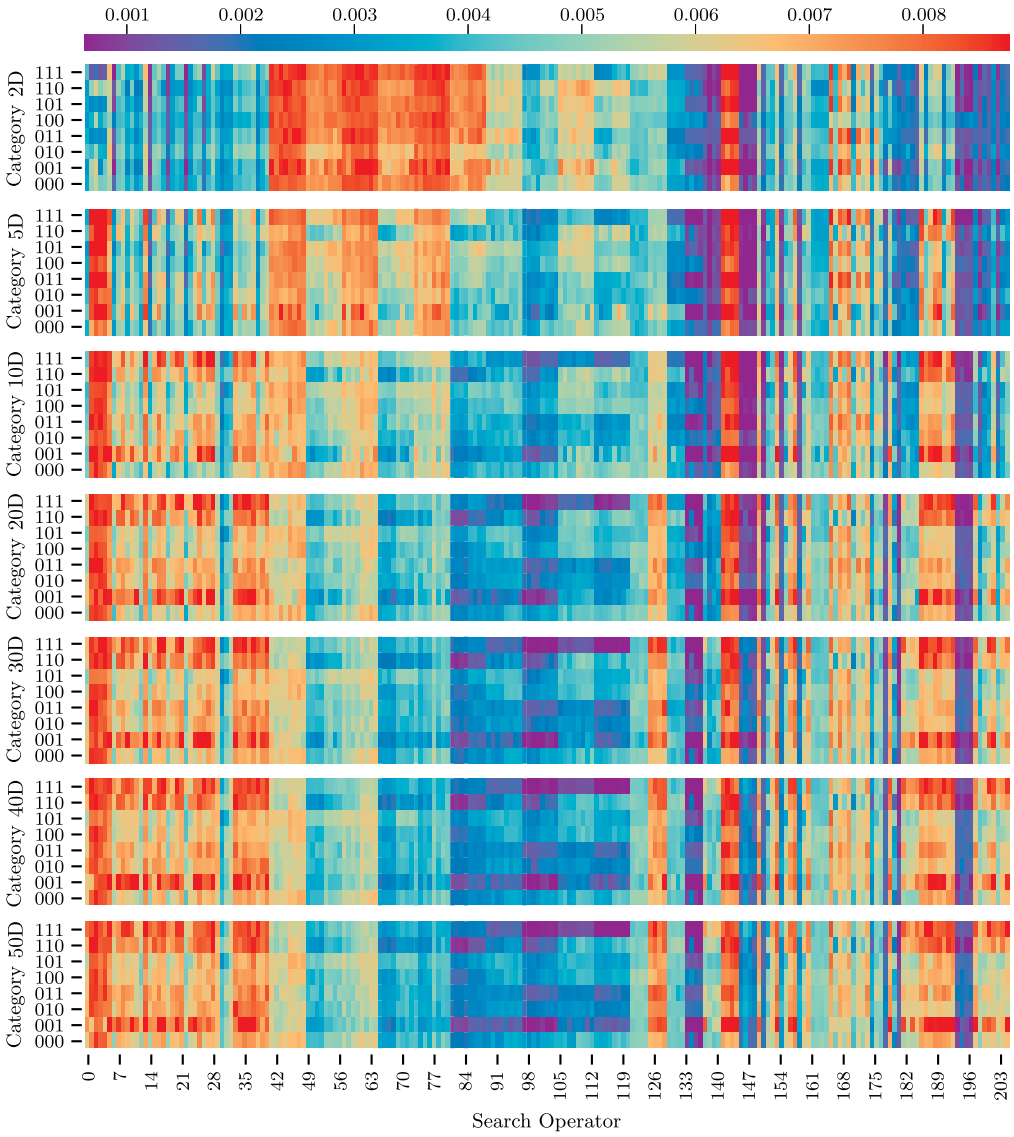


Figure 5. Normalised average ranking (the lower, the better) for each search operator solving the benchmark functions from each category and varying the dimensions. Categories are binary encoded by a DSU triplet, where each letter stands for the Differentiable, Separable, and Unimodal feature, respectively.

the literature. Figure 7 presents the normalised average ranking values found for each basic metaheuristic (cf. Table 5) solving the benchmark functions in Table 1.

In general, the most relevant basic MH is the constricted version of particle swarm optimisation (PSO) with almost all the tuning parameters employed (*i.e.,* 61–63). Conversely, the worst ranked MHs were differential evolution (DE), and the Stochastic Spiral Optimisation Algorithm (SSOA), see Table 5. In particular, Cuckoo Search (CS) implementations (*i.e.,* 36–41) were greatly ranked in problems with 2, 5 and 10 dimensions, but their performance decreased for higher dimensional problems. Nevertheless, these implementations never reached the bottom of the rank in 50D. Lastly, some isolated cases of Central Force Optimisation (CFO) are detected as good performing ones for certain types of problems with over two dimensions, *e.g.,* for non-differentiable, non-separable, and unimodal problems (category 001).

It is essential to mention that a metaheuristic achieving a poor position in the ranking (Figure 7), does not imply that such a MH is bad. Indeed, it just means that for such a category of problems and dimensionality, there is at least other MH which performs better. Take in mind that results showed in Figure 7 are just a macroscopic point of view of what really happens with performance. Certainly, if we analyse each meta-

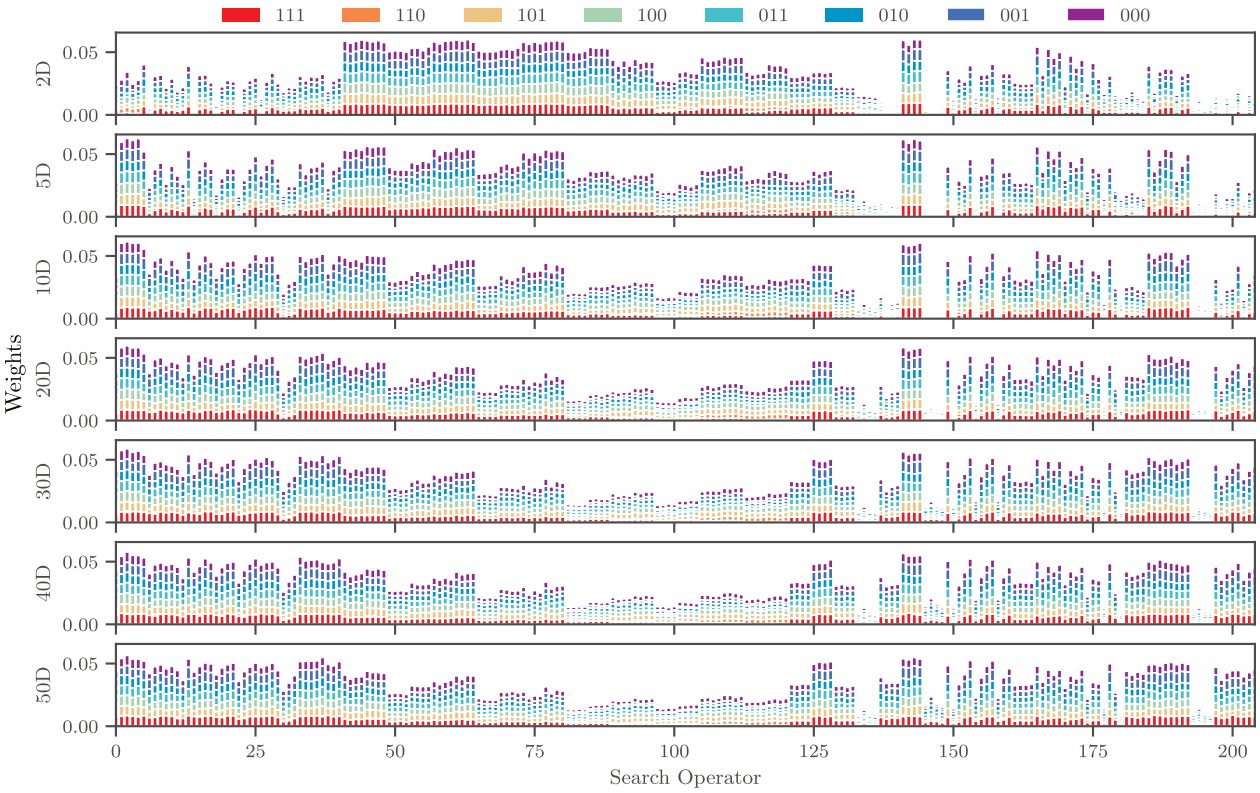


Figure 6. Weights obtained for each search operator solving the benchmark functions and varying their dimensions (*i.e.,* 2, 3, 10, 20, 30, 40, and 50). These values were obtained by grouping the results according to the problem features and dimensionality, then ranking and normalising the data. Colours indicate the problem category binary encoded by a DSU triplet, where each letter stands for the Differentiable, Separable, and Unimodal feature, respectively

heuristic and its performance for each one of the problems, we would be tailoring MHs for very specific cases. This may be related to algorithm portfolios, which is not the objective of this work. So, as with Heisenberg's uncertainty principle, we sacrifice particularity for generality at this point of the work.

4.5. Main experiments

By employing the information obtained in the previous sections, we implement the hyper-heuristic (HH) procedure powered by Simulated Annealing (SA) to customise MHs for solving the benchmark problems (Table 1). In this case, we used the weights from the brute force experiments to help the HH exploration of the heuristic space (the short collection of search operators). This naïve *a priori* knowledge is employed to bias the random selection of search operators when generating a candidate neighbour solution (cf. Pseudocode 1). For generating those MHs, two deployments of this experiment were carried out by fixing the maximum cardinality ($\varpi_{\max}$) to three and five. Recall that this parameter prevents the HH to build metaheuristics with more than $\varpi_{\max}$ operators (perturbator-selector couples).

Figures 8 and 9 show the fitness evolution of the resulting metaheuristics, when considering three and five as maximum cardinality, for eight problem functions selected for illustrative purposes and with different dimensionalities. Each one of these problems belongs to a category from those listed in Table 1. Particularly, the first one (the Sphere function) is categorised as 111, which means differentiable, separable, and unimodal; oppositely, the last one (the Needle-Eye function) is a case of the category 000, *i.e.,* non-differentiable, non-separable, and multimodal. Plus, the other functions possess different combinations of these features. These functions are organised according to their level of hardness, such that the former and latter are considered to have a low and high level, whilst the others have a mid-level. Furthermore, consider that each group of lines sharing the same colour within a plot represents the 30 repetitions (transparent lines) ran by a customised MH, and the solid line for each group indicates the mean behaviour. Each group is substantially different from the other groups and plots. For example, the metaheuristic found for the Griewank 2D problem is not

