Automated Design of Unfolded Metaheuristics and the Effect of Population Size

Jorge M. Cruz-Duarte, Ivan Amaya, José Carlos Ortiz-Bayliss
Tecnologico de Monterrey
Monterrey, Mexico
{jorge.cruz, iamaya2, jcobayliss}@tec.mx

Nelishia Pillay
University of Pretoria
Pretoria, South Africa
npillay@cs.up.ac.za

***Abstract*—Metaheuristics are a fairly standard approach for solving optimisation problems due to their success, flexibility, and simplicity. However, there is a plethora of metaheuristics available, with different performance levels for various problems. This work proposes a methodology for designing heuristic-based procedures to solve continuous optimisation problems and study how the population size affects its performance. The technique comprises the well-known Simulated Annealing algorithm as a hyper-heuristic, and a heuristic sequence taken from unfolding the conventional scheme of population-based metaheuristics reported in the literature. Our results show that the proposed approach is a reliable alternative for tackling optimisation problems. We find exciting insights, according to our data, about this primary implementation when varying the population size in different challenging problems.**

***Index Terms*—Hyper-heuristic, Dimensionality, Search Operators, Continuous Optimisation, Evolutionary Computation.**

I. Introduction

METAHEURISTICS (MHs) are well-known strategies for solving optimisation problems. Their use is customary nowadays because of their proven success, flexibility, and simplicity. Besides, they usually are more straightforward than traditional approaches based on derivatives of the objective function. However, MHs exhibit some drawbacks. For starters, there is a plethora of options available, with different performance levels for different kinds of problems. Moreover, to accurately estimate the performance level of a given approach, extensive simulations are needed. To make things even more complicated, these solvers need to be tuned appropriately since they are regularly sensitive to their inner parameter settings.

The current literature is full of examples liaised to MHs. Some of the earliest ones have existed for over 30 years and are still reasonably common. Natural processes commonly inspire metaheuristics, and so, it is customary to accompany them with a corresponding metaphor. For example, some representative MHs such as Genetic Algorithms (GA) [1] and Simulated Annealing (SA) [2], are inspired by the evolution of species and the reorganisation of crystalline structures when annealing a metal, respectively. The rapid evolution of computing power throughout the last decades, along with the competent performance of MHs and their ease of use, has led to their widespread adoption. Nowadays, it is not infrequent that new MHs appear each year [3], [4]. Even so, the *No-Free-Lunch* theorem indicates that any given solver, including MHs, will be good at solving some problems but also bad at solving others [5]. However, on average, we expect all of them to have a similar performance level. This fact can be leveraged by learning for which problems a given MH excels. In doing so, one can combine several MHs into a solver that makes an appropriate selection of them, which would outperform all metaheuristics. Although running various MHs and keeping the best one seems like a straightforward idea, this approach is unfeasible in practice because of its computational cost. Plus, it does not make much sense since it cannot be migrated to other problems. However, the data gathered could serve as a useful reference point for evaluating other approaches with the focus of learning how to combine such solvers.

When dealing with MHs, one needs to determine the values of inner parameters, which can be categorical or numerical. So, performance gaps stem not only from swapping MHs, but also from changing the inner parameters of these methods. Because of this, there have been efforts at developing self-tuned approaches [6], [7]. A portion of these efforts aims at understanding the population size effect. This parameter appears in the vast majority of MHs, and it represents the number of search agents that the method uses. Mo *et al.* analysed its effect for the Firefly Algorithm (FA) [8]. They found that when tackling easy test functions, such as the Sphere, FA is not sensitive to its population size, and any value seems to yield an acceptable performance level. However, for other functions, the method is sensitive for up to 40 agents. They also found a function (Shaffer F6) where the method keeps being sensitive for a population size near the 100 agents mark. Similarly, Nawi *et al.* analysed the effect of population size for a back-propagation method built upon the Bat Algorithm [9]. Here, the authors also identified some cases where increasing the population improves the performance of the method. Conversely, they also found some scenarios where performance deteriorates as the population grows. There have been other works, which have focused on Differential Evolution [10] and GAs [11]. Others have also targeted several MHs at once [12].

The investigation was supported by the Research Group in Intelligent Systems at the Tecnológico de Monterrey (México) and by the CONACyT Basic Science Project with grant number 287479.

Even if self-tuning helps alleviate the issue of identifying appropriate MH parameters, it still represents some drawbacks. For example, it does not tackle performance variation across MHs. Some function evaluations must be sacrificed after testing on different parameter configurations, and this process is rarely used to its fullest since the MH needs to be tuned anew when changing the problem conditions. One way to circumvent these problems implies developing a high-level solver that learns when to use different MHs with different parameter settings. This approach includes the development of hyper-heuristics (HHs), which can be of different nature. Although many types of HHs exist, in this work, we will focus on selection HHs, which focus on selecting a suitable solver from the list of available ones, usually based on some features of the problem being explored. Even though HHs have been mostly employed to solve combinatorial optimisation problems, there have been some efforts at tackling continuous ones. For example, Miranda *et al.* developed a framework for learning when to use different parameter settings for the Particle Swarm Optimisation (PSO) algorithm [13]. Their data revealed that the proposed approach led to a better performance in over 80% of the experiments while having a low computational burden compared to other parameter selection approaches.

Despite the promising results found by Miranda *et al.*, their work cannot be easily generalised to other MHs. One of the reasons is that the terminology from one MH does not necessarily apply to others. For example, for searching, GA employs chromosomes while FA uses fireflies. Nevertheless, in the end, they are merely search agents embedded within a metaphor, which are modified with different procedures, say crossover and light attraction mechanics. Consequently, we recently proposed a high-level solver that leverages this fact [14]. To do so, we defined a unified MH operator-based model [15]. Using this representation, traditional MHs can be represented as one or more generic search operators (SOs). Hence, they can be decomposed into such SOs to obtain a pool of MH building blocks. With this pool, a hyper-heuristic can find a proper combination of SOs, even if it does not have a fancy name associated with it. Moreover, this allows combining operators from the same MH with different parameter settings and with unconventional topologies. Nevertheless, the initially proposed model only considered short sequences of SOs, which were repeated until solving the problem. Plus, it was restricted to a single number of search agents, to preserve consistency across operators. These considerations belong to the traditional MH structure. So, in this work, we pursue two goals. The first one is to analyse the effect of 'unfolding' the MH model, where the master strategy controlling the procedure is disregarded, and its function is delegated to a superior strategy, *i.e.,* to a hyper-heuristic. Our second goal is to study the effect that the population size of operators has on the performance of hyper-heuristics. We do this striving to estimate the eventual benefit of allowing for different population sizes within the SOs. To the best of our knowledge, this is an unexplored sub-field in the literature that can lead to great advantages in numerous applications. Hence, this manuscript has a two-fold major contribution, as it:

(i) Provides a first step towards the 'unfolding' of MHs generated through a hyper-heuristic approach and for tackling continuous optimisation problems, and
(ii) Analyses the effect of population size in the performance of the hyper-heuristic model.

