# A Primary Study on Hyper-Heuristics Powered by Artificial Neural Networks for Customising Population-based Metaheuristics in Continuous Optimisation Problems

José M. Tapia-Avitia, Jorge M. Cruz-Duarte,
Ivan Amaya, José Carlos Ortiz-Bayliss, Hugo Terashima-Marín
*Tecnologico de Monterrey*
Monterrey, Mexico
{A00834191, jorge.cruz, iamaya2, jcobayliss, terashima}@tec.mx

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

***Abstract*—Metaheuristics (MHs) are proven powerful algorithms for solving non-linear optimisation problems over discrete, continuous, or mixed domains. Applications have ranged from basic sciences to applied technologies. Nowadays, the literature contains plenty of MHs based on exceptional ideas, but often, they are just recombining elements from other techniques. An alternative approach is to follow a standard model that customises population-based MHs, utilising simple heuristics extracted from well-known MHs. Different approaches have explored the combination of such simple heuristics, generating excellent results compared to the generic MHs. Nevertheless, they present limitations due to the nature of the metaheuristic used to study the heuristic space. This work investigates a field of action for implementing a model that takes advantage of previously modified MHs by learning how to boost the performance of the tailoring process. Following this reasoning, we propose a hyper-heuristic model based on Artificial Neural Networks (ANNs) trained with processed sequences of heuristics to identify patterns that one can use to generate better MHs. We prove the feasibility of this model by comparing the results against generic MHs and other approaches that tailor unfolded MHs. Our results evidenced that the proposed model outperformed an average of 84% of all scenarios; in particular, 89% of basic and 77% of unfolded approaches. Plus, we highlight the configurable capability of the proposed model, as it shows to be exceptionally versatile in regards to the computational budget, generating good results even with limited resources.**

***Index Terms*—Hyper-heuristic, Artificial Neural Networks, Search Operators, Optimisation, Evolutionary Computation.**

## I. Introduction

OPTIMISATION is an unquestionable field that has permeated all human technological advancements. Although already relevant, this topic will keep becoming increasingly relevant in the upcoming decades. Literature is so prolific that this fact can be quickly corroborated through an effortless search in any electronic library, where one can find astonishing approaches and applications. A particular approach, trendy and still relevant in recent years, is Metaheuristics (MHs) [1], which have shown a higher level of generalisation and have obtained competitive results for a class of problems. The term covers a plethora of algorithms of different nature. A rapid literature review [2] reveals that innovation in MH has somehow stalled or branched out far from the characteristics that make these methods striking and into hybrids and over-sophisticated methods. There is a lack of actual novel proposals for mathematical or technical procedures. One of the proposals to stop such tendency is to analyse the metaheuristics as a composition of independent modules (or simple heuristics) using metaphor-less descriptions [3]. This modular structure for metaheuristics can bring different advantages [4]. For example, it allows to systematically analyse metaheuristics according to their components and operators [5]. Moreover, it provides an easy way to configure hybrid metaheuristics, as proposed in [6]. Nevertheless, there is no MH that can approximate the optimal solution for every optimisation problem; *i.e.,* the No-Free-Lunch theorem [7]. Research and practitioners must know how to select an MH for a given problem in practice. Even then, the practitioner must also learn how to set their tuning parameters. Therefore, deciding which MH is worth implementing to solve a defined problem remains an open issue and represents a challenging and exciting path for research.

Many researchers have been working on designing algorithms with high-level abstraction to face such an issue. It means a methodology capable of generalising its process, adapting to particular environments, and reaching clever solutions compared to the algorithms from the literature. In this work, we propose an algorithm that, given a specific problem, automatically configures the composition of simple heuristics to generate modified metaheuristics that solve that problem. This approach is based on different levels of abstraction. The simple heuristics interact directly with the problem domain at the low level, either continuous or combinatorial. At the

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.

high level, it is a procedure exploring the heuristic space and testing different combinations of simple heuristics to find maximal performance. Here, we are no longer dealing with the practical problem domain. Instead, this problem was defined by Qu *et al.* [8] as an *Automated Algorithm Composition Problem* (AACP).

Hyper-Heuristics (HHs) are algorithms that search methods or have a learning mechanism for selecting or generating heuristics to solve computational search problems [9]. These techniques evidence the capacity of high-level abstraction and generalisation in tackling problems with different domains. In the last decade, HHs have received great attention, developing many good applications for real-world complex problems [10], *i.e.,* combinatorial optimisation [11], [12] and continuous optimisation [13]. However, it is hard to find many proposals in the latter domain [14]. One example of a unified framework that provides a solver for the AACP is the Customising Optimisation Metaheuristics via Hyper-Heuristic Search (CUSTOMHyS) [6]. This framework provides a methodology for extracting operators from well-known metaheuristics and multiple HH models that explore the heuristic space to generate MH. Moreover, Cruz-Duarte *et al.* reported that the framework tailors metaheuristics with good performances, getting better results than the generic MHs in several problems [13].

According to the work conducted by Cruz-Duarte *et al.* over the AACP, there are areas of opportunity to explore, as follows. Considering that it is challenging, almost impossible, to explore many valid heuristic configurations or even set fixed values to many parameters, it is evident that the MH-based approach is limited to efficiently exploring the heuristic space. Another drawback to regard 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.