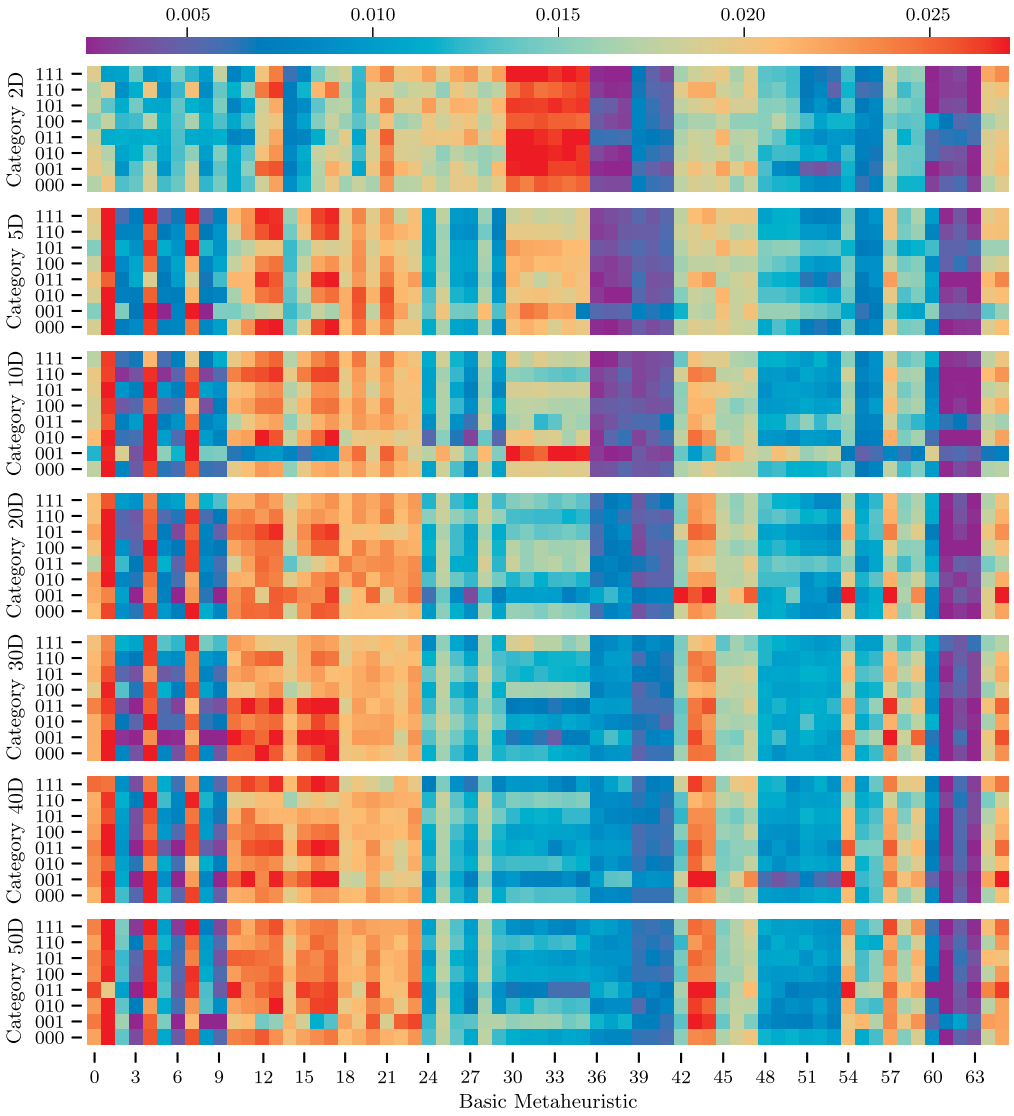
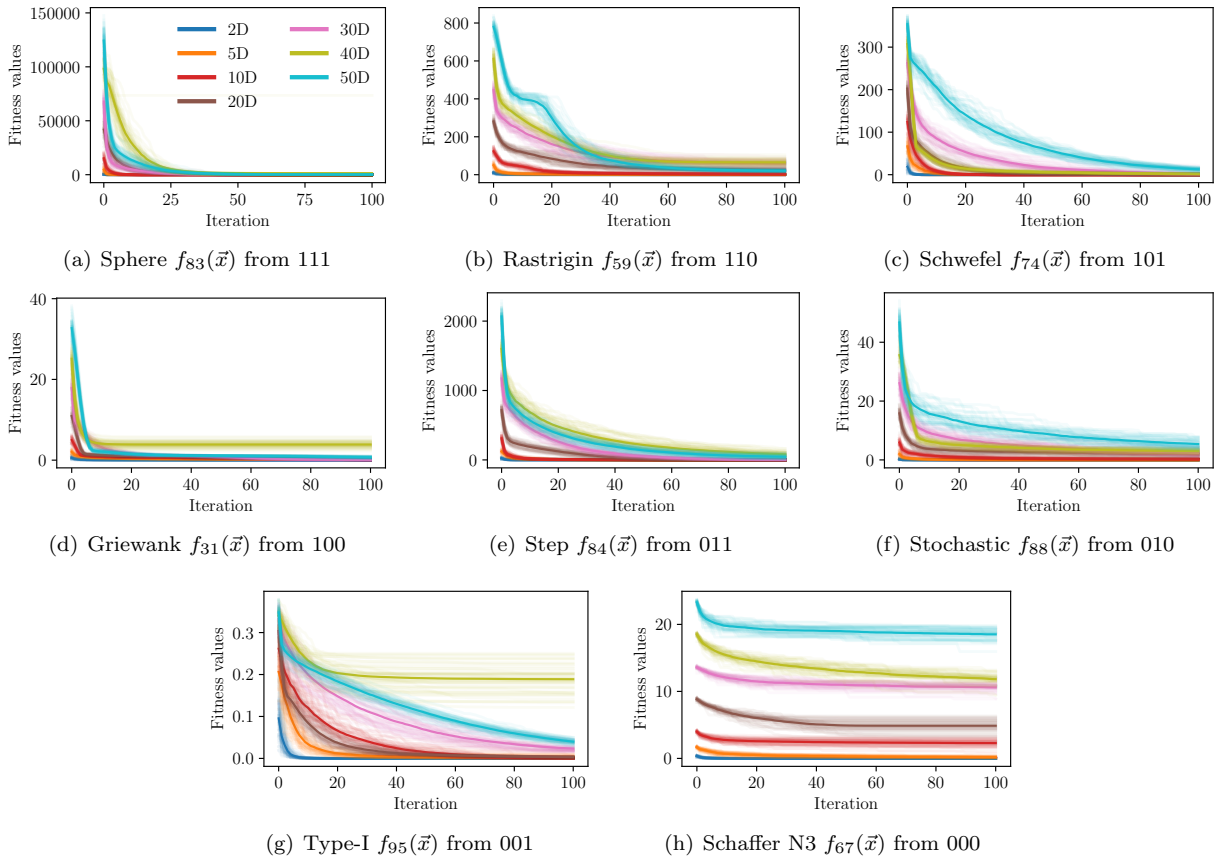


Figure 7. Normalised average ranking (the lower, the better) for each basic metaheuristic solving the benchmark functions from each category and varying the dimensions. Categories are binary encoded by a DSU triplet, where each letter stands for the Differentiable, Separable, and Unimodal feature, respectively.

the same for either that problem with different dimensionality or another two-dimensional problem. Keeping this in mind, we can notice that the customised MH exhibits a desirable convergence for the first five cases (*i.e.,* Figures 8(a)-(e) and 9(a)-(e)), even when the number of dimensions increases. It is natural to recognise that the well-known dimensionality curse affects the solution of *easy* problems such as the Sphere. For the sixth and seventh cases, when solving the Stochastic and Type-I problems, the number of iterations seems to be insufficient to reach the optimal fitness for higher dimensions, such as 40D and 50D. For the hardest category, we observed that dimensionality affects the convergence of the optimisation procedure dramatically. Analysing the curves comparatively (Figures 8 and 9), it is possible to detect that relaxing the maximum cardinality seems to be a good practice to reach more suitable metaheuristics.

Subsequently, we proceed to study the overall performance of the two deployments mentioned above. To do so, Figure 10 shows the average success rate for the MHs generated by our hyper-heuristic model. This rate is calculated as the ratio of basic MHs (cf. Figure 7) that the tailored MH outperforms for a given problem with fixed dimensionality. Hence, for each category and number of dimensions, success rates are summarised by using the average (solid lines) and standard deviation (shaded regions) values, as shown in Figure 10. In such a tier, a unitary success rate value means that the customised metaheuristic for a given



(a) Sphere $f_{83}(\vec{x})$ from 111 (b) Rastrigin $f_{59}(\vec{x})$ from 110 (c) Schwefel $f_{74}(\vec{x})$ from 101

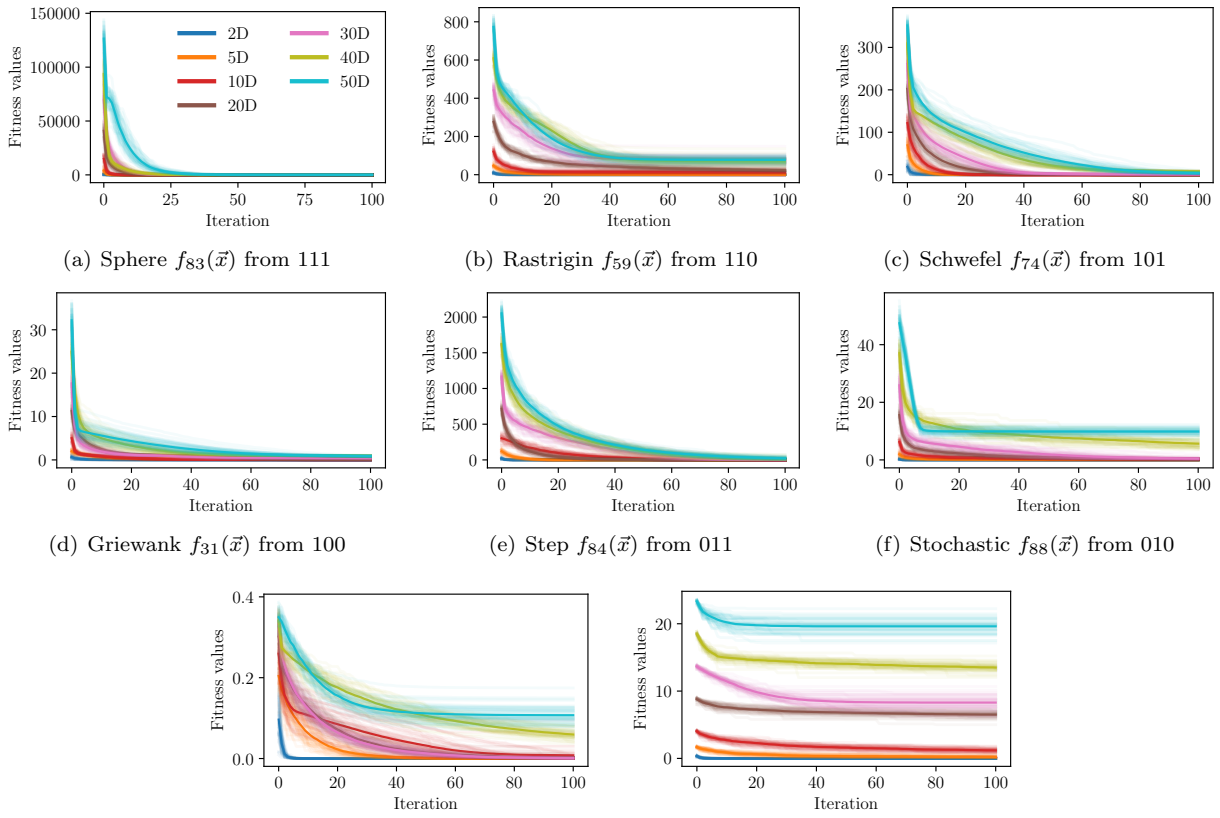
(d) Griewank $f_{31}(\vec{x})$ from 100 (e) Step $f_{84}(\vec{x})$ from 011 (f) Stochastic $f_{88}(\vec{x})$ from 010

(g) Type-I $f_{95}(\vec{x})$ from 001 (h) Schaffer N3 $f_{67}(\vec{x})$ from 000

Figure 8. Fitness evolution from the MHs achieved via the HH approach, using the first heuristic collection, when solving selected functions from each category. A maximum cardinality was set equal to three.

problem achieved better results (in terms of the performance function) than all the (66) basic MHs. First of all, we recognise that in both implementations, the customised metaheuristics rendered an average success rate of at least 50%, except for some particular cases of the categories 010, 101, and 001. The former shows low success rates in five and ten dimensions, meaning a poor performance of the tailored MHs, compared against the basic ones. The second, for category 101, it represents the problems where the HH approach is mostly affected by the dimensionality (Figure 10(a)). However, when the maximum number of search operator equals five (Figure 10(b)), resulting MHs improve the success rate. Besides, category 001 corresponds to non-differentiable, non-separable, and unimodal functions, which is an *exotique* configuration of features. This can be corroborated with the number of functions (*i.e.,* two) that compose such a category in Table 1. Withal, it is essential to comment that even if the success rate is lower than 50% but different from zero, the MH generated by our approach beats at least one of the basic metaheuristics. In other words, by using the proposed framework, we provide an automatic way of finding a suitable methodology for solving a problem. Such a task requires moderate expertise of, say, a practitioner who must know which basic MH to choose for solving a given problem. Also, selecting a metaheuristic that excels in a specific problem does not guarantee that such a performance is preserved for other scenarios; namely, the *no-free-lunch* theorem [3].

