# A Transfer Learning Hyper-heuristic Approach for Automatic Tailoring of Unfolded Population-based Metaheuristics

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*—It is no secret that optimisation is a popular topic in any practical engineering application. Similarly, Metaheuristics (MHs) are a fairly standard approach for solving optimisation problems due to their success, flexibility, and simplicity. However, it is seldom easy to find a solver from the overpopulation of metaheuristics that adequately deals with a given problem. For that reason, the solver selection is even considered an additional problem in many optimisation scenarios. This work investigates the Metaheuristic Composition Optimisation Problem, which involves designing heuristic-based procedures that solve continuous optimisation problems. Therefore, we propose two novel and still simple methodologies based on transfer learning to facilitate the automatic generation of population-based and metaphor-less MHs by using search operators from the literature. To represent these solvers, we adopt our previously proposed unfolded MH model. The first strategy deals with the problem dynamically, building the sequence while solving the low-level problem. In contrast, the second one does it statically by generating the whole candidate sequence before implementing it. Results provide us with information to prove the feasibility of these approaches via experiments using 32 problems with four different characteristic groups and four dimensionalities and varying the number of agents (30, 50, and 100) employed by the search operators. We also remark that one can compare these two methodologies on performance, but we emphasise their potential usage depending on the general application environment.**

***Index Terms*—Hyper-heuristic, Transfer Learning, Search Operators, Optimisation, Evolutionary Computation.**

## I. Introduction

TECHNOLOGICAL advancements provide us comfort and increase our quality of life. Nonetheless, it has a cost to pay: new optimisation problems emerge. In general, optimisation is an unquestionable field that has permeated in all human activities. This fact can be corroborated without much effort by consulting any electronic library where one can find astonishing approaches and applications concerning optimisation problems. In particular, metaheuristics (MHs) are well-known solvers with a customary use nowadays because of their proven success, flexibility, and simplicity. Nevertheless, MHs exhibit some drawbacks. There is a colourful palette of metaheuristics where many claim to be the best for engineering applications [1]–[3]. However, they are not so general or the definitive solver: the *No-Free-Lunch* theorem reigns [4]. So, there is an additional problem to face when practitioners deal with an optimisation scenario: how to properly select an MH for a given problem. We must also know how to set the tuning parameters and learn which ranges allow for the desired behaviour. In other words, a little expertise is required for using these methods effectively. Therefore, the question of deciding which metaheuristic is worth implementing to tackle a given problem remains open.

One way to surmount this concern is to develop a high-level solver that explores how to use different reconfigured MHs with a certain degree of intelligence. These solvers are the Hyper-Heuristics (HHs), which can be different and have a relatively high growing popularity [5], [6]. In the last decade, one can find many applications of HHs in combinatorial [7] and continuous [8] optimisation. Although, it is more like a treasure trove challenge when one is interested in the latter domain [9]. One can frame this challenge of finding the best solver for a given problem into the Metaheuristic Composition Optimization Problem (MCOP) [10]. There are a few works that have contributed to this area. For instance, Miranda *et al.* created a framework for learning when to use different tuning parameter settings for Particle Swarm Optimisation (PSO) [11]. Moreover, Cruz-Duarte *et al.* reported a framework called CUSTOMHyS, after Customising Optimisation Metaheuristics via Hyper-Heuristic Search, which generates MHs [8]. They claimed that their resulting solver reached astonishing performances compared against traditionally created (handcrafted and 'bio-inspired') MHs.

Considering this last work, it is easy to infer several opportunities to explore this field. For example, it is more than evident that solving the MCOP is limited to efficiently exploring the heuristic space. Regard that this is not a merely combinatorial space; each operator depends on its tuning parameters and its interaction with other operators when

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.

dealing with an optimisation problem. Another drawback to bear in mind is the memory-less feature of the implemented approaches. There are many problems and search operators that share characteristics with their peers, which could be fruitful to consider.

This work proposes two novel methods based on transfer learning to facilitate the automatic generation of population-based and metaphor-less unfolded metaheuristics. These methodologies contribute directly to the MCOP, a highly relevant topic in the literature. The first strategy statically generates the whole candidate sequence before implementing it. In contrast, the second one deals with the problem dynamically, building the sequence while solving the low-level problem. To test our approaches, we utilise the formal model called unfolded metaheuristics (uMHs), recently proposed in [12]. This model generalises the standard heuristic model, which is not limited to a kind of search operator iterating cyclically but considers a heterogeneous sequence of these operators. As the transfer learning component, we extracted information from previous uMH implementations to build weight matrices related to the interaction of simple heuristics over 32 continuous problems. Results provide us with information to prove the feasibility of these approaches via experiments using these problems with four different combinations of characteristics and four dimensionality values while varying the number of agents (30, 50, and 100) employed by the search operators. Moreover, we highlight that these two methodologies have comparable performance, but their potential usage depends on the general application environment.

The remainder of this manuscript is organised as follows: Section II briefly describes the theoretical concepts utilised in this work. Then, Section III details the proposed approaches described above. Section IV presents the methodology we followed to test our proposal. Subsequently, Section V discusses all results achieved from the experiments carried out. Finally, Section VI synthesises the main insights and brainstorms some future research paths.

## II. Foundations

Several of these concepts may seem trivial, but we establish those definitions which we consider paramount to avoid misunderstandings and controversies. So, for the sake of standardisation, we use the notation defined in [13] for heuristic-based methods.

### A. Optimisation

An optimisation process is defined as a mathematical procedure for minimising an objective function $f(\vec{x}) : \mathfrak{X} \mapsto \mathbb{R}$, in a feasible domain $\mathfrak{X} \subseteq \mathfrak{G}$ since $\mathfrak{G}$ is an arbitrary domain. Thus, a minimisation problem can be represented with the tuple $(\mathfrak{X}, f)$ such as

$$\vec{x_*} = \underset{\vec{x} \in \mathfrak{X}}{\arg\min}\{f(\vec{x})\}, \tag{1}$$

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

This work deals with two types of optimisation problem domains, continuous and combinatorial, at different levels of abstractions. For the continuous one, the low-level, we represent it as $\mathfrak{S} = \mathbb{R}^D$, where $D$ stands for the number of dimensions that the problem contains. For the combinatorial problem, the high-level, we say that $\mathfrak{S} = \mathfrak{H}^{\varpi}$, where $\mathfrak{H}$ is the heuristic space and $\varpi$ is the cardinality of the sequences. Indeed, this high-level problem is the Metaheuristic Composition Optimisation Problem (MCOP). We describe next the essential components for designing metaheuristics from heuristics within the heuristic space and how they interact directly with the low-level domain.