Otherwise, the rising interest in Machine Learning (ML) models is not a secret. As in any other discipline, ML has also met HHs [15]–[19], where Artificial Neural Networks (ANNs) were employed to extract information about how the sequences of heuristics are selected and used for further decisions in HH models. Therefore, we propose a methodology that explores the areas of opportunity mentioned, presenting a hyper-heuristic model powered by neural networks that learn from previous results to guide the exploration of the heuristic space, providing a boosted performance in the tailored metaheuristics.

This document is organised as follows. Section II briefly describes the background information about the fundamental concepts of this research. Section III explains the proposed approach. Section IV details the methodology employed to test the experiments. Then, Section V presents an analysis of the results, doing statistical tests to make inferences with a certain confidence. Finally, Section VI wraps up the manuscript with main insights and the paths for future work.

## II. Fundamentals

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

### A. Optimisation

We define the optimisation process as the 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

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

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 kinds of problem domains, continuous and combinatorial, at different levels of abstractions. For the former, $\mathfrak{G} = \mathbb{R}^D$, where $D$ stands for the number of dimensions that the problem contains. For the combinatorial problem, we say that $\mathfrak{G} = \mathfrak{H}^{\varpi}$, where $\mathfrak{H}$ is the heuristic space and $\varpi$ is the cardinality of the sequences. Further information can be found in [6], [20].

### B. Heuristics

A heuristic can be interpreted as a sequence of actions [14]. Such a sequence may contain a single or multiple instructions that do not necessarily follow a sequential pattern [6]. According to their level of abstraction, they can be categorised into *simple heuristics*, *metaheuristics*, and *hyper-heuristics*.

First of all, since the majority of heuristics operate over a population (*i.e.,* a set of agents), like those implemented in this work, it is necessary to define it. Thus, a population consists of 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.,* $\vec{x}_*(t) = \text{arginf}\,\{f(Z(t))\}$.

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

*Simple Heuristics* (SHs) produce (initialisers, $h_i$), modify (search operators, $h_o$), or evaluate a candidate solution (finaliser, $h_f$) [6]. 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, so we only deal with heterogeneous sequences of heuristics [21].

*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 the solution of $(\mathfrak{X}, f)$ with the maximal performance $Q_{\max} = Q(\vec{h}_*|\mathfrak{X}, f)$. Consider that $Q(\vec{h}|\mathfrak{X}, f) : H \to \mathbb{R}_+$ is a metric that measures the performance of $\vec{h}$ when it is applied to the problem $(\mathfrak{X}, f)$. In this work, we *evaluate* an MH running it several times and calculate its performance using the following $Q$ metric:

$$Q(\vec{h}|\mathfrak{X}, f) = -(\mathtt{med} + \mathtt{iqr})\left(\{\forall\ \vec{x}_{r,*} \in X_* | f(\vec{x}_{r,*})\}\right),$$

where `med` and `iqr` are the median and interquartile range operators applied to a set of fitness values $f(\vec{x}_{r,*})$ obtained from implementing an arbitrary MH $\vec{h}$.

## III. Proposed Approach

This section describes the proposed model. First of all, we must remark that the model is designed for surmounting the issue of limiting its heuristics exploration by its nature. Also, it must take advantage of previous results (learning from them) to boost the performance of the tailoring process. To do so, Fig. 1 shows an overview of the model's data flow and how it interacts with the low- and high-level problem domains. This Hyper-Heuristic based on a Neural Network (HHNN) model employs a trained NN for predicting and refining the candidate unfolded metaheuristic that solves a real-value optimisation problem. Further details about this model are given below.

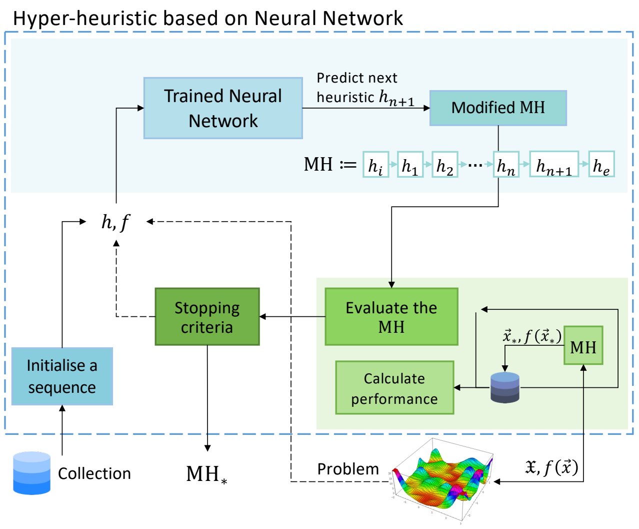
Hyper-heuristic based on Neural Network
Trained Neural Network
Predict next heuristic $h_{n+1}$
Modified MH
MH ≔ $h_i$ $h_1$ $h_2$ ⋯ $h_n$ $h_{n+1}$ $h_e$
$h, f$
Stopping criteria
Evaluate the MH
$\vec{x}_*, f(\vec{x}_*)$
MH
Calculate performance
Initialise a sequence
Collection
MH$_*$
Problem
$\mathfrak{X}, f(\vec{x})$