To obtain a overview of how the maximum cardinality $\varpi_{\max}$ influences the building of MHs, we exhibit Figure 11. It shows the frequency of cardinalities ϖ from the achieved metaheuristics per number of dimensions when $\varpi_{\max}$ is equal to 3 and 5. In the first deployment of the HH strategy (Figure 11(a)), it is worth noting that the most frequent value is $\varpi = \varpi_{\max} = 3$, and its popularity increases with the dimensionality, as expected. Cardinalities of one and two describe an almost inverse tendency, which may be due to the increasing requirements for solving harder problems. Likewise, in the second implementation (Figure 11(b)), we observe a rather similar behaviour except for the first two dimensionalities, *i.e.,* 2D and 5D. On these

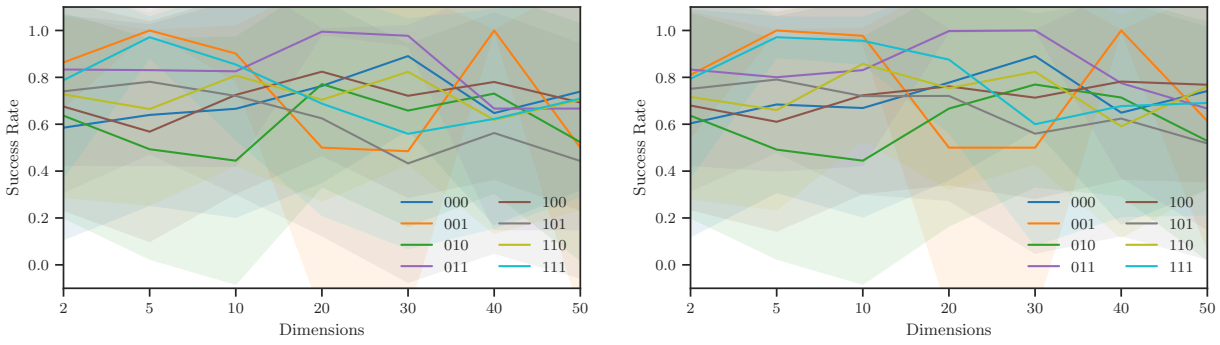


(a) Sphere $f_{83}(\vec{x})$ from 111 (b) Rastrigin $f_{59}(\vec{x})$ from 110 (c) Schwefel $f_{74}(\vec{x})$ from 101

(d) Griewank $f_{31}(\vec{x})$ from 100 (e) Step $f_{84}(\vec{x})$ from 011 (f) Stochastic $f_{88}(\vec{x})$ from 010

(g) Type-I $f_{95}(\vec{x})$ from 001 (h) Schaffer N3 $f_{67}(\vec{x})$ from 000

Figure 9. Fitness evolution from the MHs achieved via the HH approach, using the first heuristic collection, when solving selected functions from each category. A maximum cardinality was set equal to five.

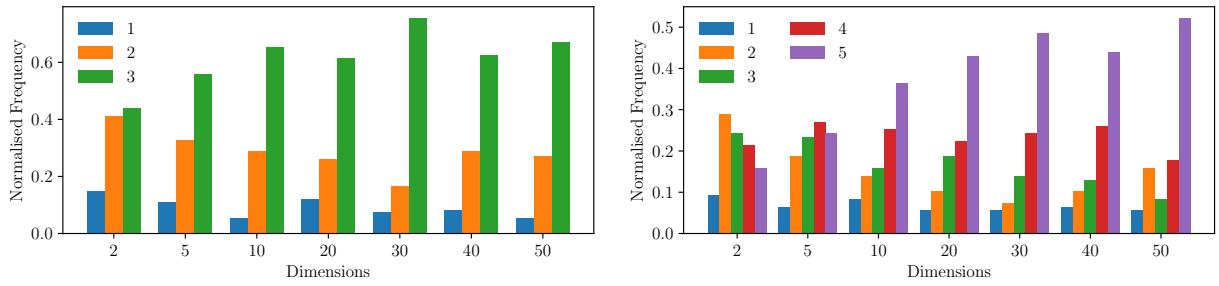


(a) $\varpi_{\max} = 3$ (b) $\varpi_{\max} = 5$

Figure 10. Average (solid lines) and standard deviation (shaded regions) of the success rate of the customised metaheuristics solving problems from each category. MHs were built via the hyper-heuristic approach and utilising the short collection of search operators. Categories are binary encoded by a DSU triplet, where each letter stands for the Differentiable, Separable, and Unimodal feature, respectively. Two values for the maximum cardinality ($\varpi_{\max}$) such as (a) three and (b) five were used.

dimensions, MHs with cardinalities between two and four are oft-generated to solve the optimisation problems. Thus, in general, by giving more flexibility to the HH in the number of search operators, increase the suitability and performance of the tailored MH. However, it has a side effect that can be a serious drawback, *i.e.,* an excessive increase of the number of possible combinations. For $\varpi_{\max} = 3$, the HH strategy must

explore a total of 204263 possible combinations, whereas for $\varpi_{\max} = 5$ it is 111673439. This number is not quite high for a combinatorial problem, but take in mind that each possibility requires 100 iterations and 30 repetitions. Moreover, a MH with more than three SOs, in principle, requires more iterations or generations to reach the optimal solution. Thus, 100 steps could be insufficient to achieve a desired performance, as remarked in Figures 8 and 9.



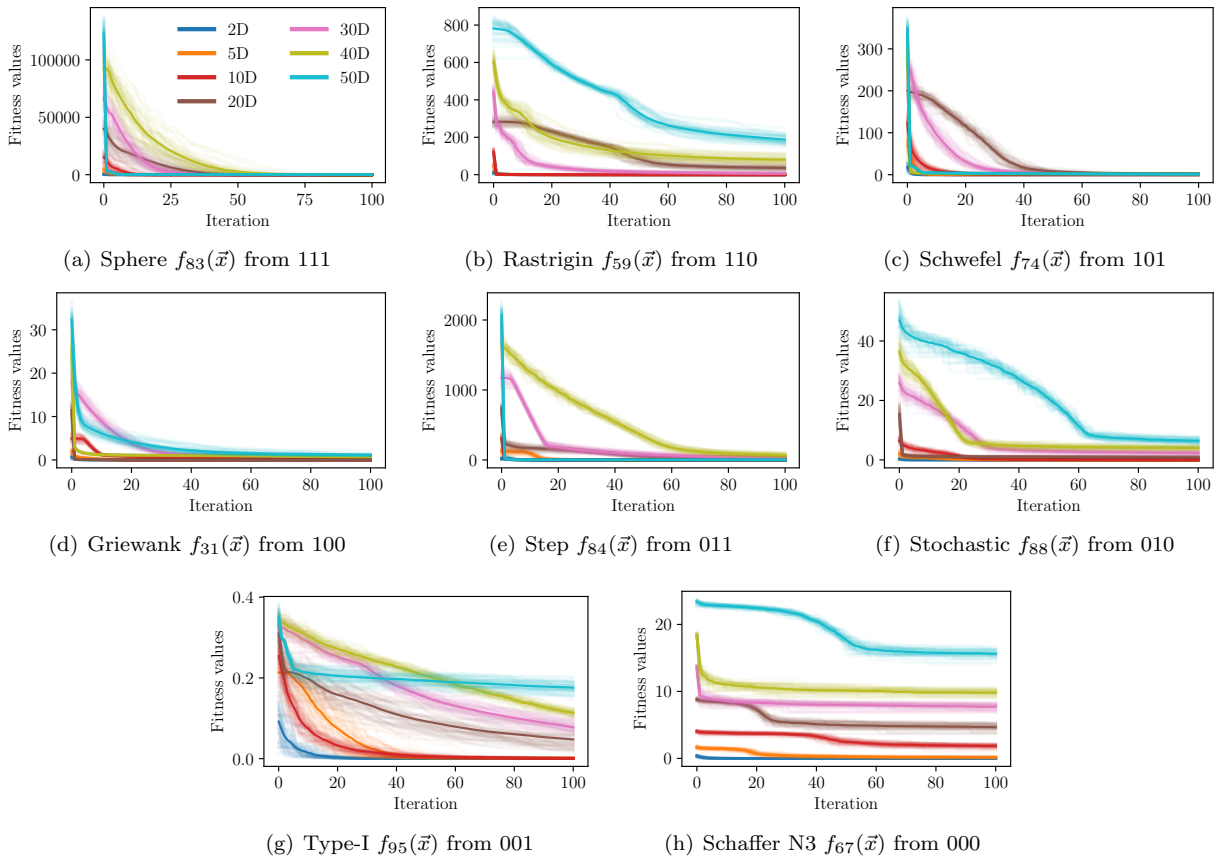
(a) $\varpi_{\max} = 3$ (b) $\varpi_{\max} = 5$

Figure 11. Frequency of the resulting cardinalities from customised MHs, obtained via the HH approach withe the short heuristic collection, for solving benchmark problems varying their dimensionalities. Two values for the maximum cardinality ($\varpi_{\max}$) such as (a) three and (b) five were used.

At this point, we have explored two implementations of the hyper-heuristic framework for tailoring metaheuristics, where both used the same heuristic collection but with a different value for the maximum cardinality ($\varpi_{\max}$). The last deployment studied in this work consists on employing the large collection of search operators (Table 3) and by setting $\varpi_{\max} = 3$ to avoid an excessive increase of the metaheuristic solution space. In this implementation, there is not *a priori* weighting probability function to help the heuristic search, mainly due to the technical unfeasibility of obtaining such information (*i.e.,* at least 1274033×107×30×100×30 function evaluations are required). Similarly to the prior implementations, Figure 12 shows the fitness evolution from the customised MH when solving the selected problems. By comparing these curves to those from Figures 8 and 9, we recognise akin tendencies. Particularly, in the last two functions (*i.e.,* Type-I and Shaffer), the fitness evolution trajectories are more accentuated towards the optimal value. Some drawbacks are also noticeable. For example, when solving the Rastrigin function, the MHs generated by the third approach require much more iterations than those achieved by the other HH implementations.