This manuscript is organised as follows. Section II presents relevant information regarding key concepts of our research. Section III illustrates our proposed approach and the experimental settings. Section IV discusses the generated data. Finally, Section V wraps up this work by providing the main takeaways and by brainstorming future research paths.

II. Foundations

This section lays the groundwork for a unified terminology seeking to avoid misunderstandings.

A. Optimisation

Optimisation and nature are intertwined. In this work, we use the optimisation problem definition given by a feasible domain and an objective function to minimise, as follows.

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

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

where $\vec{x}_*$ is the optimal vector that minimises f within $\mathfrak{X}$.

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

Remark 2 (Maximisation problem)**.** *Let $\hat{f}(\vec{x})$ be the utility function to maximise such that we can state the optimisation problem with* (1) *by using $f(\vec{x}) = -\hat{f}(\vec{x})$.*

In this work, we deal with several continuous optimisation problems and one combinatorial optimisation problem at the low and high-level of our approach, respectively. For the continuous ones, we say that $\mathfrak{S} = \mathbb{R}^D$ with D as the dimensionality of the problem. For the combinatorial problem, we use $\mathfrak{S} = \mathfrak{H}^{\varpi}$, where $\mathfrak{H}$ corresponds to the heuristic space and ϖ is w.l.o.g. the heuristic sequence length. These two problem domains are employed in the next paragraphs. Further information is provided in [15].

B. Heuristics

A heuristic is a procedure that creates or alters a candidate solution for a given problem. There are many heuristic classifications in the literature. Most of them relate to combinatorial domains [16], whilst they are seldom used in continuous ones [17], [18].

In this work, we group heuristics into three categories based on [16]–[18]: *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. However, since some heuristics use a set of search individuals while others may consider a population of probes, it is convenient to regard the following two concepts:

Definition 2 (Population)**.** *Let $X(t)$ be a finite set of N candidate solutions for an optimisation problem $(\mathfrak{X}, f)$ at time t in an iterative procedure. Then, $\vec{x}_n(t) \in X(t) \subset \mathfrak{X}$ is the n-th solution or search agent of, say, the population $X(t)$.*

Definition 3 (Best solution)**.** *Let $Z(t)$ be an arbitrary set of candidate solutions. Therefore, let $\vec{x}_*(t) \in Z(t)$ be the best candidate solution from $Z(t)$,* i.e., $\vec{x}_*(t) = \operatorname{argmin}\{f(Z(t))\}$.

1) 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*, those that generate new solutions from scratch and modify current solutions, respectively [17]. We adopt these categories and add another one, as described below.

Definition 4 (Simple Heuristic)**.** *Let $\mathfrak{H}$ be a set of simple heuristics, or heuristic space, with a composition operation $\circ : \mathfrak{H} \times \mathfrak{H} \mapsto \mathfrak{H}$. Let $\mathfrak{H}_i, \mathfrak{H}_o, \mathfrak{H}_f \subset \mathfrak{H}$ be subsets of heuristics that produce a new solution, modify an existing solution, and choose between two operators, respectively.*

Remark 3 (Initialiser)**.** *$h_i : \mathfrak{S} \mapsto \mathfrak{X} \subseteq \mathfrak{S}$ generates a candidate solution within the search space $\vec{x} \in \mathfrak{X}$ from scratch*—i.e., $\vec{x} = h_i\{\mathfrak{X}\}$.

Remark 4 (Search Operator)**.** *$h_o : \mathfrak{X} \mapsto \mathfrak{X}$ modifies a position $\vec{x} \in \mathfrak{X}$,* i.e., $\vec{x}' = h_o\{\vec{x}\}$. *A search operator mostly comprises two basic operations: perturbation and selection. Let $h_p, h_s \in \mathfrak{H}_o$ be SHs that modify ($\vec{y} = h_p\{\vec{x}\}$) and update ($\vec{x} = h_s\{\vec{y}\}$) the position $\vec{x}$ called perturbators and selectors. A perturbator always precedes a selector, so $h_o = h_s \circ h_p$.*

Remark 5 (Finaliser)**.** *$h_f : \mathfrak{H}_f = \mathfrak{X} \times \mathbb{Z}_2 \mapsto \mathfrak{H}$ evaluates the current solution performance and chooses which search operator to apply by using a criteria function $c_f : (\mathfrak{X}, \mathbb{R}, \dots) \mapsto \mathbb{Z}_2$. Hence, h_f is called a finaliser and is defined as*

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

since h_j is a search operator and h_e is the identity operator.

2) Metaheuristics (MHs): A metaheuristic is a master strategy that controls SHs [4], [19]. Most of the MHs have, w.l.o.g., a scheme that Definition 5 details.

Definition 5 (Metaheuristic)**.** *Let $H : \mathfrak{S} \to \mathfrak{X}$ be an iterative procedure called metaheuristic that renders an optimal solution $\vec{x}_*$ for a given optimisation problem $(\mathfrak{X}, f)$. An MH can be defined in terms of three basic components as shown,*

$$MH_o \triangleq \langle h_i, h_o, h_f \rangle = h_f(h_o) \circ h_i, \tag{2}$$

since $h_i \in \mathfrak{H}_\text{i}$ is an initialiser, $h_f \in \mathfrak{H}_f$ is a finaliser, and $h_o \in \mathfrak{H}_o$ is a search operator.

Remark 6. *(Cardinality) The cardinality is the number of search operators that a MH implements;* i.e., $\varpi = \#h_o$.

3) Hyper-heuristics (HHs): Many researchers describe HHs as high-level heuristics controlling simple heuristics in the process of solving a problem [16]. Therefore, HHs search within the heuristic space for a configuration that solves a given problem. With that in mind, a HH can be defined according to [17] as follows.

Definition 6 (Hyper-Heuristic)**.** *Let $\boldsymbol{h} \in \mathfrak{H}^\varpi$ be a heuristic configuration, and let $Q(\boldsymbol{h}|\mathfrak{X}) : \mathfrak{H}^\varpi \times \mathfrak{X} \to \mathbb{R}$ be its performance measure function. Therefore, let a hyper-heuristic be a technique that solves*

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

III. Methods

In this section, we explain the proposed model and the experiments we carried out for validating it.