Fig. 1: Data flow of the proposed solution model

### A. Sequence Generation

Before going deeper, it is necessary to establish a representation for the sequences of search operators. The collection of simple heuristics is a sorted list of search operators; their name, selector type, and tuning parameters are stored per operator. The implemented framework parses each search operator's information and converts it into a function to evaluate. We associate its position in the collection as an `ID` to each search operator for a given collection. Then, to easily manipulate a sequence of search operators, we represent it as a sequence of indices of the collection, *i.e.,* a sequence of numbers. This representation has pros and cons. For example, we take advantage of this representation to manipulate the sequence with more abstract models, as the Neural Networks do. However, we need to be careful because, given two different collections, the same sequence of numbers represents different sequences of heuristics. This representation is interesting enough to interpret the sequences of heuristics as a wave, a sequence of time, or even a simple list of numbers. We take the interpretation of a time sequence.

We propose a simple sequence generation, where the basic methodology for using the HH model is described as follows:

1) Initialise an empty unfolded metaheuristic.
2) Use the HH model to select which SH to include next in the current sequence.
3) Add the chosen SH at the end of the sequence.
4) Return to Step 2 if any stopping criteria are not met; otherwise, finish the procedure.

In addition to such a procedure, it is necessary to make a few considerations. First: The hyper-heuristic model considers only the `add` modification action in comparison to other works in the literature [6], where more actions are implemented. The reason for only using this action is to have a faster and simpler sequence generation, as it only needs to apply the next search operator over the population to obtain the new performance. Second: From a collection of already generated uMHs, the proposed model can learn hidden patterns about the order of each uMH. So, prioritising those patterns seems to lead to metaheuristics with better performance. Then, the model must learn how to generate each sequence, determining heuristic by heuristic until the whole sequence is obtained.

With all that mentioned, we now explain the model, how it is implemented and trained, and a few of its possible variants.

### B. Neural Network Implementation

This work considers using an Artificial Neural Network (ANN) model by its prediction capabilities over a sequence of numbers. The assigned task for ANN is to find the probability distribution to choose a simple heuristic based on its contribution to the whole metaheuristic search procedure. In layman terms, an ANN determines a probability distribution over the search operator collection. Hence, the performance of a simple heuristic in a uMH depends on the current candidate solutions, which the previously applied heuristics have modified. Fig. 2 illustrates the behaviour mentioned above of an ANN when interacting with a sequence of heuristics for a particular problem domain. Therefore, the hyper-heuristic model can choose a heuristic based on a probability distribution. In implementing the model, the heuristic is accepted if it enhances the performance of the sequence. Now, suppose the same heuristic is selected several times, and the performance is not improved. In that case, it is added to a Tabu list to ignore it in further selections, forcing the model to choose another heuristic. This Tabu list is cleaned once a heuristic is accepted; *i.e.,* the model can consider all the heuristics again.

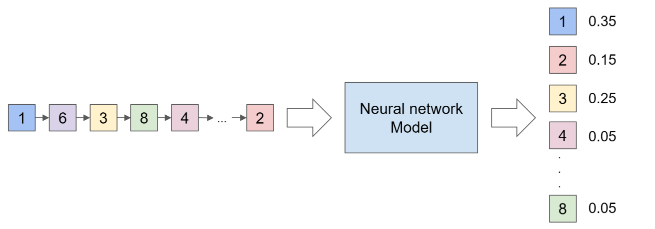

Fig. 2: Diagram representation of the overall operation of the Neural Network. In this case, for a given sequence, the model predicts the most probable simple heuristic to apply.

## C. Neural Network Architecture

The architecture considered for the Neural Network model is the Long Short-Term Memory (LSTM) because of its capability of learning from sequences of numbers. Recall that an unfolded metaheuristic can be interpreted as a discrete-time signal, where each information position corresponds to a search operator interacting directly with the low-level problem. We keep simple the LSTM configuration for the experiments to obtain preliminary results that allow us to compare the methodology against previous approaches. So, it is essential to mention that the input layer has $M$ neurons, corresponding to the maximum number of heuristics from the sequences considered to predict the next heuristic. It is worth mentioning that, using Ragged Tensors from TensorFlow [22], we implement one experiment that uses a model that allows accepting sequences with variable length, avoiding the application of an *identity encoder*. Subsequently, we consider only one hidden-layer with 20 neurons and a `Sigmoid` activation function. The output layer has $|\mathfrak{H}|$ neurons and a `Soft-Max` activation function. This configuration allows us to interpret the output as the distribution of search operators that we desired.

## D. Training Dataset

As mentioned before, one possible approach is based on training the model to generate each sequence, prioritising those sequences with better performance. We consider that not all uMHs have the same length, *i.e.,* there are ones that require fewer iterations than others for providing an optimal solution due to the nature of their search operators.