To complement the comparison, Figure 13 displays the success rates determined for the obtained metaheuristics and the frequency of their cardinalities. From Figure 13(a), one can notice that the average success rates are by far better than the previous implementations. The only exception is observed with the category 001 that, as mentioned before, presents high variability due to the number of functions that conform it and to their implicit hardness. Besides, from Figure 13(b), we identify a more natural transition from two to three as the most frequent cardinalities when the dimensionality of problems increases.

As an illustrative example, Figure 14 displays the diversity of search operators chosen for the heuristic sequence $\boldsymbol{h}$ that give place to the customised metaheuristics. In this plot, we grouped the heuristic collection into the 12 families of operators (Table 2) to facilitate the analysis. There, each row corresponds to each deployment of the proposed framework, *i.e.,* Figure 14(a) to experiment 1 with the short collection and $\varpi_{\max} = 3$, (b) to experiment 2 with the short collection and $\varpi_{\max} = 5$, and (c) to experiment 3 with the long collection and $\varpi_{\max} = 3$. Consider that coloured lines representing categories are transparent. So, more intense colours relate to paths (combinations) with higher frequency. We identify that the most popular combinations contain operators from the families of genetic crossover, differential mutation, and swarm dynamic, for experiments 1 and 2 (Figure 14(a)–(b)); and central force dynamic and swarm dynamic for experiment 3 (Figure 14(c)). Furthermore, bear in mind that horizontal lines indicate metaheuristics with sequences comprising two or more consecutive perturbations from the same family. This is a valid result because these operators perform differently (up to certain degree) depending on how distinct their configurations are (cf. Table 2). For example, two operators from the same family may have configurations such that one of them is specialised on exploration, whilst the other on exploitation. Hence, such behaviour is found chiefly for the genetic crossover and swarm dynamic families in experiments 1 and 2, and central force dynamic and swarm dynamic families in experiment 3. The difference between the first two experiments and

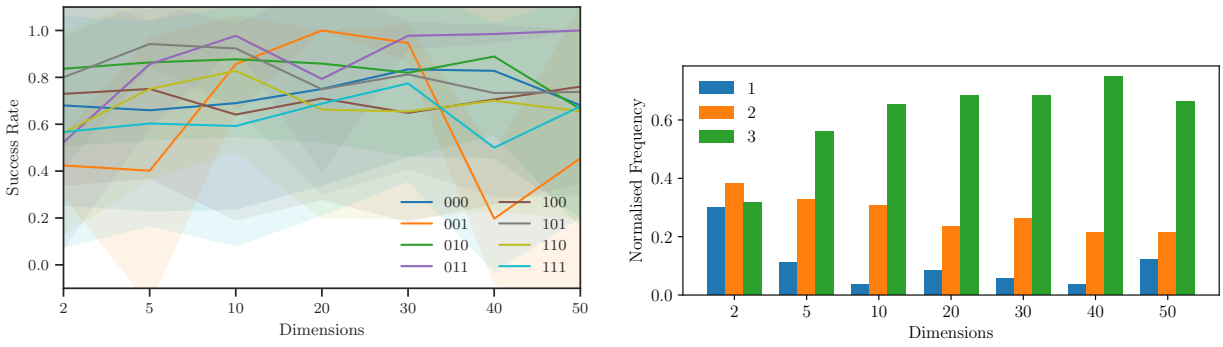


(a) Sphere $f_{83}(\vec{x})$ from 111 (b) Rastrigin $f_{59}(\vec{x})$ from 110 (c) Schwefel $f_{74}(\vec{x})$ from 101

(d) Griewank $f_{31}(\vec{x})$ from 100 (e) Step $f_{84}(\vec{x})$ from 011 (f) Stochastic $f_{88}(\vec{x})$ from 010

(g) Type-I $f_{95}(\vec{x})$ from 001 (h) Schaffer N3 $f_{67}(\vec{x})$ from 000

Figure 12. Fitness evolution from the MH achieved via the HH approach, using the second heuristic collection, when solving eight representative functions from each category. A maximum cardinality was set equal to three.



(a) (b)

Figure 13. Characteristics obtained from the MHs built via the HH approach and using the long heuristic collection: (a) Average (solid lines) and standard deviation (shaded regions), and (b) frequency of cardinalities from the customised MHs for solving the benchmark problems with different dimensionalities. Categories are binary encoded by a DSU triplet, where each letter stands for the Differentiable, Separable, and Unimodal feature.

the last one is attributed to the wide, by far, diversity of search operators (several configurations of them) that the HH model utilised in the latter. So, we sacrificed the *a priori* knowledge acquired from the brute force experiment for a more diverse collection of heuristics.

Moreover, increasing the maximum cardinality also promotes the diversity of operators used in the meta-heuristics, see Figure 14(b). It corroborates the finding from Figure 11(b), where MHs with cardinalities up

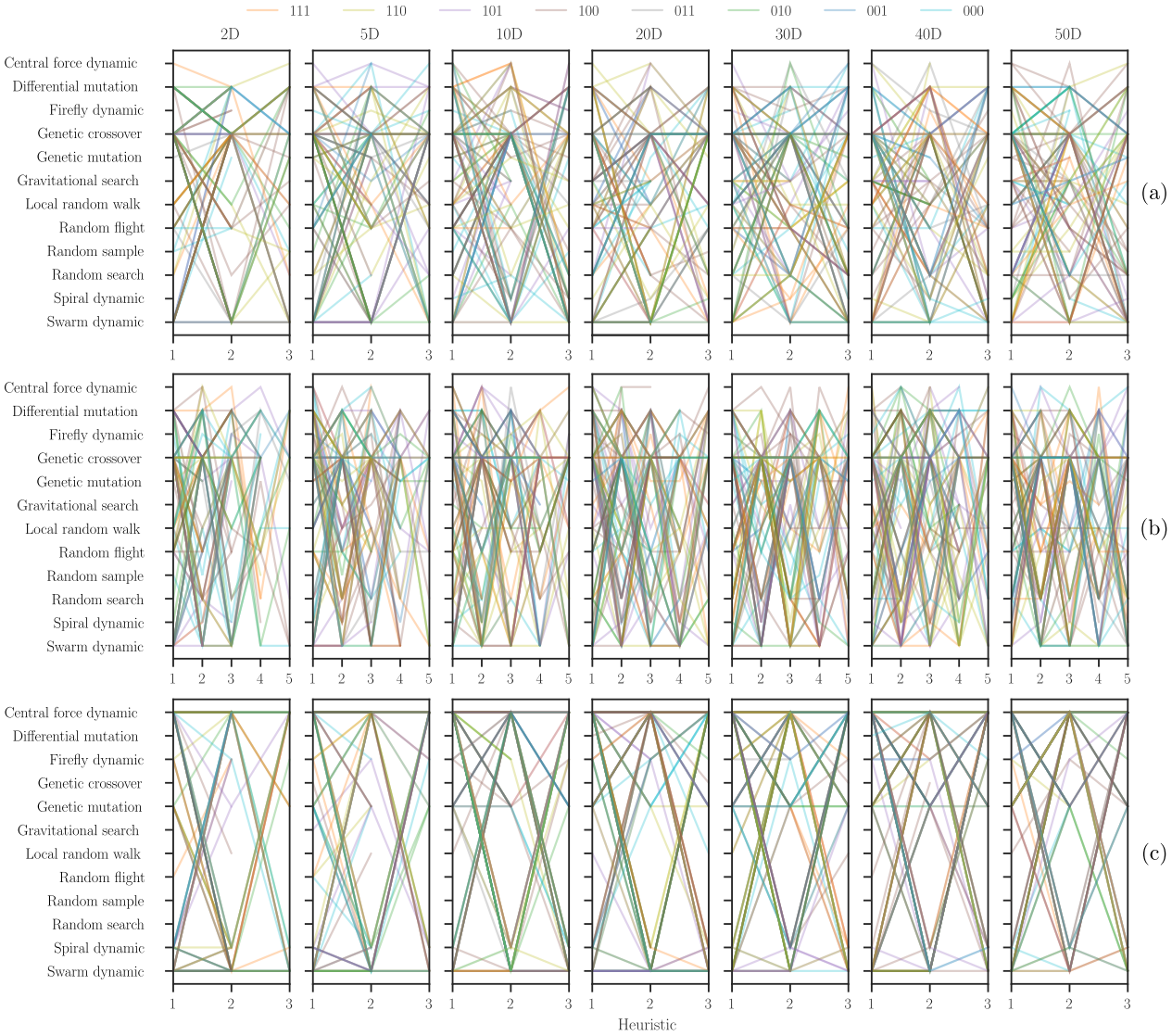


Figure 14. Heuristic sequences obtained for the tailored metaheuristics built, for solving problems of a given category and dimensionality, via the hyper-heuristic approach employing three different setups: (a) short collection and $\varpi_{\max} = 3$, (b) the short collection and $\varpi_{\max} = 5$, and (c) the long collection and $\varpi_{\max} = 3$. Search operators from the collections are grouped into 12 families and categories (colours) are binary encoded by a DSU triplet, where each letter stands for the Differentiable, Separable, and Unimodal feature.

to three are preferable for 2D and 5D problems. This is a reasonable conduct because when MHs explore bigger domains, different mechanisms to perform such a work prevent a biased exploration. In particular, the HH model generated MH with four and five operators when solving multimodal problems, also with two and five dimensions. A similar effect is observed in Figure 14 when the problem dimensionality is incremented. Another amusing fact is observed with experiment 3, Figure 14(c), where well-defined sequences of operator families are attained. Remember that for this implementation, the HH model disposes of much more search operators per family than in the other experiments. It is worth noting that perturbators based on a central force dynamic, firefly dynamic, genetic mutation, spiral dynamic, and swarm dynamic are the most oft-deemed by the framework for all the problem instances. These five operators have a common characteristic, *i.e.,* their mathematical formulation is rather sophisticated, see Sect. 4.1. Conversely, simple SOs like random sample and random search are disregarded.

Besides, Table 6 shows the metaheuristics obtained by implementing the hyper-heuristic model with the short heuristic collection and the maximum cardinality of three, *i.e.,* experiment 1. For the sake of brevity,

we avoid showing detailed results from the other experiments. This table displays the category and function identification, according to Table 1, in the first two columns. The remaining seven pairs of columns correspond to the results for each dimension. The sequence of search operators $\boldsymbol{h}$ and the performance $F(\text{MH})$ achieved for the metaheuristic MH build with $\boldsymbol{h}$ constitutes each pair of columns per dimension. Take into account that $F(\text{MH})$ is the sum of the median (med) and the interquartile range (iqr) of the fitness values retrieved by the metaheuristic after 30 runs, cf. Equation (4). Thence, if the iqr $\to 0$ then $F(\text{MH}) \to$ med, and it is likely that med $= f(\vec{x}_*)$, since $\vec{x}_*$ corresponds to the optimum. An example of this is observed in the first row of Table 1, which corresponds to the category of differentiable, separable, and unimodal two-dimensional functions; it can also be noticed along the same column for other categories. However, we cannot confirm if an MH reaches the optimum solution, as we assume that problems have no known solution akin to a practical scenario. Another reason for it is that a few of the studied benchmark functions have either unknown or known-but-approximate optimum values. Indeed, there are several discrepancies in the technical reports about benchmark functions that make difficult their implementation, but this is topic escapes our scope.