### B. Heuristics

We can interpret a heuristic as a sequence of actions [9]. Such a sequence may contain a single or multiple instructions that do not necessarily follow a sequential pattern [13]. According to their level of abstraction, they can be categorised into *simple heuristics*, *metaheuristics*, and *hyper-heuristics*.

First of all, since most heuristics operate over a population (*i.e.,* a set of agents), like those implemented in this work, it is necessary to define such a population. Thus, a population consists on a finite set of $N$ candidate solutions, which is denoted as $X(t) = \{\vec{x}_1(t), \ldots, \vec{x}_N(t)\}$. When having multiple candidate solutions, one must find a sensible way of picking up the best one. To do so, we consider an arbitrary set of candidate solutions $Z(t)$, which can be designated as, *e.g.,* the entire population ($Z(t) = X(t)$) or the historical evolution of the $n$-th candidate ($Z(t) = \{\vec{x}_n(0), \ldots, \vec{x}_n(t)\}$). Hence, let $\vec{x}_*(t) \in Z(t)$ be the best position w.r.t. the objective function from $Z(t)$, *i.e.,* $\vec{x}_*(t) = \operatorname{arginf}\{f(Z(t))\}$.

Bearing this information in mind, we proceed to briefly describe the heuristics mentioned above as follows:

*Simple Heuristics* (SHs) produce (initialisers), modify (search operators), or evaluate a candidate solution (finaliser). In that sense, it is noticeable that SHs are the building blocks or primitives for generating more sophisticated methods.

*Metaheuristics* (MHs) can be defined in general terms as sequences of heuristics. So, an MH can be an iterative procedure that renders an optimal solution for a given optimisation problem. It is common to find MHs with a master strategy, the finaliser, which controls the iterative procedure. However, in this work, we employ the general model called *Unfolded Metaheuristic* (uMH), which delegates the finaliser role to a superior strategy. Thus, we only deal with sequences of heuristics.

*Hyper-heuristics* (HHs) manage low-level heuristics to produce an adequate combination of them to solve the problem effectively, instead of manipulating the solutions of the problem to find the optimal solution. So, a HH searches for the optimal heuristic configuration (or metaheuristic) that best approaches to the solution of $(\mathfrak{X}, f)$ within the maximal performance by solving

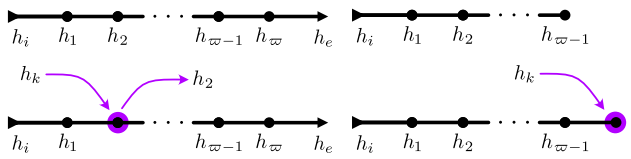
$$(\boldsymbol{h}_*; \vec{x}_*) = \underset{\boldsymbol{h} \in \mathfrak{H}^{\varpi},\, \vec{x} \in \mathfrak{X}}{\operatorname{argmax}} \{Q(\boldsymbol{h}|\mathfrak{X}, f)\} \tag{2}$$

where $Q(\vec{h}|\mathfrak{X}, f) : \mathfrak{H}^{\varpi} \rightarrow \mathbb{R}_{+}$ is a metric that measures the performance of $\vec{h}$ when it is applied to the low-level problem $(\mathfrak{X}, f)$ [12].

It is worth mentioning that the high-level optimisation problem is well known as Metaheuristic Composition Optimisation Problem (MCOP) [10]. So, it is possible to implement a process for tackling an arbitrary optimisation problem by finding its optimal solver through a hyper-heuristic procedure. In this work, we consider the continuous optimisation problems as low-level problems and the metaheuristics as solvers. To build and search within the heuristic space $\mathfrak{H}^{\varpi}$, we employed the unfolded metaheuristic model [12]. This is an extended version of the traditional MH model.

## III. Proposed approaches

In this section, we describe the two approaches proposed in this work, *i.e.,* the Static and Dynamic Transfer Learning Hyper-heuristics. Fig. 1 presents an illustrative example of these approaches to provide an overview of the most significant differences amongst them. In a few words, the Static one modifies the candidate sequence of heuristics before evaluating it. In contrast, the Dynamic strategy builds the sequence while applying the operator to the low-level problem.




(a) Static approach (b) Dynamic approach

Fig. 1: Illustrative example of the approaches proposed and implemented in this work. The sketches show one hyper-heuristic step and how the current unfolded metaheuristic can be modified.

Now, let us set some common parts that both strategies utilise for dealing with the MCOP. The first one is the weight matrix $\boldsymbol{W}$, which has the following form:

$$\boldsymbol{W} = \begin{pmatrix} w_{1,1} & \cdots & w_{1,\#\mathfrak{H}_o} \\ \vdots & \ddots & \vdots \\ w_{\varpi_{\max},1} & \cdots & w_{\varpi_{\max},\#\mathfrak{H}_o} \end{pmatrix}_{(\varpi_{\max} \times \#\mathfrak{H}_o)}, \quad (3)$$

since $\boldsymbol{W}_{(\varpi_{\max} \times \#\mathfrak{H}_o)} \cdot \vec{1}_{(\#\mathfrak{H}_o \times 1)} = \vec{1}_{(\varpi_{\max} \times 1)}$. This last expression is just the well-known property of any probability distribution. It means that each row in $\boldsymbol{W}$ is a normalised frequency distribution of search operators, which have been employed in that iteration ($\varpi$). As one may notice, $\boldsymbol{W}$ corresponds to a time-heuristic ($\varpi$-$h$) spectrum for a given low-level problem.