Keeping this in mind, it is possible to generate a dataset for training the Neural Network using a given set of search operator sequences. The procedure to obtain such a training dataset is quite simple. First, consider all its prefixes, including the empty prefix, and exclude the prefix corresponding to the whole sequence for a given sequence. Then, each prefix can be associated with the heuristic that is next in the sequence, as Fig. 3 depicts. Such a procedure transforms a sample of uMHs into a training dataset.

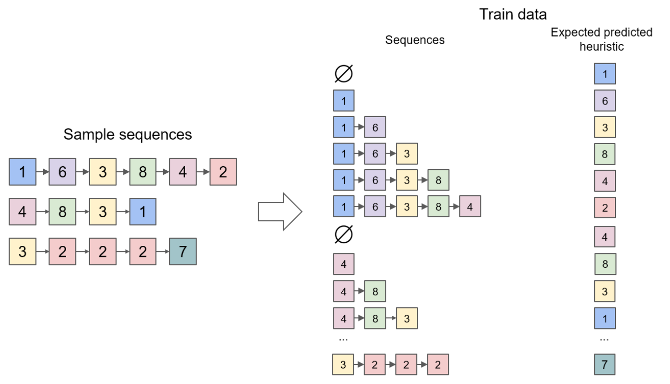

Fig. 3: Illustrative example about how to get the training dataset from sample sequences.

However, an issue circumvents the NN model: The input data should be a fixed-length vector. To surmount it, the length of the sequences accepted by the Neural Network is fixed to a number $M$, for instance. This value $M$ represents the maximum number of elements considered from a sequence to predict the following heuristic; *i.e.,* memory sequence. If the sequence's length $\varpi$ (cardinality) is greater than $M$, this sequence is cut off considering only the last $M$ heuristics. If $\varpi$ is less than $M$, the sequence is padded with dummy values that have no relation with any possible value in the sequence. Later on, we call the procedure that processes a sequence to convert it into a valid input for the Neural Network as *identity encoder*. This encoder preserves the values of uMHs, just adapting the length according to the rules described.

## IV. Methodology

In this work, we used Python 3.8.5 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 the experiments. The implementation of the proposed model is integrated into the open-access framework CUSTOMHyS v1.1. For fully detailed documentation of the version v1.0, please review the previous manuscript published by Cruz-Duarte *et al.* [23]. The development of this new framework version can be found on https://github.com/jose-tapia/nn-hh-umhs-cec22, accessed on May 23th, 2022.

We specified the domains at the two levels of abstraction considered to test and implement the proposed model. At a lower level, a total of 107 benchmark functions were selected as continuous optimisation problem domains. For further information of these benchmark functions, please review the documentation of CUSTOMHyS [23]. The dimensions selected were $2, 10, 30$, and $50$, resulting in 428 different problems. To analyse these results, the problems were categorised by their dimensionality. Given a benchmark function, the domain of the metaheuristics was the set of candidate solutions for such a low-level problem.

Moreover, a collection of population-based heuristics was used as the high-level domain for the hyper-heuristic models.

We extracted these heuristics from ten well-known metaheuristics, such as Random Search, Simulated Annealing, Genetic Algorithm, Cuckoo Search, Differential Evolution, Particle Swarm Optimisation, Firefly Algorithm, Stochastic Spiral Optimisation Algorithm, Central Force Optimisation, and Gravitational Search Algorithm. So, we utilised a collection that contains 205 search operators, obtained from varying the hyper-parameters of the simple heuristics previously extracted and combining with the selectors available. We also considered using a population of 30 agents for each population-based unfolded metaheuristic.

For a given experiment, we carried out the following process per each benchmark function:

1) We generated 100 uMHs with a Random Search using the population-based heuristic domain to solve the benchmark function. Each uMH has a cardinality up to 100.
2) We converted these metaheuristics into a training dataset with the procedure described in Section III-D.
3) We trained the Neural Network with 100 epochs. For that, we used the Categorical Cross-Entropy as the loss function, Adam with a learning rate of 0.001 as the optimiser, and Accuracy as the metric.
4) We generated 100 uMHs using the proposed HHNN model. For that reason, we implemented the maximum cardinality number of 100 and a saturation scheme related with the Tabu list as stopping criteria. So, if the model tries to add 50 different search operators and none improves the performance, it stops the HH search. Plus, we consider including a search operator to the Tabu list after applying such an operator five times.
5) We determined the performance of the 100 uMHs and saved them for posterior analysis.

Last, and definitively not least, we carried out eleven experiments. The difference amongst these experiments is the maximum number of elements considered to predict the next search operator to include in the metaheuristic. The Neural Network accepted sequences with variable length as input in the first experiment. That is why we refer to this experiment as `LSTM_variable_lenght` or `LSTM_vl`. So, this implementation allowed sub-sequences with a length between 1 and 100, an adjustable range according to the practitioner necessities. For the other ten experiments, we established the maximum value $M$ to 10, 20, …, and 100, thus we refer to each experiment as `LSTM_10`, `LSTM_20`, …, `LSTM_100`.

Finally, we considered 66 implementations obtained from the ten well-known MHs previously mentioned using the recommended parameters on the state-of-the-art applications of these MHs to benchmark our experiments for comparison purposes. Each metaheuristic in this prior implementation has employed 30 agents and was limited to 100 generations. We selected that number of generations as we can see them as unfolded MHs with cardinality ranging from 100 to 300.