Lastly, we evaluate the results by carrying out multiple comparisons using the pairwise (one-sided) Wilcoxon signed-rank test. According to the procedure and conditions given by García et al. [64], we compute the true statistical significance for multiple pairwise comparisons such as $p = 1 - \prod_i (1 - p_i)$. Besides, the performance value achieved by the metaheuristics, for each problem and dimension via (4), was employed as the measure of the algorithm's run. We set the proposed HH-based approach as the control parameter and a portion of the basic metaheuristics as the other parameters. Therefore, we respectively state the null and alternative hypotheses as follows:

$\mathbf{H}_0$: The customised metaheuristics, generated by the hyper-heuristic framework, perform equal to or worse than m (≥ 1) basic metaheuristics with a significance level $\alpha = 0.05$.

$\mathbf{H}_1$: The customised metaheuristics, generated by the hyper-heuristic framework, outperform m (≥ 1) basic metaheuristics with a significance level of $\alpha = 0.05$.

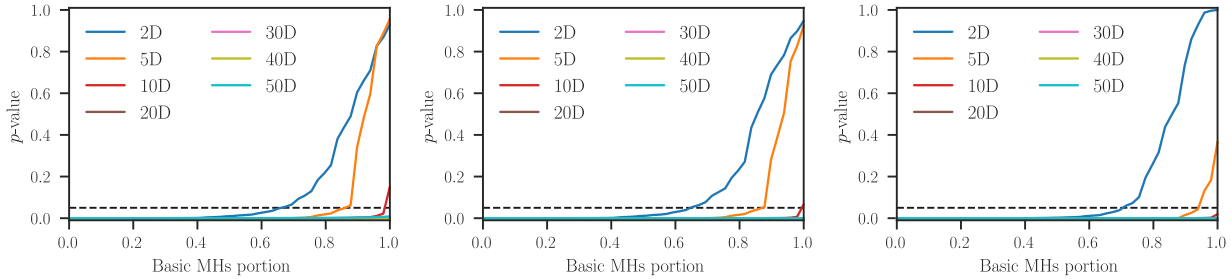
Figure 15 presents the p-values for this Wilcoxon's analysis for each number of dimensions and by varying the portion of basic metaheuristics (*i.e.,* $m/66$). A striking fact is that there is a reason ($p < 0.05$) to reject the null hypothesis when a portion of basic MHs of at most 63% is considered, for all dimensional values. Particularly, we fail to reject $\mathbf{H}_0$ for two-dimensional problems when considering the remaining portion MHs. It is a somewhat expected outcome because many metaheuristics proposed in the literature are powerful enough to tackle that kind of problems with an astounding level of performance. Indeed, we use the adverb '*somewhat*' mainly since it would be understandable if such a remaining portion would be higher than 37%, but it is not. Table 7 details the maximum portion values that make possible to accept $\mathbf{H}_1$. It is interesting to remark that the second implementation of the HH model produced slightly worst metaheuristics for 2D than the first one (*i.e.,* a difference of two MHs), although their p-value data seem quite similar as shown in Figures 15(a)–(b). Nevetheless, the third deployment of the HH framework, which utilised the long heuristic collection and a maximum cardinality of three, allow us to accept $\mathbf{H}_1$ by comparing the resulting MHs with the 69%, 94%, and 100% of basic metaheuristics for 2, 5, and 10-50 dimensions, respectively It is naturally an enhancement compared against the results from the other implementations of the proposed strategy. Note that while the null hypothesis cannot be rejected for specific portions, it does not imply that our methodology obtained poor-performing metaheuristics. By the contrary, we found customised MHs that surmounted at least one basic metaheuristic. Obviously, the larger the portion of basic metaheuristics, the better the power of the proposed framework.

5. Conclusions and future works

This work introduces an hyper-heuristic (HH) framework based on Simulated Annealing (SA) for solving continuous optimisation problems by building customised population-based metaheuristics (MHs). The HH model constructs these metaheuristic by searching within the heuristic space a combination of ϖ operators, and assembling them in a predefined MH scheme. These operators, which serve as building blocks, were extracted from common MHs from the literature. The primary motivation to develop such a framework is to halt the frenetic trend of publishing metaheuristics with similar operations but disguised with creative metaphors. So, we aim to provide a structured and innovative way to find a suitable method for solving a given problem, only using mathematical operations over a population.

Table 6. Detailed metaheuristics obtained with the short heuristic collection and cardinalities up to three. $\boldsymbol{h}$ is the sequence of search operators used to build the MH, and F(MH) is the corresponding performance.