A. Proposed approach

First of all, the idea behind our approach is based on unfolding the 'traditional' metaheuristic structure to achieve a heuristic sequence, using a HH procedure. To illustrate such an idea, we use the signal-flow representation we proposed previously [15]. Figure 1 depicts how the MH scheme gives place to a heuristic sequence. Consider the feedback dashed stroke at the MH level. The finaliser controls this element and, for each step, it either returns to the search operator (SO) or concludes the procedure. For problems with a limited budget, as common in practice, the finaliser can exit the loop prematurely when an additional condition is met, *e.g.,* a fitness threshold value. Nevertheless, if the finaliser role is delegated to a superior controller (say, a hyper-heuristic), we can unfold the MH scheme into a heuristic sequence. This controller not only decides when to stop applying the search operator, but it can also choose which SO to include from a pool (the heuristic space). It is noticeable that the intermediate step in Figure 1 corresponds to a sequence of t implementations of the same SO over the population. It resembles a very trivial heuristic sequence or a guideless metaheuristic. Otherwise, the lowest level displays a more general sequence of different (non-exclusive) SOs. In these terms, we trade the (self-controlled) MH for a heuristic sequence designed by a hyper-heuristic, which seeks to enhance performance and adaptability. Therefore, we need to solve a hyper-heuristic composition problem according to Definition 6. This problem comprises a combinatorial search within the heuristic space for finding the sequence that, at a lower level, searches a continuous domain for an optimal solution.

To solve the problem mentioned above, we implemented the well-known Simulated Annealing (SA) using combinatorial search operators. In this HH implementation of SA (SAHH), the technique iteratively draws at random one action and

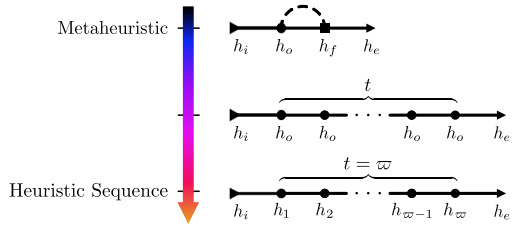

Fig. 1. Diagram of a metaheuristic and how it becomes a heuristic sequence.

applies it on the current heuristic sequence $\boldsymbol{h}$. All the implemented actions incorporate the Metropolis selector, so each couple corresponds to a SO *per se*. In simple words, SAHH renders a neighbour sequence $\boldsymbol{h}_c$ randomly, which is the typical SA implementation. In the following lines, we briefly describe the actions considered:

`Add` inserts a randomly chosen heuristic at a random l.h.s. position $i \sim \mathcal{U}\{1, \varpi+1\}$ into $\boldsymbol{h}$.

`AddMany` performs `Add` ϖ_i times since $\varpi_i \sim \mathcal{U}\{1, \varpi_{\max} - \varpi\}$, where $\varpi_{\max}$ is the maximal cardinality.

`Remove` discards a heuristic from a random position $i \sim \mathcal{U}\{1, \varpi\}$ of $\boldsymbol{h}$.

`RemoveMany` performs `Remove` ϖ_j times since $\varpi_j \sim \mathcal{U}\{1, \varpi - \varpi_{\min}\}$, where $\varpi_{\min}$ is the minimal cardinality.

`Shift` selects a random position $i \sim \mathcal{U}\{1, \varpi\}$ from $\boldsymbol{h}_c$ and changes the corresponding heuristic $h_i \in \boldsymbol{h}$ with another chosen at random.

`LocalShift` is similar to `Shift`, but the new heuristic is selected around the original one, *i.e.,* one step back or forward with cyclic indexing.

`Swap` interchanges heuristics $h_i, h_j \in \boldsymbol{h}$ at two randomly selected locations, $i, j \sim \mathcal{U}\{1, \varpi\}$.

`Restart` ignores the current sequence and randomly selects a new one with the same cardinality ϖ.

`Mirror` reorganises the sequence $\boldsymbol{h}$ in reverse order.

`Roll` moves all the heuristics in $\boldsymbol{h}$ one position back or forward at random and with cyclic indexing.

`RollMany` is similar to `Roll` but $k \geq 1$ positions randomly chosen to displace, *i.e.,* $k \sim \mathcal{U}\{1, \varpi - 1\}$.

Complementing the way of searching the heuristic space, we assess the performance for a candidate solution $\boldsymbol{h}$ such as

$$\text{perf}(X_*) = -(\text{med} + \text{iqr})\left(\{\forall\ \vec{x}_{r,*} \in X_* | f(\vec{x}_{r,*})\}\right), \quad (4)$$

where med and iqr are the median and interquartile range operators applied to the fitness values $f(\vec{x}_{r,*})$. For using (4), we run each candidate 50 times and record all fitness values. Regard the minus sign that indicates the lower the fitness (med and iqr) values, the greater the performance. Furthermore, for the remaining SA parameters, we use a maximal of 200 steps, an initial dimensionless temperature of 1.0, a minimal dimensionless temperature of 10^{-6}, and a cooling rate of 10^{-3}. Lastly, and for the sake of clarity, we summarise the procedures performed for each problem level in Pseudocode 1.

Subsequently, we outline the population-based heuristics for continuous domains implemented as the heuristic space. We avoid the full description of these heuristics for the sake of brevity, and also because they were extracted and

Pseudocode 1 Hyper-heuristic based on Simulated Annealing

Input: Domain $\mathfrak{X}$, objective function $f(\vec{x})$, heuristic space $\mathfrak{H}_o$, initialiser h_i, performance metric perf(X_*), population size N, and action set A. Additional parameters: Θ_0, $\Theta_{\min}$, δ, and $\varpi_{\max}$.
Output: Best unfolded metaheuristic $\boldsymbol{h}_*$

$\boldsymbol{h} \leftarrow$ ChooseRandomly($\mathfrak{H}_o$, $P_{\mathfrak{H}}(h)$)
for $r = \{1, \ldots, N_r\}$ **do** $X_* \ni \vec{x}_{r,*} \leftarrow$ EvaluateSeq($\boldsymbol{h}_c$) **end for**
$Q(\boldsymbol{h}|\mathfrak{X}) \approx$ perf(X_*) with Eq. (4), $\boldsymbol{h}_* \leftarrow \boldsymbol{h}$, $s \leftarrow 0$, and $\Theta \leftarrow \Theta_0$
while $(\Theta > \Theta_{\min})$ and $(s \leq s_{\max})$ **do**
  $a \leftarrow$ ChooseAction(A, a, ϖ), and $\boldsymbol{h}_c \leftarrow a\{\boldsymbol{h}\}$
  for $r = \{1, \ldots, N_r\}$ **do** $X_* \ni \vec{x}_{r,*} \leftarrow$ EvaluateSeq($\boldsymbol{h}_c$) **end for**
  $Q(\boldsymbol{h}_c|\mathfrak{X}) \approx$ perf(X_*) with Eq. (4)
  if $\mathcal{U}(0,1) \leq \exp(-(Q(\boldsymbol{h}_c|\mathfrak{X}) - Q(\boldsymbol{h}|\mathfrak{X}))/\Theta)$ **then** $\boldsymbol{h} \leftarrow \boldsymbol{h}_c$, **end if**
  if $Q(\boldsymbol{h}_c|\mathfrak{X}) < Q(\boldsymbol{h}_*|\mathfrak{X})$ **then** $\boldsymbol{h}_* \leftarrow \boldsymbol{h}$ and $s \leftarrow 0$,
  else $s \leftarrow s + 1$, **end if**
  $\Theta \leftarrow \Theta(1 - \delta)$
end while