The second essential part to bear in mind is the way of assessing how well an unfolded metaheuristic performs. This measurement is basically the objective function in the high-level optimisation problem, *i.e.,* MCOP, described in (2). So, we determine the performance of an arbitrary candidate heuristic sequence $\boldsymbol{h}$, 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 implementing (4), it is necessary to run each candidate 50 times and record all fitness values; this can guarantee the statistical significance of the metric. Still, this number of repetitions can be greater, but it will affect the overall computing time, particularly for the Static strategy.

### A. Static transfer learning hyper-heuristic

This first approach, Static Transfer Learning Hyper-Heuristic (STLHH), corresponds to a combinatorial Random Search based on rules and employing a Greedy selector. So, the hyper-heuristic randomly draws one action and applies it on the current heuristic sequence $\boldsymbol{h}$. We base this proposal on Simulated Annealing Hyper-Heuristics [14], [15], where the authors implemented several actions. However, as we now describe, we only choose and design actions that avoid drastic changes in the candidate sequence.

`Add` inserts a randomly chosen heuristic at a random l.h.s. position $i \sim \mathcal{U}\{1, \varpi+1\}$.
`AddMany` performs `Add` $\varpi_i$ times since $\varpi_i \sim \mathcal{U}\{2, \varpi_{\max}-\varpi\}$, where $\varpi_{\max}$ is the maximal cardinality.
`Remove` discards a $h_i$ from a random position $i \sim \mathcal{U}\{1, \varpi\}$.
`RemoveMany` performs `Remove` $\varpi_j$ times since $\varpi_j \sim \mathcal{U}\{1, \varpi - \varpi_{\min}\}$, where $\varpi_{\min}$ is the minimal cardinality.
`Shift` selects a random position $i \sim \mathcal{U}\{1, \varpi\}$ and changes the corresponding $h_i$ with another chosen at random.
`ShiftMany` performs `Shift` $\varpi_j \sim \mathcal{U}\{2, \varpi\}$ times.
`Swap` interchanges heuristics $h_i, h_j$ at two randomly selected locations, $i, j \sim \mathcal{U}\{1, \varpi\}$.
`RemoveLast` discards only the last heuristic $h_{\varpi}$.

It is essential to mention that only the random selection of indices use uniform probabilistic distributions. For randomly picking a heuristic up, we considered the probability distributions given by the weight matrix from the learnt information transferred to the process. Consider the illustrative example in Fig. 1. There is a simple `Shift` action being applied, where the operator $h_k$ was previously selected at random with $\boldsymbol{W}$.

Therefore, once a candidate unfolded metaheuristic is drawn, the Static method implements it in the low-level problem and then determines its performance via (4). The subsequent steps are the trivial ones from any optimisation algorithm, as one can notice in Pseudocode 1.

### B. Dynamic transfer learning hyper-heuristic

The second approach, Dynamic Transfer Learning Hyper-Heuristic (DTLHH), is conceptually different from the first one because it is intended to build the unfolded metaheuristic whilst solving the optimisation problem. In that sense, the Dynamic hyper-heuristic performs a high-level step, in the heuristic domain, for choosing the next search operator and

**Pseudocode 1** Static Transfer Learning Hyper-heuristic

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

$\boldsymbol{h} \leftarrow$ ChooseRandomly($\mathfrak{H}_o$, $\boldsymbol{W}$)
**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), $\boldsymbol{h}_* \leftarrow \boldsymbol{h}$ and $s \leftarrow 0$
**while** $(s \leq s_{\max})$ **and** AdditionalCriteria( ) **do**
  $a \leftarrow$ ChooseAction($A, a, \varpi$), 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** $Q(\boldsymbol{h}_c|\mathfrak{X}) \leq Q(\boldsymbol{h}|\mathfrak{X}))$ **then** $\boldsymbol{h} \leftarrow \boldsymbol{h}_c$, **end if**
  **if** $Q(\boldsymbol{h}_c|\mathfrak{X}) \leq Q(\boldsymbol{h}_*|\mathfrak{X})$ **then** $\boldsymbol{h}_* \leftarrow \boldsymbol{h}$ and $s \leftarrow 0$,
  **else** $s \leftarrow s + 1$, **end if**
**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**

**Pseudocode 2** Dynamic Transfer Learning Hyper-heuristic

**Input:** Domain $\mathfrak{X}$, objective function $f(\vec{x})$, heuristic space $\mathfrak{H}_o$, initialiser $h_i$, population size $N$, maximum cardinality $\varpi_{\max}$, maximum number of repetitions $N_{\text{r}}$, and weight matrix $\boldsymbol{W}$.
**Output:** Best unfolded metaheuristic $\boldsymbol{h}_*$

$\boldsymbol{W}' = \boldsymbol{W}$
**for** $r = 1, \ldots, N_r$ **do**
  $X(0), F(0), \vec{x}_*(0) \leftarrow h_i\{\mathfrak{X}, N\}$
  $\boldsymbol{h}_* \leftarrow$ Concatenate($\{\}, h_i$)
  $\varpi \leftarrow 0$, $s_t \leftarrow 0$
  **while** $\varpi \leq \varpi_{\max}$ **and** AdditionalCriteria( ) **do**
    $P_{\mathfrak{H}}(h) \leftarrow$ Choose($\boldsymbol{W}', \varpi, r$)
    $h_j \leftarrow$ ChooseRandomly($\mathfrak{H}_o$, $P_{\mathfrak{H}}(h)$)
    $X_c, F_c, \vec{x}_{c,*} \leftarrow h_j\{X(\varpi)\}$
    **if** $f(\vec{x}_{c,*}) < f(\vec{x}_*(0))$ **and** AdditionalCriteria( ) **then**
      $X(0), F(0), \vec{x}_*(0) \leftarrow X_c, F_c, \vec{x}_{c,*}$
      $\boldsymbol{h}_* \leftarrow$ Concatenate($\boldsymbol{h}_*, h_j$), $\varpi \leftarrow \varpi + 1$, $s_t \leftarrow 0$
      **else** $s_t \leftarrow s_t + 1$
    **end if**
  **end while**
  $\boldsymbol{H} \leftarrow$ Append($\boldsymbol{H}, \boldsymbol{h}_*$), and $\boldsymbol{W}' \leftarrow$ Update($\boldsymbol{H}, r$)
**end for**

**procedure** Update($\boldsymbol{H}, r$)
  **if** $r \geq N_r/e$ **then**
    **for** $\varpi = 0, \ldots, \varpi_{\max}$ **do**
      $\boldsymbol{W}' \ni \vec{W}_\varpi \leftarrow$ Histogram($\{H_{\varpi,1}, \ldots, H_{\varpi,r}\}$)
    **end for**
    **else** $\boldsymbol{W}' = \boldsymbol{W}$
  **end if**
  **return** $\boldsymbol{W}'$
**end procedure**

then moves a low-level step by applying it to the problem domain.

It is pretty easy to notice that this strategy is less cost computing than the Static one; consequently, it is faster. Nevertheless, this idea has a drawback: it generates solvers with higher performance variance. This is because of the stochastic nature of many operators, which one can extract from several well-known metaheuristics in the literature. A single evaluation of a candidate SO does not suffice for guaranteeing a proper solution. Therefore, one way to attend to this issue is to run many implementations of the same operator for each new guess to determine a statistically significant measurement of its performance. However, this is not computationally cheap. So, we implement a different approach to gather information about which search operator performs best, considering its interaction with the other operators. Keeping things as simple as possible, we propose DTLHH based on the well-known Optimal Stopping problem. So, we allow the HH procedure to use the weight matrix obtained from previous experiments for about 37% of the total budget of HH runs. These procedures are just repetitions of the same composition optimisation problem-solving. After that, the HH starts using a weight matrix determined by the frequency of the SOs employed in each position within sequences. Other additional steps are summarised in Pseudocode 2.

### C. Complexity analysis

To wrap up the description of our proposals, we now study their time complexity. First, consider the low-level implementation when only a heuristic sequence is applied to the optimisation problem. So, the time complexity for an unfolded metaheuristic composed of $\varpi_{\max}$ search operators and performing $t_{\max}$ iterations yields

$$T_{\text{uMH}} = \mathcal{O}\left(\varpi_{\max} * N * D * T_p\right), \tag{5}$$

where $N$ is the population size, $D$ is the dimensionality, and $T_p$ is the computational cost of evaluating the perturbator. For the sake of simplicity, we consider $T_p = \mathcal{O}(N^2)$ for the Genetic Crossover perturbator with Cost-based Roulette Wheel Pairing [16]. This search operator belongs to the collection of simple heuristics extracted for this work. It represents the worst case-scenario compared to the time required by the selector and finaliser [8].

At a high level, the two approaches have a complexity similar to that of a regular and single-agent MH, but with some slight differences to consider. For STLHH, we find that

$$\begin{aligned} T_{\text{STLHH}} &= \mathcal{O}\left(N_r * T_{\text{uMH}} * s_{\max}\right), \\ &= \mathcal{O}\left(N_r * \varpi_{\max} * N^3 * D * s_{\max}\right), \end{aligned} \tag{6}$$

since $N_r$ is the number of repetitions, $T_{\text{uMH}}$ is the cost of implementing a candidate sequence, and $s_{\max}$ stands for the steps carried out in this search.

Likewise, we determine the complexity for DTLHH as given

$$\begin{aligned} T_{\text{DTLHH}} &= \mathcal{O}\left(N_r * (T_{\text{uMH}} + T_{\text{updateW}})\right), \\ &= \mathcal{O}\left(N_r * \varpi_{\max}^2 * N^3 * D\right), \end{aligned} \tag{7}$$

since $T_{\text{updateW}}$ is the time complexity of updating the weight matrix, which can be disregarded for short collections of search operators and low cardinalities [17].

From the previous analysis, we must remark that the running time scales linearly with the problem dimensionality and

cubically with the population size. Such a time complexity is somehow manageable for both implemented methodologies. Now, the difference between STLHH and DTLHH is given by the maximum number of steps $s_{\max}$ and the maximum cardinality $\varpi_{\max}$. For avoiding a dramatic increase in the computing times, we decided to set these two variables equal to a hundred, $\varpi_{\max} = s_{\max} = 100$. Thus, the difference between their time complexities shall be related to the software implementation and the assumptions described above inherent to these prototyping studies.

## IV. Methodology

This work aims to test the transfer learning methodology through two different hyper-heuristic approaches (described in the previous section) to generate unfolded metaheuristics. To do so, we employed CUSTOMHyS v1.0. This open-access framework can be found at https://github.com/jcrvz/customhys along with its documentation, which was also provided by its authors in [18], and which we accessed on January 31st, 2022. CUSTOMHyS facilitates the implementation of MHs through population-based search operators acting as building blocks, which were extracted from MHs reported in the literature.

We carried out several experiments to study the Static and Dynamic implementations of hyper-heuristics powered by transfer learning. For these experiments, we employed populations of 30, 50, and 100 agents for the high-level problem domain, *i.e.,* the search operators or heuristic space. Likewise, we considered 2, 10, 30, and 50 dimensions for the continuous search space related to the benchmark functions in the low-level problem domain. Such low-level problems are described below.

We determined the initial set of weight matrices for each combination of population and dimensionality from previous results achieved with Simulated Annealing [12]; this corresponds to the transfer learning component. Note that we employed this information but did not implement the whole tests, as they build the solution from scratch, which is more time consuming than our proposed strategies. Instead, we used populations and dimensionalities previously implemented in other investigations [8], [15], [17]. Plus, we considered one dataset that used the same problems and populations for those search operators. We hereafter refer to it as the results obtained from the Base strategy [12], which employed the unfolded metaheuristic model. Thus, we compared the results from these three HH implementations.

Now, concerning the low level of abstraction related to the continuous optimisation problems, we employed 32 well-known benchmark functions from the literature [19]. These problems can also be found in CUSTOMHyS `benchmark_func` module. TABLE I lists these functions and their categorical features of Differentiability and Unimodality. Notice that we chose eight problems for each combination of these features. In all the experiments, we used 2, 10, 30, and 50 dimensions for ensuring comparisons. Furthermore, due to the wide magnitudes of extreme values in function image ranges (fitness values), we regularised them into a short-range. This metric is given by

$$\hat{f}(f) \triangleq \operatorname{sgn}(f) \log(f \cdot \operatorname{sgn}(f) + 1), \tag{8}$$

where $\hat{f}(f) : \mathbb{R} \mapsto \mathbb{R}$ is the reduced range value of the evaluated objective function $f(\vec{x})$. The other components are the sign $\operatorname{sgn} : \mathbb{R} \mapsto \{-1, 1\}$ and the decimal logarithm $\log : \mathbb{R}_{++} \mapsto \mathbb{R}$ functions. So, for comparison purposes, we used the adjusted value of $f$, $\hat{f}$, that we called 'metric', for example, to determine the performance of a given uMH according to (4).

TABLE I: Continuous optimisation problems from the literature [19] that we selected for this work.

| | Differentiable | Non-differentiable |
|---|---|---|
| Unimodal | Cigar | Schwefel 2.20 |
| | Dixon-Price | Schwefel 2.21 |
| | Ellipsoid | Schwefel 2.22 |
| | Perm 02 | Step |
| | Powel Sum | Step 2 |
| | Schwefel 1.2 | Step 3 |
| | Sphere | Step Int |
| | Trid | Type-I Simple Deceptive Problem |
| Multimodal | Bohachevsky | Alpine 1 |
| | F2 | Cross-Leg Table |
| | Griewank | Expanded Decreasing Minima |
| | Schaffer N1 | Expanded Uneven Minima |
| | Schaffer N2 | Lunacek N01 |
| | Trigonometric 2 | Price 01 |
| | W-Wavy | Stochastic |
| | Weierstrass | Type-II Medium-Complex Deceptive |

Subsequently, in the high level of abstraction, we used the default collection of 205 search operators (SOs) achieved by varying parameters, distributions, and schemes, as the hyper-heuristic problem domain. These SOs are also included in the CUSTOMHyS framework, and they are fully described in prior works [13], [18], [20]. We must remember that a search operator comprises a perturbator and a selector. Both of them are population-based simple heuristics with different roles in the low-level problem domain. Hence, TABLE II displays the perturbators and selectors utilised in the experiments of this work. These heuristics were extracted from the following ten well-known MHs: Random Search, Simulated Annealing, Genetic Algorithm, Cuckoo Search, Differential Evolution, Particle Swarm Optimisation, Firefly Algorithm, Spiral Optimisation, Central Force Optimisation, and Gravitational Search Algorithm. We skipped the acronyms for avoiding overwhelming the reader at this point, but further details are provided in [18]. Moreover, we set the maximal cardinality for the unfolded metaheuristics to 100; *i.e.,* the heuristic sequences to generate will be up to 100 search operators.

For this manuscript, we used Python 3.7 and a Dell Inc. PowerEdge R840 Rack Server with 16 Intel Xeon Gold 5122 CPUs @ 3.60 GHz, 125 GB RAM, and CentOS Linux release 7.6.1810-64 bit system for running our experiments. All the resulting data is also freely available at https://github.com/jcrvz/tl-hh-umhs, accessed on May 23th, 2022.

TABLE II: Population-based perturbators and selectors extracted from 10 well-known metaheuristics in the literature.

| | Name | Expressions |
|---|---|---|
| Perturbators | Random Sample | $\vec{y}_n = \vec{r}$ |
| | Random Walk | $\vec{y}_n = \vec{x}_n + \alpha\vec{r}$ |
| | Local Random Walk | $\vec{y}_n = \vec{x}_n + \alpha\vec{r} \odot H(\vec{r} - p) \odot (\vec{x}_{z_1} - \vec{x}_{z_2})$ |
| | Random Flight | $\vec{y}_n = \vec{x}_n + \alpha\vec{r} \odot (\vec{x}_n - \vec{x}_*)$ |
| | Genetic Crossover | $\vec{y}_n = \vec{m} \odot \hat{\vec{x}}_{z_1} + (1 - \vec{m}) \odot \hat{\vec{x}}_{z_2}$ |
| | Genetic Mutation | $\vec{y}_n = \vec{m} \odot \vec{x}_n + \alpha(1 - \vec{m}) \odot \vec{r}$, $\vec{m} = H(p_m - \vec{q})$ |
| | Differential Mutation | $\vec{y}_n = \vec{x}_{d_1} + \alpha_0(\vec{x}_{d_2} - \vec{x}_{d_3}) + \alpha_m \sum_{m=1}^{M} \left(\vec{x}_{z_{2m-1}} - \vec{x}_{z_{2m}}\right)$ |
| | Spiral Dynamic | $\vec{y}_n = \vec{x}_* + \vec{r}\mathbf{R}_D(\theta) \cdot (\vec{x}_n - \vec{x}_*)$ |
| | Swarm Dynamic | $\vec{y}_n = \vec{x}_n + \vec{v}_n(t)$, $\vec{v}_n(t) = \alpha_0\vec{v}_n(t-1) + \alpha_1\vec{r}_1 \odot (\vec{x}_{n,*} - \vec{x}_n) + \alpha_2\vec{r}_2 \odot (\vec{x}_* - \vec{x}_n)$ |
| | Firefly Dynamic | $\vec{y}_n = \vec{x}_n + \alpha_0\vec{r} + \alpha_1 \sum_{k=1,k\neq n}^{N} H(-\Delta I_{n,k})\Delta\vec{x}_{n,k} e^{-\alpha_2 \|\|\Delta\vec{x}_{n,k}\|\|_2^2}$, $\Delta I_{n,k} = f(\vec{x}_k) - f(\vec{x}_n)$, $\Delta\vec{x}_{n,k} = \vec{x}_k - \vec{x}_n$ |
| | Central Force Dynamic | $\vec{y}_n = \vec{x}_n(t) + \frac{1}{2}\vec{a}_n\Delta t^2$, $\vec{a}_n = \alpha_0 \sum_{k=1,k\neq n}^{N} \frac{m_{n,k}\cdot(\vec{x}_k - \vec{x}_n)}{\|\|\vec{x}_k - \vec{x}_n\|\|_2^{\alpha_2} + \varepsilon}$, $m_{n,k} = H(f(\vec{x}_k) - f(\vec{x}_n))(f(\vec{x}_k) - f(\vec{x}_n))^{\alpha_1}$ |
| | Gravitational Search | $\vec{y}_n = \vec{x}_n + \vec{v}_n(t)$, $\vec{v}_n(t) = \vec{r} \odot \vec{v}_n(t-1) + \vec{a}_n(t)$, $\vec{a}_n(t) = \frac{\alpha_0}{e^{\alpha_1 t}} \sum_{\substack{k=1 \\ k\neq n}}^{N} \frac{(f(\vec{x}_\circ(t)) - f(\vec{x}_n))\vec{r}_k \odot (\vec{x}_k - \vec{x}_n)}{(Nf(\vec{x}_\circ) - \sum_{k=1}^{N} f(\vec{x}_k))\|\|\vec{x}_k - \vec{x}_n\|\|_2 + \varepsilon}$ |
| Selectors | 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 | $\vec{x}_n(t+1) = \begin{cases} \vec{y}_n, & \text{if } (\Delta f_n \leq 0) \vee (r < e^{-\frac{\Delta f_n(t)}{k_B \Theta(t)}}), \\ \vec{x}_n(t), & \text{otherwise,} \end{cases}$ $r \sim \mathcal{U}(0, 1)$, $k_B \in \mathbb{R}_+$, $\Theta(t) : \mathbb{Z}_+ \to \mathbb{R}_+$ |

* $\vec{x}_n$ 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. $\vec{r}$ is a vector of i.i.d. random variables with either Uniform, Normal or Lévy stable distribution. $\hat{\vec{x}}_{z_1}$ 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. $\mathbf{R}_D(\theta)$ is the rotation matrix; $r_l, r_u$ are the lower and upper radius threshold. $\alpha_k$, $\forall k$, is an arbitrary coefficient.

## V. Results

First of all, in the following charts, we prefer the strip plot representation over the violin- or box-plots because many problems have such different magnitudes even after regularising their fitness values. So, these two kinds of charts are inappropriate.

Fig. 2 shows the distribution of metric values achieved using three hyper-heuristic strategies (Base, Dynamic, and Static), solving low-level problems with four different dimensionalities (2, 10, 30, and 50) and employing search operators from three distinct populations (30, 50, and 100). Recall that this metric value corresponds to the rescaled version for the objective function presented in (8). The low performance of the Base strategy is noticeable against the other two methodologies based on hyper-heuristics and transfer learning. In a general overview, one can regard a decreasing tendency when observing results from the Base, Dynamic, and Static techniques, in that order, and when increasing the population size for search operators to use. This improvement is quite expected due to results from the Base strategy being achieved without prior information. Naturally, Base strategy comprises an intelligent approach for searching the best heuristic sequence for solving a given problem, but in a memoryless fashion. Even remembering the previous steps, rendering a similar solution could not be possible. The reason for that is because both proposed strategies, Dynamic and Static, start from previous knowledge: the transferred information. This was achieved by determining not only the interaction of two search operators in problems sharing similar characteristics but also considering the causality of the sequence. Speaking in the Machine Learning jargon, we must say that the transfer matrices are more or less an early version of the Transformer Attention Mechanism [21]. Another important detail to consider from this first plot (Fig. 2) is that the larger the dimensionality, the higher the problem complexity, *i.e.,* the curse of dimensionality. This fact becomes evident when all the strategies deal with 30- and 50-dimensionality problems, where even using the function regularisation, some benchmark problems get larger fitness values, which goes virtually unnoticed in 10D.

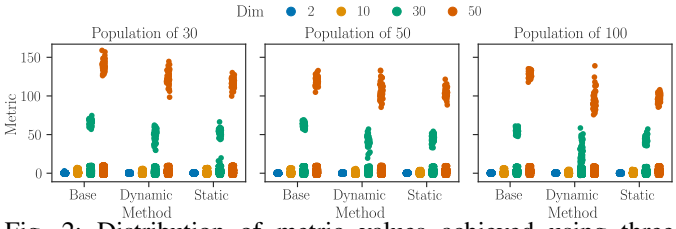



Fig. 2: Distribution of metric values achieved using three hyper-heuristic strategies employing search operators from three distinct populations for solving low-level problems with four different dimensionalities.

After glancing at the results, let us now go a little deeper into the lagoon of achieved data, in this case, considering two categorical characteristics of the low-level problems. As mentioned in the previous section, we tried to equalise the number of functions (*i.e.,* eight) per combination of these characteristics: Differentiability and Unimodality. Fig. 3 displays the results by considering these characteristics as well as population (columns) and dimensionality (rows) for all three hyper-heuristic strategies. At a glance, it is evident that it was a tough competition between all the HH implementations. There is no chance to catch a clear inference about which strategy is the best one; an expected behaviour if one remembers that two-dimensional problems are the easiest to solve. When dimensionality is ten or larger, it is easy to corroborate the first insight about the proposed methods beating the Basic one. However, regarding the characteristics and populations, we find no drastic impact on the performance of a particular strategy; the observed tendency is somehow preserved.

Now, to get a meaningful inference about the results and how well the proposed methodologies perform against the base one, we rank all the HH strategies per problem. Fig. 4 displays the frequency of first places reached by each of the hyper-heuristic methodologies for different dimensionalities and populations. Notice that these frequencies were normalised to facilitate their analysis. We can observe that the Dynamic methodology is present in all the configurations followed by the Static one. It is also evident that the proposed

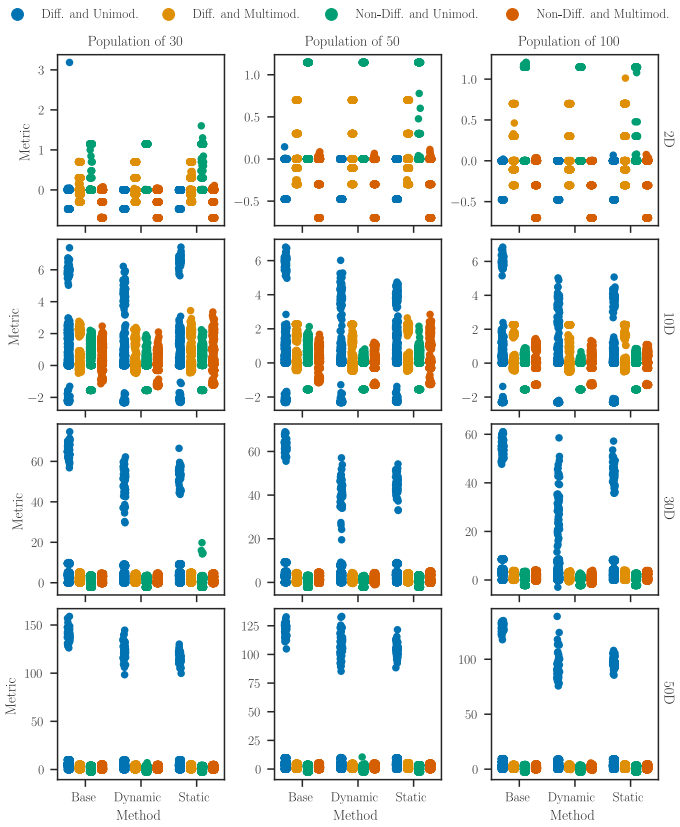



Fig. 3: Distribution of metric values achieved using three hyper-heuristic strategies employing search operators from three distinct populations for solving low-level problems with four combinations of features.

methodologies outperform the Base implementation for almost all scenarios. However, when the number of dimensions is 50, there are unfolded metaheuristics designed by Dynamic strategy that manage a lower number of agents to achieve a good solution for low-level problems. Another relevant insight is that the Basic method utilised more resources to generate the uMHs than the Static one. So, when using 30 agents, we can achieve a comparable success rate between the Static and Base implementations. The difference is that the former employs less computing time than the latter, thanks to the transferred information. Increasing the population size naturally increases the capabilities of certain types of search operators. So, even in 50D problems, Base methodology competes against the others using transfer learning. It is worth mentioning that the ranking carried out in this analysis allows ties. We considered including the classification based on the problem characteristics, but we did not find relevant information from such a representation. We attribute this to the nature of hyper-heuristic methodologies implemented; they deal with a given problem by choosing the right combination of SOs to solve it.

Subsequently, we evaluate the results by carrying out multiple comparisons using the pairwise (one-sided) Wilcoxon's signed-rank test. First, we need to establish the null and alternative hypotheses to perform Wilcoxon's tests.

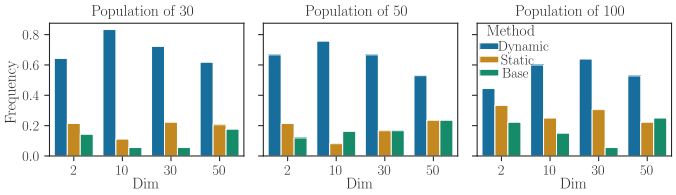



Fig. 4: Frequency of first places occupied from ranking the three hyper-heuristic methodologies implemented in this work for solving continuous optimisation problems of four different dimensionalities and employing three population sizes for the search operators.

$\mathbf{H}_0$: The unfolded metaheuristic, generated by the hyper-heuristic methodology based on transfer learning, performs equal to or worse than the unfolded metaheuristic generated by the Base strategy with a significance level $\alpha = 0.05$.

$\mathbf{H}_1$: The unfolded metaheuristic, generated by the hyper-heuristic methodology based on transfer learning, outperforms the unfolded metaheuristic generated by the Base strategy with a significance level $\alpha = 0.05$.

TABLE III shows the $p$-values achieved after carrying out Wilcoxon's test for each combination of population and dimensionality. There is evidence that we can reject the null hypothesis, so we can corroborate that DTLHH corresponds to an excellent alternative for designing unfolded metaheuristics based on previous inferences. Nevertheless, when comparing Static against Base, we can only reject the null hypothesis when the search operators use populations of 30 agents. This is expected behaviour that corroborates the previous inferences from Fig. 4. Likewise, there is not enough evidence to reject the null hypothesis for 10D and 30D, with populations of 50 agents and 50D with populations of 100 agents. However, consider that the Static outperforms the Base methodology in 9 out of 12; for those other three cases, we have not found an abysmal difference between these strategies. We believe that the STLHH's performance can be enhanced by allowing more steps during its search. Recall that it has shown comparable results to the Base implementation even in the current state, which comprised more computations. Bearing this information in mind, we must remark that any of the proposed strategies can be implemented for delivering a customised heuristic-based method for solving a problem with specific characteristics. The DTLHH approach proves to be useful for those situations when a solver is required quickly, *e.g.*, an initial algorithm for further modifications. Otherwise, the Static strategy shows its strengths in those cases where the computing budget is not a limitation. Note that we do not compare DTLHH and STLHH directly because they are two tangentially opposite approaches; it would be unfair.

## VI. Conclusions

This work presented two novel and still simple methodologies based on transfer learning to facilitate the automatic

TABLE III: $p$-values obtained from the pairwise (one-sided) Wilcoxon's signed-rank test with a significance level $\alpha = 0.05$. Values in scientific notation are those less than 0.05.

| | | $p$-value | |
|---|---|---|---|
| Pop | Dim | Base vs. Dynamic | Base vs. Static |
| 30 | 2 | $3.90\times10^{-6}$ | $2.69\times10^{-3}$ |
| | 10 | $4.46\times10^{-6}$ | $3.70\times10^{-2}$ |
| | 30 | $6.07\times10^{-7}$ | $1.21\times10^{-2}$ |
| | 50 | $4.83\times10^{-5}$ | $3.26\times10^{-3}$ |
| 50 | 2 | $7.63\times10^{-7}$ | $4.65\times10^{-5}$ |
| | 10 | $2.45\times10^{-5}$ | 0.5997 |
| | 30 | $6.07\times10^{-5}$ | 0.2243 |
| | 50 | $9.47\times10^{-3}$ | $2.43\times10^{-2}$ |
| 100 | 2 | $2.91\times10^{-4}$ | $3.22\times10^{-2}$ |
| | 10 | $1.31\times10^{-4}$ | $1.64\times10^{-3}$ |
| | 30 | $4.09\times10^{-6}$ | $3.85\times10^{-3}$ |
| | 50 | $2.05\times10^{-3}$ | 0.4814 |

generation of population-based and metaphor-less metaheuristics (MHs) by using search operators from the literature. For these MHs, we utilised the formal model called unfolded metaheuristics, which is not limited to a kind of search operator iterating cyclically but can consider a heterogeneous sequence of these operators. The first strategy does it statically by generating the whole candidate sequence before implementing it. In contrast, the second one deals with the problem dynamically, building the sequence while solving the low-level problem. To test our approaches, we used 32 continuous optimisation problems divided into four subgroups of eight based on combining of two characteristics (Differentiability and Unimodality). We considered four dimensionalities for these low-level problem domains (2, 10, 30, 50). Plus, in the high-level domain regarding the search operators (SOs) or metaheuristic building blocks, we implemented a collection of 205 SOs based on a population of agents. Additionally, we used three population sizes (30, 50, 100) to test our approaches.

Results presented us information to prove the feasibility of these approaches via several experiments using problems with different characteristics and dimensions and varying the number of agents employed by the operators. We also remarked that one could compare these two methodologies on performance, but we emphasise their potential usage depending on the general application environment. The Dynamic approach showed to be useful for those situations when a solver is required quickly, *e.g.*, an initial algorithm for further modifications. Nonetheless, we recommend the Static strategy for those cases where the computing budget is not a limitation and when a low variance method is required.

It is noticeable that there are many paths to follow from this work. The most evident is implementing these proposed hyper-heuristics to solve other problems, such as those from the CEC and GECCO competitions. We also plan to tackle practical engineering applications, such as photovoltaic arrangement design, tumour image segmentation, and electronic thermal management problems. Moreover, we will consider implementing the proposed methodology to solve new problems with similar features but not the same as those used to extract the transfer matrices. It is possible by employing a bilinear interpolation procedure to infer initial information that one can refine through an online learning methodology.

## REFERENCES


[1] K. Sörensen, M. Sevaux, and F. Glover, "A history of metaheuristics," *Handbook of Heuristics*, vol. 2-2, pp. 791–808, 2018.

[2] A. A. Juan, P. Keenan, R. Martí, S. McGarraghy, J. Panadero, P. Carroll, and D. Oliva, "A review of the role of heuristics in stochastic optimisation: from metaheuristics to learnheuristics," *Ann. Oper. Res.*, jun 2021.

[3] 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.

[4] 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.

[5] İ. Gölcük and F. B. Ozsoydan, "Q-learning and hyper-heuristic based algorithm recommendation for changing environments," *Engineering Applications of Artificial Intelligence*, vol. 102, p. 104284, 2021.

[6] V. C. SS and A. HS, "Nature inspired meta heuristic algorithms for optimization problems," *Computing*, pp. 1–19, 2021.

[7] M. Sánchez, J. M. Cruz-Duarte, J. carlos Ortíz-Bayliss, H. Ceballos, H. Terashima-Marin, and I. Amaya, "A systematic review of hyper-heuristics on combinatorial optimization problems," *IEEE Access*, vol. 8, pp. 128068–128095, 2020.

[8] J. M. Cruz-Duarte, I. Amaya, J. C. Ortiz-Bayliss, S. E. Conant-Pablos, H. Terashima-Marín, and Y. Shi, "Hyper-heuristics to customise metaheuristics for continuous optimisation," *Swarm and Evolutionary Computation*, vol. 66, p. 100935, 2021.

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

[10] R. Qu, G. Kendall, and N. Pillay, "The general combinatorial optimisation problem: Towards automated algorithm design," *IEEE Comp. Intel. Magazine*, vol. 15, no. 2, pp. 14–23, 2020.

[11] 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.

[12] J. M. Cruz-Duarte, J. C. Ortiz-Bayliss, I. Amaya, and N. Pillay, "Global Optimisation through Hyper-Heuristics: Unfolding Population-Based Metaheuristics," *Appl. Sci.*, vol. 11, p. 5620, June 2021.

[13] 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.

[14] F. Garza-Santisteban, R. Sanchez-Pamanes, L. A. Puente-Rodriguez, I. Amaya, J. C. Ortiz-Bayliss, S. Conant-Pablos, and H. Terashima-Marin, "A Simulated Annealing Hyper-heuristic for Job Shop Scheduling Problems," in *2019 IEEE Congress on Evolutionary Computation (CEC)*, pp. 57–64, IEEE, 6 2019.

[15] J. M. Cruz-Duarte, I. Amaya, J. C. Ortiz-Bayliss, and N. Pillay, "Automated design of unfolded metaheuristics and the effect of population size," in *IEEE Congress on Evolutionary Computation (CEC)*, pp. 1155–1162, 2021.

[16] 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.

[17] J. M. Cruz-Duarte, I. Amaya, J. C. Ortiz-Bayliss, and N. Pillay, "Naïve hyper-heuristic online learning to generate unfolded population-based metaheuristics to solve continuous optimization problems," in *IEEE Symposium Series on Computational Intelligence (SSCI)*, pp. 1–8, 2021.

[18] 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.

[19] M. Jamil and X. S. Yang, "A literature survey of benchmark functions for global optimisation problems," *International Journal of Mathematical Modelling and Numerical Optimisation*, vol. 4, no. 2, p. 150, 2013.

[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] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2017.