Cat.	Id.	$\boldsymbol{h}$ 2D	F(MH)	$\boldsymbol{h}$ 5D	F(MH)	$\boldsymbol{h}$ 10D	F(MH)	$\boldsymbol{h}$ 20D	F(MH)	$\boldsymbol{h}$ 30D	F(MH)	$\boldsymbol{h}$ 40D	F(MH)	$\boldsymbol{h}$ 50D	F(MH)
111	10	194, 101, 189	4.6e-37	18, 137, 98	1.1e-22	87, 5, 92	75.52	134, 179, 155	1.2e6	136, 135, 196	9.7e5	157, 180, 114	5.2e7	164, 162, 105	1.4e10
	22	11, 99	1.3e-54	98, 137	3.3e-19	146, 158	0.02	107, 202, 100	0.11	120, 145, 133	0.08	135, 22, 10	0.20	162, 135, 6	4.25
	33	187, 107	0.00	204, 110, 105	2.9e-13	113, 147, 179	1.5e-9	96, 25, 120	0.64	79, 135, 115	0.19	200, 158, 100	2.20	170, 174, 98	13.04
	53	102, 204, 123	4.2e-44	81, 199, 202	6.1e-17	87, 58, 165	5.8e-6	181, 196, 66	7.6e-15	204, 100, 170	9.4e-11	188, 196	3.0e-6	14, 196	1.4e-5
	56	26, 99	3.2e-130	32, 117, 202	2.2e-17	12, 196	5.1e-21	135	4.0e-6	100, 172, 18	9.8e-6	195, 190	0.14	100, 32	14.78
	62	138	7.6e-13	14, 143, 104	5.9e-10	72, 203, 4	0.09	139, 145, 99	0.91	154, 135, 35	16.21	99, 167, 19	797.22	168, 111, 18	2.2e4
	73	108, 19, 81	4.1e-88	199, 18, 104	2.1e-53	61, 115, 175	8.5e-3	75, 193, 184	2.8e-6	83, 23, 177	7.27	119, 171, 19	218.14	97, 204, 179	17.50
	80	53, 204, 124	4.3e-179	107, 138	9.8e-35	150, 122, 136	4.4e-13	179, 159, 177	1.3e-6	173, 172, 102	2.3e-4	82, 89, 172	474.83	103, 202, 199	102.13
	83	98, 53, 195	2.7e-80	186, 195, 91	2.5e-31	180, 101, 20	9.3e-19	85, 58, 204	0.25	113, 140, 176	16.12	87, 22, 177	0.30	199, 174, 104	52.79
	91	146, 142	8.2e-24	148, 100	7.3e-8	4, 180, 26	0.01	173, 51, 96	2.02	133, 182, 194	0.28	102, 179, 202	3.63	54, 173, 134	58.26
110	4	116, 146, 162	-6.13	125, 99	-113.42	100, 139, 160	-6.7e3	135	-2.7e6	136	-6.6e9	133	-4.2e13	21, 74	7.0e3
	5	9, 115	0.00	160, 194	10.0e-4	3, 133	0.35	150, 134	2.56	65, 116, 203	15.59	133	10.25	179, 135, 18	47.69
	16	160, 93	-1.00	117, 155, 196	-1.00	154, 92	-0.96	133	-0.99	153, 166, 69	-0.95	66, 169, 99	-0.94	28, 103	-0.79
	17	192, 49, 102	-1.00	9, 101	-1.00	165, 113, 53	-1.00	133, 135, 99	-1.00	135, 54, 41	-0.99	4, 36, 86	-0.76	179, 10, 47	-0.82
	30	195	0.06	141, 196, 76	-0.74	97, 98, 201	-2.08	176, 60, 97	-4.75	162, 1, 134	-7.41	54, 137, 149	-7.78	99, 70, 192	-11.63
	41	19, 51, 90	-1.80	59, 127, 83	-4.58	76, 16, 92	-8.03	155, 83	-14.91	116, 83, 192	-19.49	66, 150, 18	-20.99	136	-33.70
	55	145, 183	1.3e-10	146, 120, 137	1.5e-29	147, 107, 154	0.26	196, 180, 138	0.02	119, 99, 173	1.3e4	136, 2, 146	3.5e3	170, 202, 196	2.0e5
	57	7, 103	0.00	111, 147, 34	2.3e-5	146, 194	0.04	102, 98, 175	10.90	194, 3, 139	19.84	134, 150	22.22	61, 7, 175	2.4e3
	59	188, 105	0.00	129, 53, 91	1.43	136	5.45	134, 96, 160	33.43	54, 195, 201	78.11	117, 14, 167	80.81	136, 162	24.29
	70	91, 18	-18.10	130, 154	-475.41	15	2.32	188	0.08	185	5.4e-4	192, 1	1.8e-5	5	4.6e-7
	71	159, 120, 157	-29.68	19, 18, 70	-68.32	51, 201, 88	-126.47	69, 166	-138.40	154, 85, 172	-255.29	157, 88, 159	-350.95	196, 82, 171	-349.10
	72	67, 203	-25.74	99, 153, 175	-57.61	81, 158	-92.68	195, 203	-147.74	52, 192	-161.73	1, 169, 51	-215.68	88, 155, 34	-299.71
	81	117, 199	0.00	8, 96	2.0e-4	2, 202	0.03	142, 9, 197	11.06	26, 146, 176	14.56	204, 84, 148	29.55	10, 140, 171	30.38
	90	195, 194	-78.33	198, 26	-194.16	181, 139, 195	-360.90	173, 65	-681.08	83, 140, 174	-1.1e3	95, 18, 180	-1.3e3	134, 202, 71	-1.7e3
	98	195, 12	0.00	67, 133, 101	0.14	74, 133	0.30	135	0.11	153, 69, 2	0.47	27, 107	0.29	15, 16, 117	0.42
	99	162, 28, 116	4.00	192, 104	40.08	138, 202	183.12	135, 170	762.43	135, 50, 20	1.7e3	135	3.1e3	99, 168	4.9e3
	105	108, 145, 134	0.00	107, 137, 133	2.27	167, 116, 178	19.75	158, 140, 97	33.33	135	14.89	99, 170, 198	101.77	1, 83, 196	190.44
101	6	197, 54	1.4e-87	148	2.6e-5	147, 95	3.2e-3	195, 147, 166	1.2e-5	81, 154, 170	0.11	178, 134	0.24	199, 133, 196	4.52
	7	6, 85, 17	9.9e-31	79, 145	1.4e-4	99, 188, 1	1.2e-4	194, 145	3.8e-5	135, 87, 137	0.02	104, 18, 192	5.37	84, 179, 145	4.88
	9	195, 193, 41	1.1e-160	117, 204, 66	8.0e-50	18, 196, 163	1.6e-30	160, 23, 114	0.25	176, 53, 105	741.72	12, 196, 190	601.36	101, 27, 170	2.0e3
	19	42, 193	2.5e-32	26, 107, 91	9.2e-15	191, 86, 196	0.55	169, 115, 70	1.38	150, 195, 26	14.98	99, 203, 179	20.27	168, 136	7.99
	28	139, 102	-1.00	194, 130	-1.00	61, 178, 12	-1.00	196, 203	-1.00	200, 120	-0.94	133, 166	-1.00	155, 104, 16	-0.80
	36	193, 99, 5	5.4e-41	174, 137, 112	-5.12	116, 112, 140	-10.24	195, 140	-18.05	82, 148, 135	3.14	98, 155	1.1e5	18, 134, 158	19.25
	43	7	2.00	25	2.00	190, 16	2.00	138, 6, 123	2.00	9, 150	2.00	111, 189	2.00	31, 166, 188	2.00
	50	101, 28	9.9e-32	193, 135, 10	3.86	99, 1, 174	1.4e13	23, 193, 47	1.6e46	163, 14, 167	7.4e82	98, 166, 123	2.6e122	195, 7, 1	3.3e163
	51	193	0.00	136	0.05	11, 1, 174	1.27	154, 134	19.30	139, 179, 120	4.1e32	158, 24, 98	2.1e86	175, 117, 203	2.5e86
	60	189, 109	-2.24	146, 120, 155	-2.24	139, 193	-2.24	145, 194	-2.24	36, 91, 193	-2.21	192, 101	-2.03	133, 126	-2.22
	74	114, 146	7.1e-42	118, 5, 22	2.3e-7	197, 194, 146	2.2e-8	99, 194, 139	0.05	148, 97, 26	2.42	135, 133	2.09	154, 134, 127	4.00
	75	89, 19, 58	2.8e-5	196, 193, 24	1.1e-19	134, 3	0.33	155, 24, 116	175.10	135, 170, 11	195.39	116, 96, 19	7.4e3	151, 134, 121	264.96
	76	125, 71, 199	0.00	178, 16, 90	1.6e-8	102, 4, 201	4.3e-3	96, 195, 137	0.06	174, 199, 140	22.88	204, 184, 113	13.30	103, 134, 190	58.06
	92	202	-2.00	134	-29.97	90, 10, 168	-209.58	90, 150, 23	840.37	202, 23, 111	1.7e4	199, 171, 103	7.5e4	115, 174, 166	2.4e5
	103	111, 199, 125	-1.00	143, 2	2.8e-39	181	7.6e-78	108, 26	5.7e-155	94, 22	4.3e-232	115, 126, 144	infe-309	190	0.00
	106	96, 26	1.9e-29	18, 155, 90	5.8e-9	22, 154, 94	3.4e-4	18, 83, 177	1.44	15, 196, 150	140.48	193, 30, 147	279.56	172, 160, 134	243.79
100	1	100, 10	2.1e-8	147, 44, 4	0.06	175, 75	7.78	138, 133	0.81	153, 104, 136	1.90	17, 107, 25	4.77	116, 25	0.00
	2	101, 22	-3.30	7, 148, 85	-11.54	136	-19.99	101, 159, 8	-7.68	154, 173, 101	-16.92	115, 139	-1.05	133	-17.09
	18	28, 111, 13	-1.00	151, 118	-0.84	204, 115	-0.84	169, 135	-0.37	100	1.51	70, 176, 155	2.54	133, 158	0.41
	20	26, 196, 22	-1.00	132, 193	-0.94	194, 12, 34	-0.78	160, 193	-0.17	133	-0.23	162, 58, 193	-0.09	132, 116, 162	-0.08
	21	42, 197, 162	-959.64	14, 138, 158	-3.2e3	6, 76, 203	-5.9e3	1, 75, 26	-9.8e3	84, 149, 160	-1.7e4	65, 139, 166	-1.9e4	203, 75, 7	-2.1e4
	29	145, 90, 182	-1.00	142, 201, 118	-0.03	20	1.1e-12	192, 202	1.2e-35	96	6.3e-58	201, 188	2.9e-81	136, 187	6.3e-105
	31	96, 204	0.02	195, 128	0.29	151, 134	0.17	136	0.14	193, 149, 18	0.13	98	4.73	135, 130	0.96
	32	193	0.03	100, 200	0.17	153, 4, 196	0.39	19, 108, 71	2.65	133, 182	0.73	112, 81, 190	65.25	134	1.07
	34	195, 127	-1.00	195, 167	-2.95	132, 138, 47	-4.81	147, 136	-9.78	84, 55, 131	-14.20	152, 86	-11.61	133, 156	-34.47
	35	108, 26	124.36	68, 1, 188	13.38	71, 204, 19	23.95	88, 183, 120	74.90	150, 19, 10	140.72	114, 148, 18	237.61	102, 26, 199	387.51
	40	162, 146	9.3e-12	68, 129, 190	9.81	164, 65, 193	26.56	136, 52, 140	62.76	155, 97, 146	91.05	122, 97, 28	216.13	168, 99, 131	330.85
	42	65, 39	2.00	187	2.00	23, 132	2.00	68, 0, 183	2.00	138, 140	2.00	64, 186, 156	2.00	14, 147	2.00
	44	140	1.5e-10	41, 158	5.2e-8	152, 177, 119	0.00	154, 108, 159	1.4e11	24, 32, 19	7.0e64	186	6.7e95	95, 170, 17	9.3e128
	48	204, 118	0.50	4, 75	2.00	50, 12, 35	4.50	14	9.50	96, 71, 188	14.50	52, 76, 21	19.50	145	24.50
	49	113, 17	0.90	93, 116, 168	1.00	86, 82, 165	1.01	135, 105, 24	1.05	135	1.04	80, 178, 18	1.11	27, 107, 178	0.90
	52	105, 184, 185	0.00	74, 96, 18	2.4e-16	134, 145	1.04	138, 107, 7	526.31	101, 152, 140	3.2e3	137, 114, 150	5.5e3	124, 136, 163	3.0e3
	58	1, 19, 171	-625.31	89, 204	-2.2e3	20, 4, 199	-4.9e3	84, 185, 7	-9.8e3	190, 159, 142	-1.5e4	187, 157, 143	-2.0e4	201, 128, 183	-2.2e4
	61	26, 22	3.1e-8	112, 199	4.78	135, 105, 204	57.06	182, 196, 178	71.36	183, 167, 195	402.58	135, 154	133.94	178, 133, 172	382.54
	63	173, 18, 4	0.07	169, 72, 105	0.11	194, 204	0.28	102, 11, 170	0.75	202, 175, 103	1.85	178, 193, 159	2.10	113, 154, 65	14.02
	64	119, 138	1.8e-32	101, 11, 146	2.7e-36	148, 145, 112	0.06	152, 147, 195	4.18	196, 179, 147	13.48	159, 23, 100	1.6e3	201, 195, 41	5.4e4
	65	160, 108, 9	-0.28	78, 27, 189	-0.64	12, 165, 84	0.50	13, 1, 107	2.61	6, 84	5.00	159, 52	2.94	67, 152	7.56
	66	43, 199, 200	-0.28	201, 192, 65	-1.00	24, 79	-0.22	204, 72, 74	0.23	6, 142	4.00	16, 182, 49	5.43	26, 178	10.49
	69	102, 158, 187	-0.28	199, 77, 112	-1.12	175, 195, 187	-2.53	190	-4.71	185, 120, 122	-6.01	134, 16, 186	-10.83	186, 76, 8	-11.15
	89	106, 147, 19	0.00	50, 170	0.23	204, 71, 95	6.8e-4	16, 115, 158	1.00	158, 114, 137	1.25	136	5.93	103, 140, 182	34.60
	93	125	0.00	65, 132, 151	0.00	62, 35, 75	0.00	203	0.00	27, 138, 190	0.00	155, 12	0.00	65, 137, 193	2.5e4
	94	147	1.00	74, 197, 98	1.00	18, 196	21.89	23, 182, 195	132.95	135, 133	250.36	133, 170	394.61	95, 195, 128	1.5e5
	97	198, 59, 159	-2.00	12, 95	-5.00	102, 188, 10	-10.00	88, 176	-19.95	175, 73	-24.71	135, 194	-39.99	73, 173	-35.18
	100	7, 120, 127	-3.97	195, 199	-18.10	140, 106, 18	-48.49	136, 4	-103.0	98, 171, 53	34.12	136, 150, 39	-194.14	138, 134	38.26
011	77	115, 146	2.4e-42	66, 27, 113	1.3e-8	22, 179	0.73	68, 199, 67	1.03	100, 140, 170	4.32	139, 134, 113	9.89	133, 134	11.84
	78	195	2.4e-19	193, 26	4.5e-13	122, 119, 129	12.59	133, 10	0.77	196, 167, 186	25.38	132	10.34	176, 107, 1	35.83
	84	114	0.00	10, 45, 197	0.00	116, 135	0.00	165, 160, 117	0.00	139, 142, 195	5.00	184, 97, 18	127.00	10, 113, 199	78.50
	85	136	0.00	136	0.00	152, 81, 200	0.00	204, 102, 170	1.00	29, 195, 120	2.0e3	147, 179, 97	15.00	103, 139, 169	118.25
	86	160, 46	0.00	179	0.00	4, 67, 8	0.00	7, 193, 47	0.00	81, 112, 202	34.00	174, 113, 86	52.50	104, 170, 202	27.75
	87	20, 77, 169	13.00	75, 183	-5.00	148, 172	-35.00	22, 160	-95.00	202	-155.00	186, 9	-215.00	147, 158	-275.00
010	3	23, 110	0.00	105, 51, 200	7.6e-14	105, 24, 73	2.4e-3	140, 194	0.09	50, 138, 133	0.27	136, 168	0.30	83, 108	40.36
	11	36	-2.20	137	-5.50	95, 34, 180	-11.00	10	-22.00	156, 73, 23	-33.00	139, 138	-44.00	26, 117, 153	-55.00
	15	53, 193, 197	2.9e-220	79, 118, 33	0.00	179, 166, 186	3.0e-15	169, 80, 101	2.4e-9	108, 37	0.00	28, 196, 104	0.03	111, 13	0.00
	25	110, 160, 14	0.00	104, 192, 52	2.7e-4	94, 178, 19	26.52	202, 147, 89	232.26	157, 193, 198	226.61	135, 151, 74	946.70	180, 157, 120	1.4e3
	26	148, 71, 184	0.00	156, 147	16.67	86, 197, 69	296.16	102, 203, 7	408.19	105, 120, 176	654.14	166, 99, 203	876.01	107, 198, 18	1.9e3
	54	195, 112, 163	0.00	186, 55, 196	0.00	138, 82, 194	1.1e-7	195, 130, 16	0.79	174, 147, 117	142.31	23, 133	173.58	119, 114, 172	1.9e3
	82	198, 193	-490.09	17, 2, 21	-433.80	172, 62, 10	-350.44	134, 82, 154	-305.32	185, 52	-247.23	98, 147, 202	-246.55	203, 185, 105	-229.86
	88	118, 50, 21	8.4e-15	108, 126, 202	2.0e-3	20, 130, 68	0.45	101, 171, 162	2.82	14, 24, 101	3.31	135	4.20	61, 200, 118	7.19
	101	13, 104, 41	4.8e-27	69, 176, 124	7.8e-6	97, 79, 28	7.4e-8	135, 73, 103	3.5e-8	174, 98, 148	0.02	26, 136, 174	4.1e-5	133, 136	2.7e-4
001	79	103, 166, 156	5.4e-5	188, 108, 67	3.9e-3	200, 64, 196	6.1e-6	133, 51, 200	4.64	27, 77, 98	104.32	102, 133, 137	18.76	139, 103, 137	329.45
	95	201, 72	1.7e-14	198, 58	5.5e-3	148, 105	1.5e-3	133, 66, 85	3.4e-3	54, 6, 173	0.03	52, 146	0.22	117, 153, 174	0.05
000	8	158, 193, 82	-24.16	155, 196	-3.3e3	138, 67, 180	-8.7e5	150, 153, 88	-6.7e7	74, 170, 15	-7.7e11	170, 159	-2.6e11	140, 1, 202	-9.7e10
	12	196	-2.06	103, 200, 166	-1.98	174, 116	-1.80	24, 107, 98	-1.04	149, 96, 69	-0.53	99, 134	-0.54	153, 99	-0.28
	13	117, 35	-1.00	98, 116, 24	-1.00	146, 111, 161	-1.00	110, 9	-1.00	119, 42, 15	-1.00	106, 189	-1.00	3, 14, 68	-1.00
	14	84, 200, 59	1.2e-3	195, 49	2.8e-4	111, 188	1.0e-4	139, 118	1.0e-4	145, 85, 178	1.0e-4	73, 85, 195	1.0e-4	6, 109	1.0e-4
	23	140, 59, 16	6.1e-3	138, 91, 154	0.46	150, 10, 178	1.97	133	15.94	134, 146	24.97	106, 166, 198	188.48	178, 147, 98	46.73
	24	100, 74, 15	6.6e-4	147	0.42	204, 118, 195	2.8e-8	176, 4, 65	14.44	2, 133	21.34	172, 116, 202	48.42	109, 83, 166	6.5e3
	27	55, 200	-4.00	90, 18	-10.00	28, 196, 88	-19.81	146, 97, 202	-32.19	26, 100, 179	-34.91	98, 174, 139	-28.17	136, 166	-61.42
	37	44, 154	1.00	186	1.00	38, 119	1.00	19, 152	1.00	139, 171	1.00	160, 199	1.00	203, 38	1.00
	38	195, 16, 87	1.5e-32	109, 193, 141	8.4e-22	182, 135	5.2e-3	176, 101, 63	1.92	71, 27, 115	14.08	135, 134, 165	0.93	176, 90, 22	23.41
	39	174, 70, 63	1.0e-7	138	0.16	175, 102	10.00	13, 135, 117	20.01	170, 98, 55	37.47	84, 10, 168	40.09	59, 100, 198	80.98
	45	197, 43	0.00	161, 8, 148	0.00	182, 153, 193	0.00	14, 198, 141	0.00	136, 144, 195	4.7e-10	83, 177, 11	4.1e-7	168, 193, 87	3.1e-4
	46	138	1.6e-8	154, 109, 153	0.04	159, 190, 119	0.19	196, 145, 158	0.07	117, 171, 186	0.65	170, 118	1.62	120, 178, 204	0.54
	47	114, 106, 23	1.00	187, 185, 107	1.00	101, 190, 128	299.49	25, 108	843.22	82, 136, 1	2.6e3	85, 165, 51	3.9e3	28, 27, 108	2.4e3
	67	196	1.6e-3	129, 68	0.33	195, 56, 61	2.83	84, 157, 98	5.91	172, 96, 162	11.23	69, 129	12.65	114, 108, 192	19.87
	68	160, 113, 147	0.29	195, 157	1.49	129, 64, 178	3.99	44, 193, 71	8.87	146, 74	13.66	198, 52, 196	18.47	153, 41, 66	23.61
	96	99, 138, 14	1.8e-7	150, 46	0.09	118, 97, 23	0.11	159, 195	0.19	169, 66	0.20	140, 109, 50	0.23	120, 106, 144	0.24
	102	145, 99	1.3e-31	94, 30, 84	0.06	136, 97	7.0e-4	154	3.3e-7	18, 117, 133	2.6e-11	4, 97, 182	1.8e-15	74, 115, 188	1.0e-19
	104	92, 18	-1.00	174, 110, 151	0.03	85, 54, 136	4.5e-4	98, 137	4.2e-8	57, 53, 166	5.9e-12	136, 46	8.5e-16	170, 71	7.5e-20
	107	44, 12	0.00	193	1.00	26	0.00	191	0.00	144, 0	0.00	144	0.00	159, 119	1.00