procedure EvaluateSeq($\boldsymbol{h}$)
  $X(t) \ni \vec{x}_n(t) \leftarrow h_i\{\mathfrak{X}\}, \forall\ n \in \{1, \ldots, N\}$
  $F(t) \ni f_n(t) \leftarrow f(\vec{x}_n(t)),\ \forall\ n \in \{1, \ldots, N\}$
  $\vec{x}_*(t) \leftarrow \vec{x}_k(t)$ since $k = \text{arginf}\{F(t)\}$
  for $m = \{1, \ldots, \varpi\}$ **do**
    $X(t), F(t) \leftarrow h_m\{X(t)\},\ \forall\ h_m \in \boldsymbol{h}$
    $\vec{x}_*(t) \leftarrow h_s\{X(t)\},\ h_s \in \mathfrak{H}_s$
  end for
  return $\vec{x}_*(t)$
end procedure

documented in prior works [14], [15], [20]. Bearing this in mind, Tables I and II present the perturbators and selectors employed in the following experiments and according to Definition 4. These simple heuristics were obtained from the following ten well-known MHs: Random Search (RS), Simulated Annealing (SA), Genetic Algorithm (GA), Cuckoo Search (CS), Differential Evolution (DE), Particle Swarm Optimisation (PSO), Firefly Algorithm (FA), Stochastic Spiral Optimisation Algorithm (SSOA), Central Force Optimisation (CFO), and Gravitational Search Algorithm (GSA).

To conclude the description of our proposal, we analyse its time complexity. In the high-level, SAHH has a complexity similar to that of a regular MH, such as

$$T_{\text{HH}} = \mathcal{O}\left(N_r * T_{\text{MH}} * s_{\max}\right), \quad (5)$$

since N_r is the number of repetitions, T_{MH} is the cost of implementing a candidate sequence, and $s_{\max}$ stands for the steps carried out in this search. Plus, SAHH is a single-agent algorithm, so selection, perturbation, and acceptance are actions considered to have constant time.

In the low-level, the time complexity for an MH composed of $\varpi_{\max}$ search operators and performing $t_{\max}$ iterations yields

$$T_{\text{MH}} = \mathcal{O}\left(t_{\max} * \varpi_{\max} * N * D * T_p\right), \quad (6)$$

where N is the population size, D is the dimensionality, and T_p is the computational cost of evaluating the perturbator; which represents the worst case-scenario compared to the time required by the selector and finaliser. It is noteworthy that the worst-case value of T_p coincides with the Cost-based Roulette Wheel Pairing $\mathcal{O}(N^2)$ [30]. Besides, we focus on analysing the procedure and not the problem to solve, so we can adopt w.l.o.g that the time complexity of evaluating the fitness is

TABLE I
POPULATION-BASED PERTURBATORS EXTRACTED FROM 10 WELL-KNOWN METAHEURISTICS IN THE LITERATURE

Name, Reference	Expressions[a]	Parameters[b]
Random Sample, [21]	$\vec{y}_n = \vec{r}$	$\vec{r} \ni r_i \sim \mathcal{U}(-1, 1)$
Random Walk, [2]	$\vec{y}_n = \vec{x}_n + \alpha\vec{r}$	$\alpha \in]0, 1]$, $\vec{r} \ni r_i \sim \mathcal{U}(-1, 1)$
Local Random Walk, [22]	$\vec{y}_n = \vec{x}_n + \alpha\vec{r} \odot H(\vec{r} - p) \odot (\vec{x}_{z_1} - \vec{x}_{z_2})$	α,$p \in]0, 1]$,$\vec{r} \ni r_i \sim \mathcal{U}(0,1)$,$z_1, z_2 \sim \mathcal{U}\{1, N\}$,$z_1 \neq z_2$
Random Flight, [22]	$\vec{y}_n = \vec{x}_n + \alpha\vec{r} \odot (\vec{x}_n - \vec{x}_*)$	$\alpha \in]0, 1]$, $\vec{r} \ni r_i \sim \mathcal{L}(\beta)$, $\beta \in [1.25, 1.75]$
Genetic Crossover,[c] [23]	$\vec{y}_n = \vec{m} \odot \hat{\vec{x}}_{z_1} + (1 - \vec{m}) \odot \hat{\vec{x}}_{z_2}$	$z_1, z_2 \in \{1, \ldots, M\}$, $n \in \{M + 1, \ldots, N\}$
Genetic Mutation, [23]	$\vec{y}_n = \vec{m} \odot \vec{x}_n + \alpha(1 - \vec{m}) \odot \vec{r}$, $\vec{m} = H(p_m - \vec{q})$	$\alpha \in]0, 1]$, $\vec{r} \ni r_i \sim \mathcal{U}(-1, 1)$, $\vec{q} \ni q_i \sim \mathcal{U}(0, 1)$, $p_e, p_m \in]0, 1]$, $\forall\, n \in \{\lceil p_e N \rceil, \ldots, N\}$
Differential Mutation,[d] [24]	$\vec{y}_n = \vec{x}_{d_1} + \alpha_0(\vec{x}_{d_2} - \vec{x}_{d_3}) + \alpha_m \sum_{m=1}^{M} \left(\vec{x}_{z_{2m-1}} - \vec{x}_{z_{2m}}\right)$	$\forall \alpha_m \in]0, 3]$,$M \in \mathbb{Z}_+$,$\forall z_i \sim \mathcal{U}\{1, N\} : \bigcap_j z_j = \emptyset$
Spiral Dynamic,[e] [25]	$\vec{y}_n = \vec{x}_* + \vec{r}\mathbf{R}_D(\theta) \cdot (\vec{x}_n - \vec{x}_*)$	$\vec{r} \ni r_i \sim \mathcal{U}(r_l, r_u)$, $r_l, r_u \in]0, 1[$, $\theta \in]0, 2\pi[$
Swarm Dynamic,[f] [26]	$\vec{y}_n = \vec{x}_n + \vec{v}_n(t)$, $\vec{v}_n(t) = \alpha_0\vec{v}_n(t-1) + \alpha_1\vec{r}_1 \odot (\vec{x}_{n,*} - \vec{x}_n) + \alpha_2\vec{r}_2 \odot (\vec{x}_* - \vec{x}_n)$	$\alpha_0 \in]0, 1[$, $\alpha_1, \alpha_2 \in]0, 4]$, $\vec{r}_i \ni r_j \sim \mathcal{U}(0, 1), \forall\, i$
Firefly Dynamic, [27]	$\vec{y}_n = \vec{x}_n + \alpha_0\vec{r} + \alpha_1 \sum_{k=1,k\neq n}^{N} H(-\Delta I_{n,k}) \Delta\vec{x}_{n,k} e^{-\alpha_2 \lvert\lvert \Delta\vec{x}_{n,k} \rvert\rvert_2^2}$ $\Delta I_{n,k} = f(\vec{x}_k) - f(\vec{x}_n)$, $\Delta\vec{x}_{n,k} = \vec{x}_k - \vec{x}_n$	$\alpha_0, \alpha_1 \in]0, 1]$, $\alpha_2 \in]0, 10^3]$, $\vec{r} \ni r_i \sim \mathcal{U}(-\frac{1}{2}, \frac{1}{2})$
Central Force Dynamic, [28]	$\vec{y}_n = \vec{x}_n(t) + \frac{1}{2}\vec{a}_n\Delta t^2$, $\vec{a}_n = \alpha_0 \sum_{k=1,k\neq n}^{N} \frac{m_{n,k}\Delta\vec{x}_{n,k}}{\lvert\lvert \Delta\vec{x}_{n,k} \rvert\rvert_2^{\alpha_2} + \varepsilon}$, $m_{n,k} = H(\Delta M_{n,k})(\Delta M_{n,k})^{\alpha_1}$, $\Delta M_{n,k} = f(\vec{x}_k) - f(\vec{x}_n)$, $\Delta\vec{x}_{n,k} = \vec{x}_k - \vec{x}_n$	$\alpha_0, \alpha_1 \in]0, 1]$, $\alpha_2 \in]1, 2[$, $\Delta t = 1$, $\varepsilon = 10^{-23}$
Gravitational Search, [29]	$\vec{y}_n = \vec{x}_n + \vec{v}_n(t)$, $\vec{v}_n(t) = \vec{r} \odot \vec{v}_n(t-1) + \vec{a}_n(t)$ $\vec{a}_n(t) = \alpha_0 e^{-\alpha_1 t} \sum_{k=1,k\neq n}^{N} \frac{f(\vec{x}_\circ(t)) - f(\vec{x}_n)}{N f(\vec{x}_\circ) - \sum_{k=1}^{N} f(\vec{x}_k)} \vec{r}_k \odot \frac{\Delta\vec{x}_{n,k}}{\lvert\lvert \Delta\vec{x}_{n,k} \rvert\rvert_2 + \varepsilon}$	$\vec{r} \ni r_i \sim \mathcal{U}(0, 1)$, $\alpha_0 \in]0, 1]$, $\alpha_1 \in]0, 0.1]$, $\varepsilon = 10^{-23}$

[a] $\vec{x}_n$ is the vector position of the n-th agent; $\vec{x}_*$ and $\vec{x}_\circ$ correspond to the best and worst positions; $\odot$ is the Hadamard-Schur product and H is the Heaviside function. [b] $\vec{r}$ is a vector of i.i.d. random variables with either Uniform, Normal or Lévy stable distribution. [c] $\hat{\vec{x}}_{z_1}$ and $\hat{\vec{x}}_{z_2}$ are the parents from a mating pool using a pairing scheme, and $\vec{m}$ is the mask vector determined via a crossover mechanism. [d] $\vec{x}_{d_1}, \vec{x}_{d_2}, \vec{x}_{d_3}$ are reference vectors selected according to the differential mutation mechanism. [e] $\mathbf{R}_D(\theta)$ is the rotation matrix; r_l, r_u are the lower and upper radius threshold. [f] $\alpha_0, \alpha_1, \alpha_2$ are coefficients picked up according to the kind of swarm dynamic.

TABLE II
SELECTORS EMPLOYED IN THIS WORK

Name	Expressions
Direct	$\vec{x}_n(t+1) = \vec{y}_n$
Greedy	$\vec{x}_n(t+1) = \begin{cases} \vec{y}_n, & \text{if } \Delta f_n \leq 0, \\ \vec{x}_n(t), & \text{otherwise.} \end{cases}$
Probabilistic	$\vec{x}_n(t+1) = \begin{cases} \vec{y}_n, & \text{if } (\Delta f_n \leq 0) \vee (p_s < r), \\ \vec{x}_n(t), & \text{otherwise.} \end{cases}$ $p_s =]0, 1[$, $r \sim \mathcal{U}(0, 1)$
Metropolis	$\vec{x}_n(t+1) = \begin{cases} \vec{y}_n, & \text{if } (\Delta f_n \leq 0) \vee (r < e^{-\frac{\Delta f_n(t)}{k_B \Theta(t)}}), \\ \vec{x}_n(t), & \text{otherwise,} \end{cases}$ $r \sim \mathcal{U}(0, 1)$, $k_B \in \mathbb{R}_+$, $\Theta(t) : \mathbb{Z}_+ \to \mathbb{R}_+$

constant. Moreover, recall that for the unfolded metaheuristics (heuristic sequences) $t_{\max} = \varpi_{\max}$.

Plugging (6) in (5), we can express the time complexity of the overall approach T_{HH}, as follows,

$$T_{\text{HH}} = \mathcal{O}\left(s_{\max} * N_r * \varpi_{\max} * D * N^3\right). \tag{7}$$

From the above analysis, it is exciting to surmise that the running time scales linearly with the problem dimensionality and cubically with the population size; which is somehow manageable. The chief reason is the high-level metaheuristic (*i.e.,* SAHH) controlling the process, which performs one modification per step in a single element of the unfolded metaheuristic. For the scope of this research, SAHH constitutes a good prospect with a low computational burden.

B. Experimental description

The objective of this work is to study the influence of the problem dimensionality when designing unfolded metaheuristics via a hyper-heuristic approach. On the one hand, it is essential to remark that this unfolding procedure has not been explored yet in the literature. Thus, we employed the framework CUSTOMHyS v1.0 to test this idea with ease (freely available at https://github.com/jcrvz/customhys). CUSTOMHyS facilitates the implementation of MHs through population-based search operators as building blocks. Particularly, we used the default collection of 205 SOs achieved by varying parameters, distributions, and schemes, as described in Tables I and II. Also, we set the maximal number of iterations $t_{\max}$ to one, since the heuristic sequence will be a 'unfolded' metaheuristic, relaxed the maximal cardinality $\varpi_{\max} \in [1, 100]$, and included the SAHH procedure (including the actions) as explained in Section III-A. Further details about the framework are provided in [14].

On the other hand, we carried out the offline experiments over a benchmark function collection of 107 problems, which also belongs to the CUSTOMHyS framework. We chose four dimensionality values for these problem domains, such as 2, 10, 30, and 50. Plus, we categorised them by using qualitative characteristics such as Differentiability and Unimodality in four binary-encoded duads, *i.e.,* DU. To study the effect of the population size, the experiments were run using populations of 30, 50, and 100 agents for each hyper-heuristic procedure.