## V. Results

Fig. 4 shows the results for the first configuration `LSTM_10` of the proposed methodology, which uses an ANN architecture of an LSTM with a memory of 10 search operators. The percentages indicate that the proposed approach outperforms the results obtained from the basic metaheuristics in most cases. It is noticeable that the lowest rate of 83% presented at two-dimensional problems. As an additional insight, we attach the percentage of cases where the proposed approach outperforms against the sequences of heuristics used for training, achieved from Random Search, proving that the methodology is helpful to enhance the HH search. These results support our hypothesis that the proposed methodology can enhance the search in the heuristic space, generating metaheuristics with better performance.

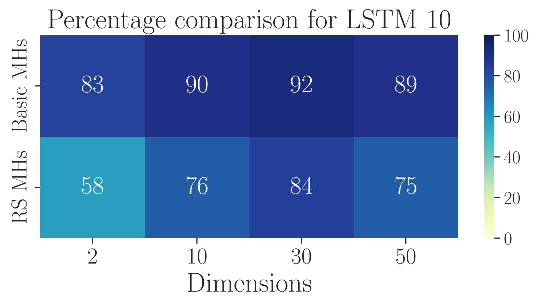

Fig. 4: Percentage of cases where the experiment `LSTM_10` outperform the basic MHs and Random Search MHs.

These results show thrilling percentages that let us overview the general behaviour from the proposed approach, but it is possible to extract detailed information from this comparison. A natural follow-up is to ask ourselves if the proposed method outperforms each basic metaheuristic. So, instead of using the percentage of cases where a method outperforms the others, we consider using the pairwise (one-sided) Wilcoxon signed-rank test, according to [24], [25]. Let us say that an “experiment `A`” is the performance values of the metaheuristics generated by the hyper-heuristic model using the configuration `A`. Then, we establish the null and alternative hypotheses for Wilcoxon's tests as follows:

**Hypothesis 0** (**$H_0$**: *Null*)**.** The experiment or metaheuristic `A` performs equal to or worse than the experiment or metaheuristic `B` with a significance level of $\alpha$.

**Hypothesis 1** (**$H_a$**: *Alternative*)**.** The experiment or metaheuristic `A` outperforms experiment or metaheuristic `B` with a significance level of $\alpha$.

Bearing this information in mind, we perform Wilcoxon's test between the experiment `LSTM_10` and every basic metaheuristic with a significance level of $\alpha = 0.05$. Fig. 5 shows the $p$-values from this Wilcoxon's analysis. Observe that all but one of the MHs lead to a rejection of the null hypothesis (**$H_0$**). Thus, with a significance level of $\alpha = 0.05$, we validate that the experiment `LSTM_10` produces enhanced metaheuristics concerning all the considered basic MHs but one. The basic MH that fails to reject **$H_0$** is an implementation of the Particle Swarm Optimisation algorithm. However, this is a particular case, and naturally, this MH does not preserve such performance in all problems. We consider that obtaining outperforming results with this first experiment against 65 of 66 basic metaheuristics is an excellent step towards our goal.

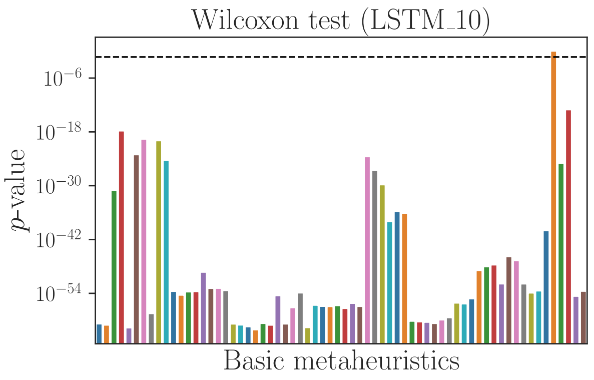

Fig. 5: Wilcoxon's analysis results from comparing the experiment `LSTM_10` against the basic metaheuristics.

Moreover, we perform Wilcoxon's test over the experiment `LSTM_10` and the Random Search MHs with a significance level of $\alpha = 0.05$, intending to provide more evidence that the proposed method produces enhanced metaheuristics. We find in this comparison results with a $p$-value of $1.22 \times 10^{-10}$, that is less than $0.05$, allowing us to reject the null hypothesis. With such evidence, we can ensure that the proposed methodology allows us to enhance the exploration of the heuristic space rather than doing a Random Search.

The initial analysis shows us a glance at how the proposed model produces excellent results using the configuration `LSTM_10`. We now proceed to analyse the performance of the tested configurations described in the methodology. The principal difference between these experiments is the memory length, so we aim to find evidence about which one generates better metaheuristics.

We compute the percentage of cases where each experiment obtains the best performance. Fig. 6 depicts a summary of these percentages in four categories, one per dimension. If there is a tie, such a case counts for all the experiments with the best performance. We observe that, in each dimension, an experiment outperforms in comparison to the other. For instance, in 2D, `LSTM_80` presents the highest percentage with almost 14%. For 10D, `LSTM_70` and `LSTM_80` reach the higher percentages; but not so far from `LSTM_100`. In 30D, it is noticeable the difference between `LSTM_variable_length`, where it obtained over 14% overall cases, more than 3% of the second place achieved by `LSTM_50`. Lastly, in 50D, `LSTM_variable_length` keeps dominating the scenario but no so far from the second place that is `LSTM_90`.