(a) Experiment 1 (b) Experiment 2 (c) Experiment 3

Figure 15. True statistical significance (p-value) obtained from Wilcoxon's test for multiple comparisons. Metaheuristics generated by the hyper-heuristic model were set as the control parameter, whilst a portion of basic metaheuristics was used for second parameters. Columns correspond to the experiments carried out with the short and long collection, and the maximum cardinality of three and five. Dashed black line represents the level of significance $\alpha = 0.05$.

Table 7. Maximum portions (in %) of the basic MHs, used in the multi-comparison Wilcoxon's analysis with the customised MHs as the control parameter, which permit the null hypothesis ($\mathbf{H}_0$) rejection. The customised MHs were obtained from the HH model under three different conditions (experiments).

Dim.	2	5	10	20–50
Exp. 1	65.31	85.71	97.96	100.0
Exp. 2	63.27	85.71	97.96	100.0
Exp. 3	69.39	93.88	100.0	100.0

To study this model, we first defined in detail all the concept employed to develop either the framework and metaheuristics. This aspect is really important to take in mind because several foundations related with this work are currently matter of discussion. Then, we analysed ten well-known metaheuristics and extracted their search operators (SOs). With this information, we generated two collections, one with 205 SOs and another with 1274033 SOs, which we labeled as short and long collections, respectively. In addition, we constructed a collection with 66 predefined metaheuristics derived from those which were analysed early on. To carry out all the experiments, we selected and categorised 107 continuous benchmark functions using three features: differentiability, separability, and unimodality. Subsequently, we implemented the SOs from the short collection as MHs of $\varpi = 1$ for solving all the functions, and by varying their dimensionalities (*i.e.,* 2, 5, 10, 20, 30, 40, and 50), as a preliminary stage. From the achieved results, we detected the suitability of each search operator when solving problems with similar features. Likewise, we repeated the same experiment but using the collection of predefined MHs, namely basic metaheuristics. This provided a background to compare the achieved metaheuristics. Subsequently, we deployed three experiments of the HH model by utilising the short collection with a maximum allowed cardinality ($\varpi_{\max}$) of three and five, and the long collection with $\varpi_{\max}$ of three.

First of all, we tested the proposed model and identified its potential for devising metaheuristics to solve continuous optimization problems with different dimensionalities and characteristics. In other words, we provided an automatic way to find a suitable methodology for solving a problem. Such a task requires moderate expertise of, say, a practitioner who must know which basic MH choose for solving a given problem. This is not an easy task and a particular metaheuristic cannot fulfil the requirements for solving all the problems with a guaranteed high performance, *i.e.,* the NFL theorem. In fact, when we compared the tailored metaheuristics against the basic ones, the former outperformed the latter in at least 63% of the tests. We believe this to be worthwhile, as it removes the need of handcrafting a metaheuristic, as well as the corresponding limitation of requiring solid foundations to do so.

Our data yield the following insights: Using a short collection of SOs with *a priori* knowledge represents a low-cost implementation of the proposed HH model. However, increasing the maximum cardinality of the customised MHs does not improve much the performance of such methods. This naturally promotes the generation of diverse metaheuristics in terms of the types of search operators that they use. From all results, we noticed that the most popular SOs were, in general, those based on central force dynamic, genetic crossover, and swarm dynamic. Conversely, the most infrequent ones were those barely using simple mathematical

formulation such as the random sample, random search, and local random search.

This primary work lays the groundwork for future research. We plan on increasing the size of the heuristic families by including other well-known SOs from the literature. Also, we shall seek a way for considering cardinality and population size, for example, as design variables. This way, the HH would not only find a proper set of search operators, but also the set length that should be used and the number of agents that should be considered in the search. We also look forward to complementing the proposed strategy with Data Science techniques to enhance exploration of the heuristic space [65, 66]. This way, a more robust approach based on Machine Learning could be used when tuning the hyper-heuristic.

By using this building block scheme, we can think about multiple extensions, some of which are now mentioned. First, if we interpret the scheme proposed for a MH as a serial topology (using a circuit analogy), we could state MHs with different topologies such as parallel, star, delta, Y, T, and so forth. In the case of a metaheuristic with a parallel topology, we can enclose some well-known approaches from the literature, *e.g.,* those employing sub-populations or neighbourhoods such as Particle Swarm Optimisation [32]. But in such a case, we would not be limited to using the same search operators for each neighbourhood. In addition, the topology assumption considers that all agents perform the same number of operations (cardinality). However, our strategy is flexible enough as to accept different operations for each branch in the MH scheme.

Lastly, taking in mind the notation MH^{ϖ} where ϖ is the metaheuristic's cardinality or order, we plan to explore using one or more operators for a limited number of iterations during the search process, so that ϖ could be real instead of being restricted to integers. In other words, we will consider including "fractional" metaheuristics in hyperheuristic processes.

References

[1] K. Sörensen, M. Sevaux, F. Glover, A history of metaheuristics, Handbook of Heuristics 2-2 (2018) 791–808. doi:10.1007/978-3-319-07124-4_4.

[2] K. Hussain, M. N. M. Salleh, S. Cheng, Y. Shi, Metaheuristic research: a comprehensive survey, Artificial Intelligence Review 52 (4) (2019) 2191–2233.

[3] S. P. Adam, S.-A. N. Alexandropoulos, P. M. Pardalos, M. N. Vrahatis, No Free Lunch Theorem : A Review, in: I. Demetriou, P. Pardalos (Eds.), Approximation and Optimization, Springer, Cham, 2019, pp. 57–82. doi:10.1007/978-3-030-12767-1_5.

[4] K. Sörensen, Metaheuristics—the metaphor exposed, International Transactions in Operational Research 22 (1) (2015) 3–18. doi:10.1111/itor.12001.

[5] T. Dokeroglu, E. Sevinc, T. Kucukyilmaz, A. Cosar, A survey on new generation metaheuristic algorithms, Computers & Industrial Engineering 137 (2019) 106040.

[6] C. W. Ahn, Practical genetic algorithms, Vol. 18, Wiley-Interscience, 2006. doi:10.1007/11543138_2.

[7] R. Storn, K. Price, Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces, Journal of global optimization 11 (4) (1997) 341–359.

[8] X.-S. Yang, S. Deb, Cuckoo search via Lévy flights, in: 2009 World Congress on Nature & Biologically Inspired Computing (NaBIC), IEEE, 2009, pp. 210–214.

[9] J. Del Ser, E. Osaba, D. Molina, X.-S. Yang, S. Salcedo-Sanz, D. Camacho, S. Das, P. N. Suganthan, C. A. C. Coello, F. Herrera, Bio-inspired computation: Where we stand and what's next, Swarm and Evolutionary Computation 48 (2019) 220–250.

[10] E. K. Burke, M. R. Hyde, G. Kendall, G. Ochoa, E. Özcan, J. R. Woodward, A classification of hyper-heuristic approaches: revisited, in: Handbook of Metaheuristics, Springer, 2019, pp. 453–477.

[11] N. Pillay, R. Qu, Hyper-Heuristics: Theory and Applications, Springer, 2018.

[12] I. Amaya, J. C. Ortiz-Bayliss, A. Rosales-Pérez, A. E. Gutiérrez-Rodríguez, S. E. Conant-Pablos, H. Terashima-Marín, C. A. Coello Coello, Enhancing Selection Hyper-Heuristics via Feature Transformations, IEEE Computational Intelligence Magazine 13 (2) (2018) 30–41.