All the experiments were run on a Dell Inc. PowerEdge R840 Rack Server with 16 Intel Xeon Gold 5122 CPUs @ 3.6 GHz, 125 GB RAM, and CentOS Linux 7.6.1810-64 bit. All the resulting data are also freely available at https://github.com/jcrvz/unfolded-metaheuristics-preliminary.

IV. RESULTS

Once the experiments were performed, as we described in the previous section, we plotted the results in different inferential manners to find relevant insights. Remember that we aim to prove the automated design of unfolded metaheuristics via a hyper-heuristic approach to solve continuous optimisation problems. We are also interested in observing the effect of the population size during the implementation. For starters, Figure 2 presents an overview of the data, where we rank the setups according to the performance achieved when solving each problem with a given dimensionality. Consider that the

possible values for these rankings are 1, 2, and 3. Heuristic sequences with larger populations generally represent an excellent option for any scenario. This is quite expected, although there are some problem domains where sequences achieved using 100 agents are not the best option, particularly for lower dimensionalities. Even so, the remaining population sizes exhibit similar distributions with some slight differences. Hence, they do not draw any particular conclusions at this moment. Notice that, ideally, violins for the populations of 30, 50, and 100 agents should be unimodal and centred around 3, 2, and 1, respectively. Naturally, this is not the case. Such an outcome can be explained through the 'hardness' of problems, which depends on dimensionality, but it can also be associated with other inherent factors. So, increasing the population size can contribute to dealing with such issues. In our results, it seems somewhat straightforward because each experiment setup corresponds to a different unfolded metaheuristic.

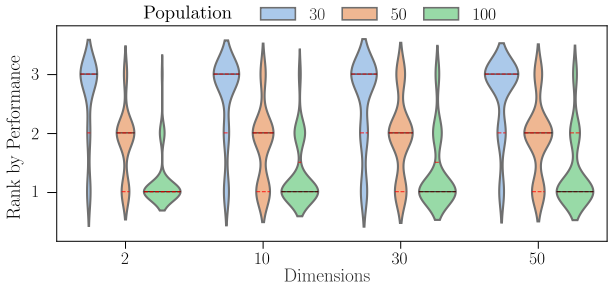

Fig. 2. Violin plots of rank values for the heuristic sequences achieved varying the population size and solving problems with different dimensionalities.

Seeking to detail these inherited characteristics, Figure 3 discriminates the rank distributions based on their categorical features. As we commented in the previous section, we chose the categorical (Cat) features corresponding to Differentiability (first digit) and Unimodality (second digit). Thus, Figure 3 provides the binary encoded categories of 11, 10, 01, and 00, from top to bottom. So, one would expect that the distribution of each subplot resembles three vertical stripes, as the ranking should be closely related to population size. However, such a pattern is barely observed. Instead, it resembles more a descending staircase for many of the scenarios. Keep in mind that, for our experiments, the population size is a design condition, rather than a hyper-parameter. Hence, the HH procedure must deal with that when searching within the heuristic space.

For the 'easiest' problem set, *i.e.,* Cat = 11 and 2D, it is common for sequences achieved with a population of 100 agents to be in the first place. However, the other two setups also rank at this position and do so in over 25% of the tests. This can be explained by the fact that many methods can reach excellent results (even optimal fitness values) for such kind of problems. Even so, this effect is also observable for other problem categories with low dimensionalities. Moreover, there is a somewhat systematic tendency in the ranking distributions for problems belonging to the extreme categories, *i.e.,* 11 and 00. Nevertheless, this pattern is highly disrupted in the Non-differentiable and Unimodal category (Cat = 01) when the dimensions are greater than 10, *i.e.,* at 30D and 50D. In this case, we must remark that the proposed approach deals with the inherent 'hardness' of these problems, which is somehow amplified by the dimensionality. For example, consider problems in 50D where heuristic sequences achieved with a population of 30 agents seem to never rank first. This may be explained by the fact that some SOs perform well with reduced dimensionalities (*e.g.,* PSO- and SSOA-based), whereas others do so with larger values (*e.g.,* DE- and GA-based). Therefore, throughout its search, SAHH found suitable combinations of 30-population operators, but which were not as good as when using the other population sizes. Further details about the resulting unfolded metaheuristics can be found in https://github.com/jcrvz/unfolded-metaheuristics-preliminary.

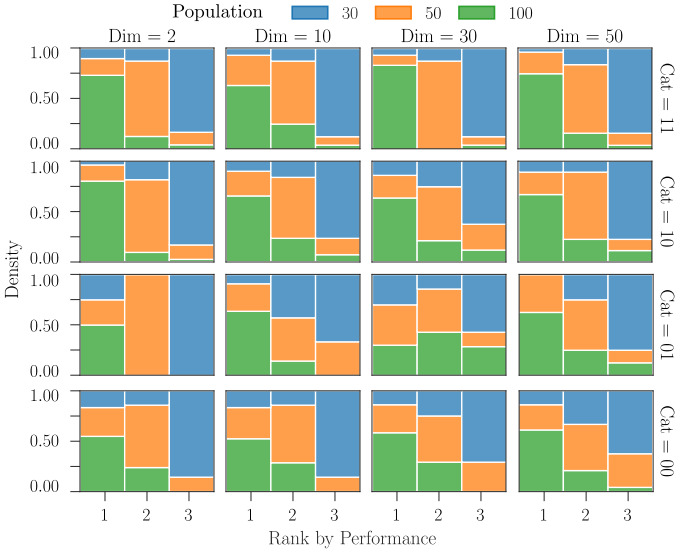

Fig. 3. Distribution of the ranking positions for the heuristic configurations achieved varying the population sizes, and solving problem domains with different dimensionalities (Dim) and characteristics (Cat).

Figures 4 and 5 display the distribution of cardinality and steps obtained for each implementation setup, respectively. Recall that the cardinality refers to the number of search operators in the sequence and the steps are the number of iterations performed by SAHH. Concerning the cardinality (see Figure 4), we notice a great concentration of values near the upper limit of 100. This means that most of the heuristic configurations required a nearby number of SOs. In general, the height of this peak slightly increases with the number of dimensions, which indicates that the proposed approach deals with the dimensionality 'curse' by including more varied iterations (*i.e.,* using different operators). In some cases, such as for two dimensions, there also are some small peaks between 40–60, and an even smaller one near 1 (*e.g.,* a short heuristic sequence with a population of 100 agents). This means that the SAHH found unfolded metaheuristics with cardinalities below 100 that are good at solving certain problems. This scenario becomes more infrequent as dimensionality increases. Even so, we do not detect any notorious effect of population size over the cardinality of generated sequences.

In the case of the steps, Figure 5 shows that SAHH required

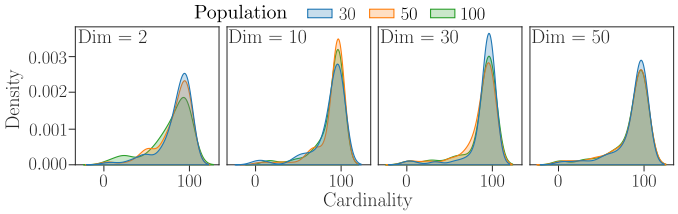

Fig. 4. Cardinality distribution for heuristic sequences achieved varying population and solving problem domains with different dimensions (Dim).

fewer ones (around 50) to find heuristic sequences in most of the 2D problems. Certainly, there are some cases, *e.g.,* harder problems, that needed higher values, as indicated by the lobes near the 200 steps mark. Actually, distributions for higher dimensionalities appear to mirror those for two dimensions.

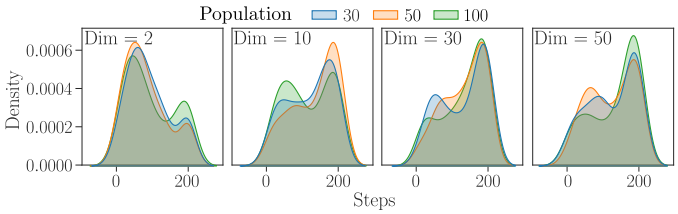

Fig. 5. Steps distribution for heuristic sequences achieved varying population and solving problem domains with different dimensions (Dim).

In the last step of our analysis, we study the order of the numerical operations that each implementation performs when solving a given problem. This estimation is based on the time complexity analysis from Section III-A, but using data gathered throughout the experiments. Figure 6 displays this information as boxplots, where the effect of the population size and the problem dimensionality is evident. It is noticeable that the increment is not too dramatic, as we foresaw with the theoretical analysis. This is due to the low computational burden that SA imposes on the HH approach. Even so, bear in mind that this cost could become excessive with other strategies and it would be interesting to investigate this more deeply. It is also worth mentioning that many of the outliers in Figure 6 appear when solving differentiable and multimodal problems. This is an interesting combination of characteristics, since problems from this category exhibit diverse behaviours. Such a fact must be a matter to be considered in further works.

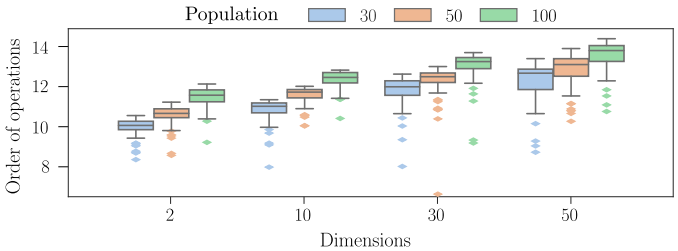

Fig. 6. Order of the numerical operations performed varying the population size and solving problem domains with different dimensionalities.

Table III illustrates the resulting performance values for one selected problem per category. Recall that this performance metric comprises the median value and the interquartile range of the sampled fitness values. The first problem is the Sphere function, which is somehow classic. But here we consider larger domain constraints than usual, *i.e.,* $[-100, 100]^D$. This remark is the keystone for understanding why performance scales so badly with dimensionality, despite it being one of the simplest problems from the literature. For this problem, the 100-population setup seems to be the most suitable one. The performance values quickly rise with the dimensionality, but they appear to be mitigated when specifying a larger population size. Similar behaviour but in sober proportions is observed in the Rastrigin function, which belongs to differentiable and multimodal functions. Notice that in the 30D case, a population of 50 is enough for finding an excellent performance. Subsequently, the Step function presents some curious results where setups with a population of 30 agents render the best solution for the 30-dimensional problem domain. A somewhat related scenario is regarded with the Zero Sum problem, particularly when dimensions are 30. With this example, we show how diverse the outcomes of the proposed methodology can become. This relates to the imposed problem conditions (population size, problem domain, heuristic space, and cardinality) and our proposed approach seeks to deal with them by designing a suitable unfolded metaheuristic.

TABLE III
PERFORMANCE VALUES FOR SELECTED PROBLEMS, WITH DIFFERENT DIMENSIONALITY (DIM), FROM EACH CATEGORY (CAT) SOLVED EMPLOYING THE PROPOSED APPROACH VARYING THE POPULATION SIZE.

Problem, Cat	Dim	Performance by population size 30	50	100
Sphere, 11	2	3.4×10^{-5}	2.8×10^{-6}	$\mathbf{6.1\times10^{-8}}$
	10	6.2	2.4	**0.9**
	30	3894.5	2477.6	**1390.3**
	50	12380.2	9074.7	**1877.1**
Rastrigin, 10	2	8.0×10^{-4}	2.5×10^{-7}	$\mathbf{2.3\times10^{-8}}$
	10	40.7	17.4	**3.6**
	30	197.0	**175.0**	216.3
	50	463.8	382.4	**265.1**
Step, 01	2	**0.0**	**0.0**	**0.0**
	10	7.0	1.0	**0.0**
	30	**116.0**	178.0	119.0
	50	667.8	**239.0**	354.8
Zero Sum, 00	2	**0.0**	**0.0**	**0.0**
	10	**0.0**	**0.0**	**0.0**
	30	7.7	**1.7**	1.9
	50	7.4	8.5	**3.0**

V. CONCLUSIONS

This work proposed a novel and simple methodology for designing heuristic-based procedures to solve continuous optimisation problems. The technique comprises the well-known Simulated Annealing algorithm, which searches within the heuristic space for a sequence that, in a lower level, explores the continuous problem domain for the optimal solution. In general terms, our approach is a wide-sense heuristic procedure for tailoring methods, so once a problem is analysed, the achieved unfolded metaheuristic is like a recipe to follow step-by-step. In this methodology, the unfolded MHs abandon their self-controlling mechanisms, allowing a high-level heuristic to decide which SOs to use and in which order. There is also room for enhancing these unfolded metaheuristics, but a little of expertise is required. Besides, we also investigated the effect of varying the population size when implementing the approach for different problem domains.

Our results proved that the proposed approach is a reliable alternative for tackling optimisation problems. This is achieved by trading the conventional structure of MHs, which sometimes limit the search to the repetitive application of one or two SOs, by a more diverse process. Moreover, we observed that using a population size is just a design condition in the hyper-heuristic problem. The proposed approach has shown to deal with this additional condition effectively. Naturally, a larger population size benefits the cardinality and steps of the resulting sequence, as long as the dimensionality of the problem remains low. This also has an important impact in the number of operations (related to the time complexity) performed by the approach. Nevertheless, we showed that the computational burden of our proposal is not excessively affected when using larger populations. We want to mention that using other population sizes do not imply that our approach finds worse solutions. Instead, it is directed towards a suitable sequence for the conditions given. In the end, this is one of the critical features of (meta-)heuristics: flexibility.

Due to the notable potential that stems from our approach, we believe that this work is only the *hors d'oeuvre* for several subsequent studies and applications. The most straightforward one consists on extending the experimental setup to other kinds of problems, such as those from the CEC and GECCO competitions. In the same alley, we plan to include real-world applications problems, *e.g.,* microelectronic thermal management and photovoltaic arrangement design problems. Besides, we found that the categorical features used to characterise the problems may not be meaningful enough for identifying patterns from the obtained results. So, we expect to implement other types of features, particularly those based on landscape analysis, to guide the HH procedure. Moreover, we also plan to focus on analysing the effect of other hyper-parameters and features in hyper-heuristic and metaheuristic levels over the overall performance. Last but not least, and because of its potential as a low-cost methodology, we plan to exploit our approach by fully characterising the SOs to include a constraint related to the allowed computing budget.

REFERENCES

[1] D. Goldberg and J. Holland, "Genetic algorithms and machine learning," *Machine learning*, vol. 3, no. 2-3, pp. 95–99, 1988.

[2] S. Kirkpatrick, C. D. Gelatt Jr., and M. P. Vecchi, "Optimization by Simulated Annealing Optimization by Simulated Annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.

[3] N. Khanduja and B. Bhushan, "Recent advances and application of metaheuristic algorithms: A survey (2014–2020)," *Metaheuristic and Evolutionary Computation: Algorithms and Applications*, pp. 207–228, 2021.

[4] K. Hussain, M. N. M. Salleh, S. Cheng, and Y. Shi, "Metaheuristic research: a comprehensive survey," *Artificial Intelligence Review*, vol. 52, no. 4, pp. 2191–2233, 2019.

[5] S. P. Adam, S.-A. N. Alexandropoulos, P. M. Pardalos, and M. N. Vrahatis, "No Free Lunch Theorem: A Review," in *Approximation and Optimization* (I. Demetriou and P. Pardalos, eds.), pp. 57–82, Springer, Cham, 2019.

[6] R. Durgut and M. E. Aydin, "Adaptive binary artificial bee colony algorithm," *Applied Soft Computing*, vol. 101, p. 107054, 2021.

[7] C. E. d. S. Santos, R. C. Sampaio, L. d. S. Coelho, G. A. Bestarsd, and C. H. Llanos, "Multi-objective adaptive differential evolution for SVM/SVR hyperparameters selection," *Pattern Recognition*, vol. 110, p. 107649, 2021.

[8] Y. B. Mo, Y. Z. Ma, and Q. Y. Zheng, "Optimal choice of parameters for firefly algorithm," in *4th International Conference on Digital Manufacturing and Automation*, pp. 887–892, 2013.

[9] N. M. Nawi, M. Z. Rehman, and A. Khan, "The effect of bat population in Bat-Bp algorithm," *Lecture Notes in Electrical Engineering*, vol. 291 LNEE, pp. 295–302, 2014.

[10] S. Chen, J. Montgomery, and A. Bolufé-Röhler, "Measuring the curse of dimensionality and its effects on particle swarm optimization and differential evolution," *Applied Intelligence*, vol. 42, no. 3, pp. 514–526, 2015.

[11] K. Kalaiselvi and A. Kumar, "An empirical study on effect of variations in the population size and generations of genetic algorithms in cryptography," in *IEEE International Conference on Current Trends in Advanced Computing*, vol. 2018-January, pp. 1–5, 2017.

[12] Q. Li, S. Y. Liu, and X. S. Yang, "Influence of initialization on the performance of metaheuristic optimizers," *Applied Soft Computing Journal*, vol. 91, p. 106193, 2020.

[13] P. B. Miranda, R. B. Prudêncio, and G. L. Pappa, "H3ad: A hybrid hyper-heuristic for algorithm design," *Information Sciences*, vol. 414, pp. 340–354, 2017.

[14] J. M. Cruz-Duarte, I. Amaya, J. C. Ortiz-Bayliss, H. Terashima-Marín, and Y. Shi, "CUSTOMHyS: Customising Optimisation Metaheuristics via Hyper-heuristic Search," *SoftwareX*, vol. 12, p. 100628, 2020.

[15] J. M. Cruz-Duarte, J. C. Ortiz-Bayliss, I. Amaya, Y. Shi, H. Terashima-Marín, and N. Pillay, "Towards a Generalised Metaheuristic Model for Continuous Optimisation Problems," *Mathematics*, vol. 8, no. 11, p. 2046, 2020.

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

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

[18] J. H. Drake, A. Kheiri, E. Özcan, and E. K. Burke, "Recent advances in selection hyper-heuristics," *European Journal of Operational Research*, vol. 285, pp. 405–428, 9 2020.

[19] T. Dokeroglu, E. Sevinc, T. Kucukyilmaz, and A. Cosar, "A survey on new generation metaheuristic algorithms," *Computers & Industrial Engineering*, vol. 137, p. 106040, 2019.

[20] J. M. Cruz-Duarte, I. Amaya, J. C. Ortiz-Bayliss, S. E. Conant-Pablos, and H. Terashima-Marín, "A primary study on hyper-heuristics to customise metaheuristics for continuous optimisation," in *IEEE Congress on Evolutionary Computation*, pp. 1–8, IEEE, 2020.

[21] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *Journal of machine learning research*, vol. 13, no. Feb, pp. 281–305, 2012.

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

[23] C. W. Ahn, *Practical genetic algorithms*, vol. 18. Wiley, 2006.

[24] S. Das, S. S. Mullick, and P. N. Suganthan, "Recent advances in differential evolution-An updated survey," *Swarm and Evolutionary Computation*, vol. 27, pp. 1–30, 2016.

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

[26] M. Clerc and J. Kennedy, "The particle swarm-explosion, stability, and convergence in a multidimensional complex space," *Evolutionary Computation, IEEE Transactions on*, vol. 6, no. 1, pp. 58–73, 2002.

[27] X.-S. Yang *et al.*, "Firefly algorithm," *Nature-inspired metaheuristic algorithms*, vol. 20, pp. 79–90, 2008.

[28] R. A. Formato, "Central force optimization: A new deterministic gradient-like optimization metaheuristic," *Opsearch*, vol. 46, no. 1, pp. 25–51, 2009.

[29] A. Biswas, K. K. Mishra, S. Tiwari, and A. K. Misra, "Physics-Inspired Optimization Algorithms: A Survey," *Journal of Optimization*, vol. 2013, pp. 1–16, 2013.

[30] D. E. Goldberg and K. Deb, "A comparative analysis of selection schemes used in genetic algorithms," in *Foundations of genetic algorithms*, vol. 1, pp. 69–93, Elsevier, 1991.