As it is illustrated in Fig. 6, each dimension has different behaviour. It is not so easy to conclude that there is a tendency of which memory would be better for the LSTM architecture. For two and ten dimensions, the tendency is for a memory close to 80 units. For 30D problems, the tendency is near 50. Moreover, for 50D, memory falls between 50 and 100.

The continuous optimisation problems can be categorised according to their mathematical attributes. Two of them are

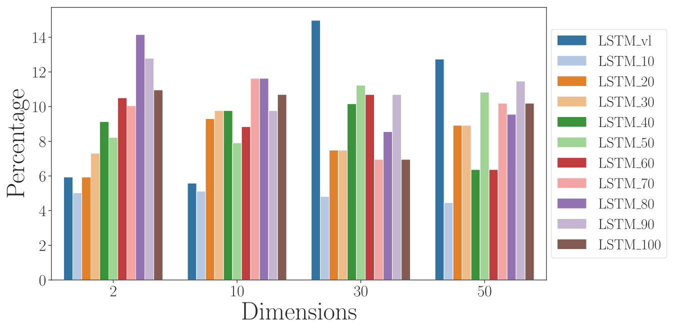

Fig. 6: Comparison of different experiments (models) as a function of dimensions. The percentage of each experiment represents for how many problems the given model outperforms the other configurations.

of particular interest to us, as they help us classify the continuous optimisation problems with more precision, such as *unimodality*, for those real-valued functions that only have one local (thus, global as well) optimum value, and *differentiability*, that is derivable in every point of its domain. The hardness of a continuous optimisation problem can be related to mathematical characteristics. For example, for a unimodal and differential problem, the search operator `gradient descent` would find the optimal value in a few tries; *i.e.,* exactly one if it is a quadratic function.

As expected, Fig. 7 highlights further information about the behaviour of the experiments along the dimensions and the possible mathematical feature combinations. We categorise the problems into 16 categories according to their mathematical attributes and dimension. In this figure, each plot contains a title that indicates which category corresponds to each category according to the format: `Cat = XY, Dim = D`, where `XY` indicates a binary encoding for the *unimodality* and *differentiability* features, respectively. For example, `01` corresponds to a multimodal and differentiable problem. Moreover, `D` represents the dimensionality for such a problem set. It is worth mentioning that the number of continuous optimisation problems that belong to `Cat = XY`, for any `XY`, is the same for all the dimensions. Plus, looking at Fig. 7, this could be contradictory as it seems that the number of low-level problems decreases as the number of dimensions increases, in particular for `Cat = 01`. We expected this behaviour; if tied, the case counts for all the experiments involved. Then, for lower dimensions, it is more common to have ties.

The experiment that uses variable-length sequences for training presents a poor performance for lower dimensions, in comparison to higher ones where it beats the other experiments for the majority of the categories, as is illustrated in Fig. 7. Nevertheless, the other LSTM configurations exhibit better performance, especially for memory sizes between 50 and 80 units.

Appreciate that, the last place in any category illustrated at Fig. 6 and Fig. 7 is for the experiment `LSTM_10`. Such a configuration already shows evidence that it outperforms results against the 66 basic MHs and Random Search MHs. Since the other experiments seem to have better results, we

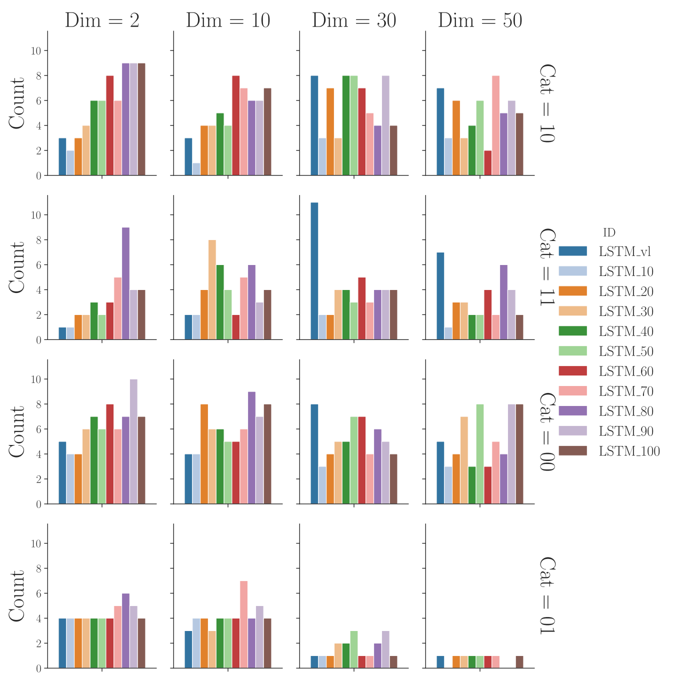

Fig. 7: Comparison of different experiments along the dimensions and four mathematical characteristic combinations. The count of each experiment represents how many problems it achieved better performance than the others.

think all configurations can easily beat the basic MHs and Random Search MHs. In the previous figures, we contribute graphical support to compare the experiments. To provide statistical evidence, we perform the Wilcoxon's test described at the initial analysis to make a pairwise comparison among all the configurations (Fig. 8).

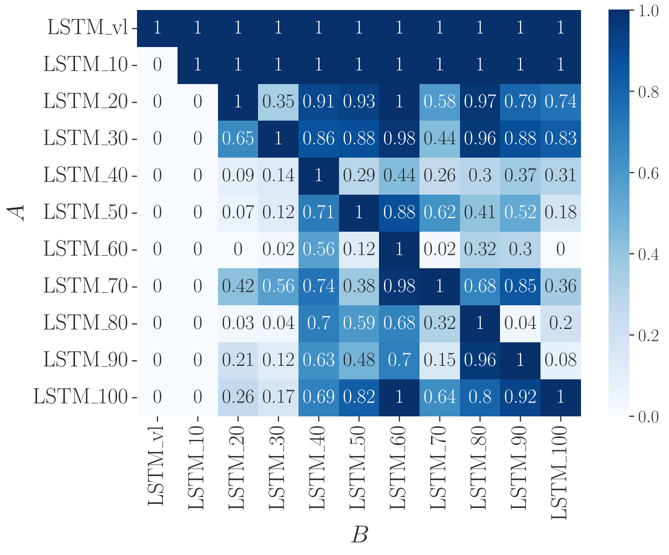

Fig. 8: $p$-values from a Wilcoxon's analysis across all pairs of configurations. The null and alternative hypotheses are described in the analysis of the initial results.

Bear in mind that in Fig. 8, the $y$-axis represents experiment A and the $x$-axis represents experiment B. Notice that, although configuration `LSTM_variable_length` obtained a great percentage at 30D and 50D (Fig. 6), there is not enough evidence to reject the null hypothesis when compared to all configurations. The experiment `LSTM_10` proves to only outperform `LSTM_variable_length`.

Considering a significance level of 0.05, the experiment that rejects the null hypothesis in the majority of the cases is the one with configuration `LSTM_60`, which outperforms `variable-length` and memories of 10, 20, 30, 70, and 100 search operators. Although, there is not enough evidence to establish if it outperforms the remaining experiments. Then, we conclude that these experiments have at most the same performance.

It is worth noticing that we have conclusions related to all the interactions that could have the experiment `LSMT_60`; in the majority of the cases, it performs better; in all the other cases, it performs at least equal to them. Then, we can say that the experiment `LSTM_60` is the ideal choice among the presented configurations.

Even if experiment `LSTM_60` seems to have the best performance, all experiments outperform the 66 basic MHs and Random Search MHs, as Fig. 9 illustrates. It is possible to observe that all experiments obtain better performance scores in the majority of cases, with the worst performance at `LSTM_variable_length`; this partial conclusion is coherent with the statistical analysis.

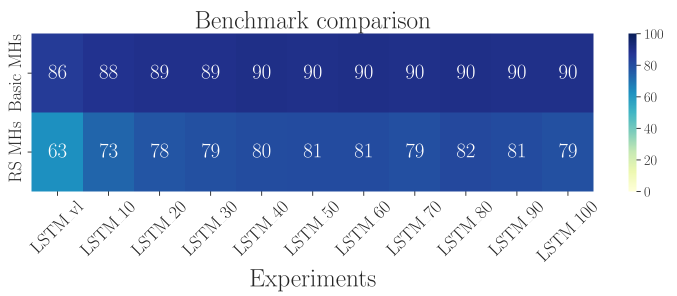

Fig. 9: Percentage of cases where each experiment outperform the Basic MHs and Random Search MHs (RS MHs).

## VI. Conclusions

This work presented a new heuristic-based solver model based on Artificial Neural Networks for continuous optimisation problems, which can easily extend to arbitrary domains. When comparing against the previous solvers implemented, the proposed model can refine and predict which heuristics to consider at the end of a sequence of heuristics to enhance its performance, changing the paradigm used in the other solvers. Hence, this implies a contribution to the Automated Algorithm Composition Problem (AACP) with a new solver that tailors metaheuristics that tackle a specific optimisation problem.

Besides implementing the proposed model, we contribute to conceptualising ideas that one can use to integrate other Machine Learning models and a methodology to generate training data for them. We provided statistical evidence that supports that our proposed approach can be a great alternative solver to produce solutions for given continuous optimisation problems compared to basic metaheuristics. Nevertheless, the experiment `LSTM_60` highlighted over them, proving that we can tune the configuration of a neural network to enhance its performance. In this work, we introduced an innovative methodology that was able to show an improvement in performance concerning the previous solvers. We are aware that we introduced very few configurations of the proposed model. However, as a primary analysis, we consider that these configurations provide enough evidence to prove our hypothesis, helping us to indicate which direction to follow next to improve our performance. However, we should mention that there are many areas worth exploring. A few of them that we have in mind are:

- Adopt a different strategy to generate the training data. So, we need to consider the metaheuristics generated by any model or human mind instead of those generated by Random Search.
- Consider using a different collection of search operators to explore the heuristic space. Even if this collection has a considerable diversity of search operators, we can explore other sets that can help us identify model capabilities to learn patterns.
- Explore diverse configurations for the neural networks. In this work, we only regarded the same and unique hidden layer for all the models, so we plan to consider more hidden layers and different encoders to use over the sequences. Several possible options are one-hot encoding, auto-encoders, and adding features extracted from the continuous functions.
- We can easily extend the proposed model to other models, such as Multi-Layer Perceptron, Recurrent Neural Networks, or one of the recent hot-topics, Transformers [26].

Furthermore, we presented an analysis of the model's performance. Still, it is necessary to analyse the proposed model's time and memory complexities to provide evidence of its feasibility over the traditional metaheuristics and the other approaches. Plus, we consider adding further benchmark problems, including those utilised in algorithm competitions, *e.g.,* CEC and GECCO.

## References

[1] A. E. Ezugwu, A. K. Shukla, R. Nath, A. A. Akinyelu, J. O. Agushaka, H. Chiroma, and P. K. Muhuri, "Metaheuristics: a comprehensive overview and classification along with bibliometric analysis," *Artif. Intell. Rev.*, vol. 54, no. 6, pp. 4237–4316, 2021.

[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] F. Campelo and C. Aranha, "Sharks, zombies and volleyball: Lessons from the evolutionary computation bestiary," 2021.

[4] H. Stegherr and J. Hähner, "Analysing metaheuristic components," 2021.

[5] J. de Armas, E. Lalla-Ruiz, S. L. Tilahun, and S. Voß, "Similarity in metaheuristics: a gentle step towards a comparison methodology," *Natural Computing*, pp. 1–23, 2021.

[6] 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.

[7] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," *IEEE transactions on evolutionary computation*, vol. 1, no. 1, pp. 67–82, 1997.

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

[9] 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.

[10] K. Chakhlevitch and P. Cowling, "Hyperheuristics: recent developments," in *Adaptive and multilevel metaheuristics*, pp. 3–29, Springer, 2008.

[11] I. Amaya, J. C. Ortiz-Bayliss, A. Rosales-Perez, A. E. Gutierrez-Rodriguez, S. E. Conant-Pablos, H. Terashima-Marin, and C. A. C. Coello, "Enhancing selection hyper-heuristics via feature transformations," *IEEE Computational Intelligence Magazine*, vol. 13, no. 2, pp. 30–41, 2018.

[12] 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.

[13] 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 *2020 IEEE Congress on Evolutionary Computation (CEC)*, pp. 1–8, IEEE, 2020.

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

[15] J. Li, E. K. Burke, and R. Qu, "Integrating neural networks and logistic regression to underpin hyper-heuristic search," *Knowledge-Based Systems*, vol. 24, no. 2, pp. 322–330, 2011.

[16] R. Tyasnurita, E. Özcan, A. Shahriar, and R. John, "Improving performance of a hyper-heuristic using a multilayer perceptron for vehicle routing," *15th UK Workshop on Computational Intelligence*, 2015.

[17] J. C. Ortiz-Bayliss, H. Terashima-Marín, and S. E. Conant-Pablos, "A neuro-evolutionary hyper-heuristic approach for constraint satisfaction problems," *Cognitive Computation*, vol. 8, no. 3, pp. 429–441, 2016.

[18] E. Lara-Cárdenas, X. Sánchez-Díaz, I. Amaya, and J. C. Ortiz-Bayliss, "Improving hyper-heuristic performance for job shop scheduling problems using neural networks," in *Mexican International Conference on Artificial Intelligence*, pp. 150–161, Springer, 2019.

[19] W. B. Yates and E. C. Keedwell, "Offline learning for selection hyper-heuristics with elman networks," in *International Conference on Artificial Evolution (Evolution Artificielle)*, pp. 217–230, Springer, 2017.

[20] 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*, p. 100935, 2021.

[21] 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, jun 2021.

[22] L. Moroney, "Introducing ragged tensors." https://blog.tensorflow.org/2018/12/introducing-ragged-tensors.html, 2018. Accessed: 2022-02-11.

[23] 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.

[24] J. Carrasco, S. García, M. Rueda, S. Das, and F. Herrera, "Recent trends in the use of statistical tests for comparing swarm and evolutionary computing algorithms: Practical guidelines and a critical review," *Swarm and Evolutionary Computation*, vol. 54, p. 100665, may 2020.

[25] S. García, D. Molina, M. Lozano, and F. Herrera, "A study on the use of non-parametric tests for analyzing the evolutionary algorithms' behaviour: A case study on the CEC'2005 Special Session on Real Parameter Optimization," *Journal of Heuristics*, vol. 15, no. 6, pp. 617–644, 2009.

[26] T. Wolf, J. Chaumond, L. Debut, V. Sanh, C. Delangue, A. Moi, P. Cistac, M. Funtowicz, J. Davison, S. Shleifer, *et al.*, "Transformers: State-of-the-art natural language processing," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 38–45, 2020.