[13] I. Amaya, J. C. Ortiz-Bayliss, S. Conant-Pablos, H. Terashima-Marin, Hyper-heuristics Reversed: Learning to Combine Solvers by Evolving Instances, in: 2019 IEEE Congress on Evolutionary Computation (CEC), IEEE, 2019, pp. 1790–1797. doi:10.1109/CEC.2019.8789928.

[14] P. B. Miranda, R. B. Prudêncio, G. L. Pappa, H3ad: A hybrid hyper-heuristic for algorithm design, Information Sciences 414 (2017) 340–354.

[15] T. Abell, Y. Malitsky, K. Tierney, Fitness Landscape Based Features for Exploiting Black-Box Optimization Problem Structure, Tech. Rep. December, IT University, Copenhagen (2012).

[16] L. Cao, The cat that catches mice: China's challenge to the dominant privatization model, Brook. J. Int'l L. 21 (1995) 97.

[17] M. Jamil, X. S. Yang, A literature survey of benchmark functions for global optimisation problems, International Journal of Mathematical Modelling and Numerical Optimisation 4 (2) (2013) 150. doi:10.1504/IJMMNO.2013.055204.

[18] S. S. Rao, Engineering optimization: theory and practice, John Wiley & Sons, 2009.

[19] R. W. Garden, A. P. Engelbrecht, Analysis and classification of optimisation benchmark functions and benchmark suites, Proceedings of the 2014 IEEE Congress on Evolutionary Computation, CEC 2014 1 (2014) 1641–1649. doi:10.1109/CEC.2014.6900240.

[20] J. M. Dieterich, B. Hartke, Empirical Review of Standard Benchmark Functions Using Evolutionary Global Optimization, Applied Mathematics 03 (10) (2012) 1552–1564. arXiv:1207.4318, doi:10.4236/am.2012.330215.

[21] B. Y. Qu, J. J. Liang, Z. Y. Wang, Q. Chen, P. N. Suganthan, Novel benchmark functions for continuous multimodal optimization with comparative results, Swarm and Evolutionary Computation 26 (2016) 23–34. doi:10.1016/j.swevo.2015.07.003.

[22] G. Woumans, L. De Boeck, J. Beliën, S. Creemers, A column generation approach for solving the examination-timetabling problem, European Journal of Operational Research 253 (1) (2016) 178–194.

[23] D. Goldberg, J. Holland, Genetic algorithms and machine learning, Machine learning 3 (2-3) (1988) 95–99. doi:10.1023/A:1022602019183.

[24] M. Dianati, I. Song, M. Treiber, An introduction to genetic algorithms and evolution strategies, Sadhana 24 (4-5) (2002) 293–315.

[25] S. Kirkpatrick, C. D. Gelatt Jr., M. P. Vecchi, Optimization by Simulated Annealing Optimization by Simulated Annealing, Science 220 (4598) (1983) 671–680.

[26] A. Franzin, T. Stützle, Revisiting simulated annealing: A component-based analysis, Computers and Operations Research 104 (2019) 191–206. doi:10.1016/j.cor.2018.12.015.

[27] D. Delahaye, S. Chaimatanan, M. Mongeau, Simulated Annealing: From Basics to Applications, in: M. Gendreau, J.-Y. Potvin (Eds.), Handbook of Metaheuristics, 3rd Edition, Springer, 2019, Ch. 1, pp. 1–35. doi:10.1007/978-3-319-91086-4_1.

[28] S. Salcedo-Sanz, Modern meta-heuristics based on nonlinear physics processes: A review of models and design procedures, Physics Reports 655 (2016) 1–70. doi:10.1016/j.physrep.2016.08.001.

[29] K. Price, R. Storn, Differential evolution-A simple and efficient adaptive scheme for global optimization over continuous space, Technical Report, International Computer Science Institute (1995).

[30] S. Das, S. S. Mullick, P. N. Suganthan, Recent advances in differential evolution-An updated survey, Swarm and Evolutionary Computation 27 (2016) 1–30. doi:10.1016/j.swevo.2016.01.004.

[31] J. Kennedy, R. Eberhart, Particle swarm optimization (PSO), in: Proc. IEEE International Conference on Neural Networks, Perth, Australia, 1995, pp. 1942–1948.

[32] M. Clerc, J. Kennedy, The particle swarm-explosion, stability, and convergence in a multidimensional complex space, Evolutionary Computation, IEEE Transactions on 6 (1) (2002) 58–73.

[33] X.-S. Yang, et al., Firefly algorithm, Nature-inspired metaheuristic algorithms 20 (2008) 79–90.

[34] I. Fister, X.-S. Yang, J. Brest, A comprehensive review of firefly algorithms, Swarm and Evolutionary Computation 13 (2013) 34–46. doi:10.1016/j.swevo.2013.06.001.

[35] X. Yang, S. Deb, Cuckoo search: recent advances and applications, Neural Computing and Applications (2014) 1–9arXiv:1408.5316, doi:10.1007/s00521-013-1367-1.

[36] M. Shehab, A. T. Khader, M. A. Al-Betar, A survey on applications and variants of the cuckoo search algorithm, Applied Soft Computing Journal 61 (2017) 1041–1059. doi:10.1016/j.asoc.2017.02.034.

[37] R. A. Formato, Central force optimization: A new deterministic gradient-like optimization metaheuristic, Opsearch 46 (1) (2009) 25–51. doi:10.1007/s12597-009-0003-4.

[38] M. Behniya, A. H. Ayati, A. Derakhshani, A. Haghighi, Application of the central force optimization (CFO) method to the soil slope stability analysis, in: International Conference on Progress in Science and Technology, 2016, p. 11.

[39] R. A. Formato, Determinism in electromagnetic design & optimization—part ii: BBP-derived π fractions for generating uniformly distributed sampling points in global search and optimization algorithms, in: Forum for Electromagnetic Research Methods and Application Technologies (FERMAT), 2017, p. 6.

[40] J. González, I. Amaya, R. Correa, Design of an Optimal Multi-layered Electromagnetic Absorber through the Central Force Optimization Algorithm, PIERS Proceedings 1 (1) (2013) 1082–1086.

[41] K. Tamura, K. Yasuda, Primary study of spiral dynamics inspired optimization, IEEJ Transactions on Electrical and Electronic Engineering 6 (S1) (2011) S98–S100.

[42] J. M. Cruz-Duarte, I. Martin-Diaz, J. U. Munoz-Minjares, L. A. Sanchez-Galindo, J. G. Avina-Cervantes, A. Garcia-Perez, C. R. Correa-Cely, Primary study on the stochastic spiral optimization algorithm, in: 2017 IEEE International Autumn Meeting on Power, Electronics and Computing (ROPEC), 1, IEEE, 2017, pp. 1–6.

[43] E. Rashedi, H. Nezamabadi-pour, S. Saryazdi, GSA: A Gravitational Search Algorithm, Information Sciences 179 (13) (2009) 2232–2248. doi:10.1016/j.ins.2009.03.004.

[44] A. Biswas, K. K. Mishra, S. Tiwari, A. K. Misra, Physics-Inspired Optimization Algorithms: A Survey, Journal of Optimization 2013 (2013) 1–16. doi:10.1155/2013/438152.

[45] J. Bergstra, Y. Bengio, Random search for hyper-parameter optimization, Journal of machine learning research 13 (Feb) (2012) 281–305.

[46] A. R. Al-Roomi, Unconstrained Single-Objective Benchmark Functions Repository (2015).
URL `https://www.al-roomi.org/benchmarks/unconstrained`

[47] A. Gavana, Global Optimization Benchmarks and AMPGO (2013).
URL `http://infinity77.net/global_optimization`

[48] N. Hansen, S. Finck, R. Ros, A. Auger, Real-parameter black-box optimization benchmarking 2009: Noiseless functions definitions, Tech. rep., INRIA Saclay (2009).

[49] H. Pohlheim, Examples of objective functions, Retrieved 4 (10) (2007) 2012.

[50] M. Molga, C. Smutnicki, Test functions for optimization needs, Test functions for optimization needs 101 (2005).

[51] M. A. Ardeh, Benchmark function toolbox (2016).
URL `http://benchmarkfcns.xyz/about/`

[52] J. Sakuma, S. Kobayashi, Real-coded ga for high-dimensional k-tablet structures, Transactions of the Japanese Society for Artificial Intelligence 19 (2004) 28–37.

[53] H. Suzuki, H. Sawai, Chemical genetic algorithms-coevolution between codes and code translation, in: Proceedings of the Eighth International Conference on Artificial Life (Artificial Life VIII), 2002, pp. 164–172.

[54] F. Garza-Santisteban, R. Sanchez-Pamanes, L. A. Puente-Rodriguez, I. Amaya, J. C. Ortiz-Bayliss, S. Conant-Pablos, H. Terashima-Marin, A Simulated Annealing Hyper-heuristic for Job Shop Scheduling Problems, in: 2019 IEEE Congress on Evolutionary Computation (CEC), IEEE, 2019, pp. 57–64. doi:10.1109/CEC.2019.8790296.

[55] F. Garza-Santisteban, J. M. Cruz-Duarte, I. Amaya, C. Ortiz-Bayliss, S. E. Conant-Pablos, H. Terashima-Marin, Influence of Instance Size on Selection Hyper-Heuristics for Job Shop Scheduling Problems, in: 2019 IEEE Symposium Series on Computational Intelligence (SSCI), IEEE, Xiamen, China, 2019, p. 8.

[56] M. A. Schumer, K. Steiglitz, Adaptive Step Size Random Search, IEEE Transactions on Automatic Control 13 (3) (1968) 270–276. doi:10.1109/TAC.1968.1098903.

[57] R. N. Mantegna, H. E. Stanley, Stochastic process with ultraslow convergence to a gaussian: the truncated lévy flight, Physical Review Letters 73 (22) (1994) 2946.

[58] D. Zaharie, A Comparative Analysis of Crossover Variants in Differential Evolution, in: Proceedings of the International Multiconference on Computer Science and Information Technology, IMCSIT 2007, no. December in 1, 2007, pp. 171–181.

[59] A. K. Kar, Bio inspired computing–a review of algorithms and scope of applications, Expert Systems with Applications 59 (2016) 20–32.

[60] M. R. Bonyadi, Z. Michalewicz, Particle swarm optimization for single objective continuous space problems: a review (2017).

[61] M. Imran, R. Hashim, N. E. A. Khalid, An overview of particle swarm optimization variants, Procedia Engineering 53 (1) (2013) 491–496. doi:10.1016/j.proeng.2013.02.063.

[62] Y. Zhang, S. Wang, G. Ji, A comprehensive survey on particle swarm optimization algorithm and its applications, Mathematical Problems in Engineering 2015 (2015).

[63] J. S. Dai, Euler–Rodrigues formula variations, quaternion conjugation and intrinsic connections, Mechanism and Machine Theory 92 (2015) 144–152.

[64] S. García, D. Molina, M. Lozano, F. Herrera, 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, Journal of Heuristics 15 (6) (2009) 617–644.

[65] A. Jasuja, Feature selection using diploid genetic algorithm, Annals of Data Science 7 (1) (2020) 33–43.

[66] S. M. H. Bamakan, H. Wang, T. Yingjie, Y. Shi, An effective intrusion detection framework based on mclp/svm optimized by time-varying chaos particle swarm optimization, Neurocomputing 199 (2016) 90–102.

Declaration of interests

☒ The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

☐The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: