applied sciences

MDPI

Article

A Feature-Independent Hyper-Heuristic Approach for Solving the Knapsack Problem

Xavier Sánchez-Díaz, José Carlos Ortiz-Bayliss *, Ivan Amaya, Jorge M. Cruz-Duarte, Santiago Enrique Conant-Pablos and Hugo Terashima-Marín

Tecnologico de Monterrey, School of Engineering and Sciences, Monterrey 64849, Mexico; sax@tec.mx (X.S.-D.); iamaya2@tec.mx (I.A.); jorge.cruz@tec.mx (J.M.C.-D.); sconant@tec.mx (S.E.C.-P.); terashima@tec.mx (H.T.-M.)

* Correspondence: jcobayliss@tec.mx

Abstract: Recent years have witnessed a growing interest in automatic learning mechanisms and applications. The concept of hyper-heuristics, algorithms that either select among existing algorithms or generate new ones, holds high relevance in this matter. Current research suggests that, under certain circumstances, hyper-heuristics outperform single heuristics when evaluated in isolation. When hyper-heuristics are selected among existing algorithms, they map problem states into suitable solvers. Unfortunately, identifying the features that accurately describe the problem state—and thus allow for a proper mapping—requires plenty of domain-specific knowledge, which is not always available. This work proposes a simple yet effective hyper-heuristic model that does not rely on problem features to produce such a mapping. The model defines a fixed sequence of heuristics that improves the solving process of knapsack problems. This research comprises an analysis of feature-independent hyper-heuristic performance under different learning conditions and different problem sets.

Keywords: hyper-heuristics; knapsack problem; optimization

Citation: Sánchez-Díaz, X. Ortiz-Bayliss, J.C.; Amaya, I.; Cruz-Duarte, J.M.; Conant-Pablos, S.E.; Terashima-Marín, H. A Feature-Independent Hyper-Heuristic Approach for Solving the Knapsack Problem. *Appl. Sci.* **2021**, *11*, 10209. https://doi.org/10.3390/app112110209

Academic Editor:Peng-Yeng Yin

Received: 24 September 2021
Accepted: 26 October 2021
Published: 31 October 2021

1. Introduction

The intersection between combinatorial optimization and Hyper-Heuristics (HHs) is a relevant and active area in literature, as Sánchez et al. detailed with their thorough systematic review [1]. The former considers optimization problems where a permutation of feasible values gives candidate solutions. The hardness of these problems could be easy or hard to solve, depending on different parameters. Among them resides a category known as NP-hard problems, for which analytical solvers cannot be scaled due to an excessive computational burden. Therefore, approximate solvers are commonly sought for NP-hard problems, including those based on heuristics. It is here where HHs shine bright. This approach has been deemed as high-level heuristics useful to tackle hard-to-solve problems [2], particularly NP-hard ones [3]. A classification of HHs considers two main groups: selection HHs (those that select heuristics from an available set) and generation HHs (those that create new heuristics using components of existing ones) [4]. Although both groups have proved of great interest to the scientific community, in this work we focus on the former.

Selection HHs deal with problems indirectly. They browse a set of available heuristics, which are selectively applied to solve the problem at hand [5]. A selection HH analyzes a set of available heuristics and chooses the most suitable one according to a given performance metric. Most of the current selection HH models include two key phases: heuristic selection and move acceptance [6]. The former represents the strategy for deciding which heuristic should be selected. Conversely, the latter determines whether the new solution is accepted or discarded. The approach proposed in this work simplifies the overall model by only focusing on heuristic selection. Thus, changes resulting from applying a particular heuristic are always accepted.

Evolutionary computation is a recurrent approach among the many learning methods employed in the literature to produce HHs. Some examples include, but are not limited to, Genetic Programming (GP) [5,7–9], Grammatical Evolution (GE) [10], Genetic Algorithms (GA) [11,12], and Artificial Immune Systems (AIS) [13,14]. The literature contains other related proposals, such as those that evolve HHs by analyzing the set of problem instances [15]. These findings support the idea of using an evolutionary strategy as a learning mechanism for HHs, one which improves their performance via crossover or mutation. This work focuses on the latter.

HHs have been used to tackle packing problems in the past. Hart and Sim describe a variant of AIS used as a hybrid HH for the Bin Packing Problem [13,14,16]. Falkenauer [11] proposed a GA-based HH model, and Burke et al. [17], Hyde [8], and Drake et al. [9,18,19] have studied GP rules for the Bin Packing and Multidimensional Knapsack problems. More recent studies of HHs have been conducted on the binary knapsack domain using Ant Colony Optimization [20] and Quartile-related Information [21]. Further information about the state-of-the-art of HHs and their applications are provided in [4,22].

As mentioned above, several combinatorial optimization problems have been tackled with HHs [1]. This paper focuses on the Knapsack Problem (KP), which has been studied in great depth due to its simple structure and broad applicability. Example applications include cargo loading, cutting stock, allocation, and cryptography [23]. When a constructive approach is used to solve this problem, the solution is built one step at a time by deciding if one particular item must be packed or ignored. For simplicity, our setting states that once the process chooses an item, there is no way to change such a decision. Using a constructive approach leads to different subproblems throughout the solving process, depending on the heuristic used. This happens because packing an item produces an instance with a reduced knapsack capacity (the previous knapsack capacity minus the weight of the packed item) and a reduced list of items (the item recently packed or ignored is no longer part of the items to pack). This behavior raises the question of which heuristic, among the different options, should be used to maximize the overall profit resulting from the items contained in the knapsack. The literature usually refers to the problem of selecting the best algorithm for a particular situation as the Algorithm Selection Problem [24].

Despite the extensive use of selection HHs [4,25], only a few works explore the insights of their behavior. A few examples include a run-time analysis [26,27], the use and control of crossover operators [19], and heuristic interaction when applied to constraint satisfaction problems [28]. Furthermore, traditional selection HH models, that represent the relation between problem states and heuristics through ⟨condition, action⟩ rules, exhibit a significant limitation: they require the definition of a set of features to characterize the problem state [29]. Finding such a set of features implies an additional layer of complexity to the model. To the best of our knowledge, there is only one work on feature-independent HHs for the knapsack problem, which obtained only a few preliminary results [30]. Therefore, our work aims at filling this knowledge gap through two particular contributions:

- It proposes an evolutionary-powered hyper-heuristic framework capable of combining the strengths of individual heuristics when solving sets of KP instances.
- Besides the model itself, it provides the analysis and understanding of the performance of the hyper-heuristics designed under this model on instance sets formed by challenging instances, both balanced and unbalanced (in terms of heuristic performance).

The remainder of this document is organized as follows. Section 2 defines the fundamental ideas associated with our work. Section 3 describes the HH model and the rationale behind it. Section 4 details the experiments conducted and analyzes the results. Finally, Section 5 presents the conclusions and provides an overview of future directions for this investigation.

2. Background

This section introduces several basic concepts utilized in this work. It starts by describing the fundamental Knapsack Problem (KP). Then, it presents heuristics in general. Later on, it explains the basic foundations of Evolutionary Algorithms (EAs).

2.1. Knapsack Problem

In layman's terms, a knapsack problem seeks to store a set of items into a knapsack with limited capacity. Therefore, one must choose a subset of the items based on some criteria. For example, the profit or weight to select the items that must go into the knapsack. In formal terms, let $\vec{x} \in \mathbb{Z}_2^n$ be a binary-valued vector of n items, where each element $x_i \in \vec{x}$ represents an item and its selection according to its position and value, respectively, for example, if there is a stock of three items and we choose the second one, we have $\vec{x} = (0, 1, 0)^\intercal$. Likewise, let $\vec{p} \in \mathbb{R}_+^n$ and $\vec{w} \in \mathbb{R}_+^n$ be the profit and weight vectors directly related to the items, i.e., the i-th item has profit $p_i \in \vec{p}$ and weight $w_i \in \vec{w}$. Thus, this problem (widely known as the 0/1 Knapsack Problem) can be defined as

$$\begin{aligned} \vec{x}_* &= \underset{\vec{x}}{\operatorname{argmax}}\{\vec{p} \cdot \vec{x}\}, \\ \text{s.t.} \quad & \vec{w} \cdot \vec{x} \leq c, \\ \text{with} \quad & \vec{x} \in \mathbb{Z}_2^n \quad \text{and} \quad \vec{p}, \vec{w} \in \mathbb{R}_+^n \end{aligned} \tag{1}$$

where $c \in \mathbb{R}_+$ is the total capacity of the knapsack.

Although many solving strategies can be found in the literature, the initial solvers for the knapsack problem were initially divided into two categories: exact methods and approximate ones. Exact methods provide optimal solutions but have limited applications due to their consumption of computational resources [31,32]. Conversely, approximated methods make some assumptions to simplify the solving process to give usually suboptimal solutions [33].

However, the advance in computing power and creativity has lead to extensive diversity of techniques that can be found nowadays, most of them hybrids. To provide a glance at such diversity, we mention some representative works. For example, Genetic Algorithms (GA) and Particle Swarm Optimization (PSO) have been used for solving the multidimensional KP [34,35]. Gómez et al. proposed a Binary Particle Swarm strategy with a genetic operator, also for the multidimensional KP [36]. Another example of recent solving methods includes the one by Razavi and Sajedi, where the authors proposed using Gravitational Search (GSA) for solving the 0/1 KP [37]. Inspired by Quantum Computing, Patvardhan et al. proposed a novel method to solve the KP by using a Quantum-Inspired Evolutionary Algorithm [38]. Modifications to apparently simple ideas can also be found nowadays. For example, Lu et al. solve the 0/1 KP by using Greedy Degree and Expectation Efficiency (GDEE) [39]. Some probabilistic models include Cohort Intelligence (CI) [40] and hybrid heuristics. For example, by using Mean Field Theory [41], Banda et al. generated a probabilistic model capable of replacing a complex distribution of items with an easier one.

The literature on the KP includes a wide variety of solution methods. For a thorough treatment, we refer the reader to the textbooks found in [42,43]. Broadly speaking, the solution methods are of three types: exact methods, heuristic methods, and approximation algorithms (which are special heuristic methods that yield solutions of a guaranteed quality). The best exact methods are now capable of solving instances with thousands of items to proven optimality in reasonable computing times [32,44–46]. However, it is possible to devise instances that are very challenging for exact methods [45]. For this reason, heuristics remain of interest.

2.2. Heuristics

A heuristic is a procedure that creates or modifies a candidate solution for a given problem instance. There are many classifications of heuristics in the literature. Most of them

relate to combinatorial optimization domains [47]. In this work, we use the term 'heuristic' to refer to the low-level operations to apply to a problem instance [4]. An illustrative example is a specific way to organize a knapsack based on item size. In mathematical terms, a heuristic $h : \mathfrak{X} \mapsto \mathfrak{X}$ applied to an instance problem $\mathfrak{X}$, may use the current candidate solution $\vec{x} \in \mathfrak{X}$, and delivers a new candidate $\vec{x}'$ as shown,

$$\vec{x}' = h\{\vec{x}\}. \tag{2}$$

Consider that if $\vec{x} = \emptyset$, the heuristic creates a candidate solution; otherwise, it modifies the current one.

A HH is described as a high-level strategy that controls heuristics in the process of solving an optimization problem [47]. Therefore, HHs move in the heuristic space to find a heuristic configuration that solves a given problem. With that in mind, a HH can be defined as follows [4]. Let $\boldsymbol{h} \in \mathfrak{H}^{\varpi}$ be a heuristic sequence from the heuristic space $\mathfrak{H}$, and let $F(\boldsymbol{h}|\mathfrak{X}) : \mathfrak{H}^{\varpi} \times \mathfrak{X} \mapsto \mathbb{R}$ be its performance measure function. Here, $\mathfrak{X}$ is the problem domain in an optimization problem with an objective function $f(\vec{x}) : \mathfrak{X} \mapsto \mathbb{R}$. For the particular case of the knapsack problem, $\mathfrak{X} = \mathbb{Z}_2^n$. Then, a solution $\vec{x}_* \in \mathfrak{X}$ and its corresponding fitness value $f(\vec{x}_*)$ are found when a $\boldsymbol{h}$ is applied on $\mathfrak{X}$, so its performance $F(\boldsymbol{h}|\mathfrak{X})$ can also be determined. Therefore, let a HH be a technique that solves

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

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

It is essential to mention that there also exists something we might categorize as mid-level heuristics: Metaheuristics (MHs). These methods are trendy because of their proven performance on different scenarios and applications [48]. MHs are defined as master strategies that control simple heuristics [49]. Therefore, by finding a middle point between definitions for low and high-level heuristics, it is possible to say that an MH corresponds to a heuristic sequence $\boldsymbol{h}$ applied recurrently until reaching the desired performance. This idea has been extended by Cruz-Duarte et al. [50,51]. Thus, for a given optimization problem, we can define the process of approximating the optimal solution $\vec{x}_* \in \mathfrak{X}$ via an MH such as

$$\vec{x}_* \triangleq \prod_{h_k \in \boldsymbol{h}}^{h_f} h_k\{\mathfrak{X}\} \tag{4}$$

where h_f is also an operator to evaluate stopping criteria, and Π is the iteral operator based on that in [52]. This operator indicates a recurrent application of heuristics from a sequence $\boldsymbol{h}$ until h_f marks the halt. For example, for a MH defined with two operations ($\varpi = 2$), say crossover (h_1) and mutation (h_2), thence $\boldsymbol{h} = \{h_1, h_2\}$. Consider h_f as a common fitness tolerance criterion such as $f(\vec{x}) \leq \varepsilon$. Therefore, the metaheuristic applies first h_1, followed by h_2, and then it checks the condition given by h_f. If such a condition is not met, the process is repeated until it complies.

2.3. Evolutionary Algorithms (EAs)

EAs are a particular subgroup of the population-based metaheuristics inspired by evolution and biological processes observed in nature [53]. The most notable examples of these methods are Differential Evolution (DE) [54] and Genetic Algorithms (GAs) [55]. In a general sense, the individuals of an EA interact with each other to explore the problem domain and exploit the candidate solutions, i.e., they evolve. Such evolution is possible due to operators such as selection, crossover, mutation, and reproduction. It is easy to notice that these operators are just (low-level) heuristics. In this work, we chiefly employ mutation-based operations, which are detailed in the next section.

3. Proposed Hyper-Heuristic Model

The HH model developed in this work can be classified as an offline selection HH [56]. Internally, the HH model relies on a variant of the $(1+1)$ Evolutionary Algorithm (EA) to find the sequence of heuristics to apply. The original EA flipped each bit with probability $1/\varpi$ (where ϖ is the chromosome length). Conversely, our approach chooses one among various available mutation operators based on a uniform random probability distribution. This EA implementation considers some of the features used by Lehre and Özcan [27]. However, the set of available operators is different as we work with constructive heuristics while they have used perturbative ones.

We choose four simple packing heuristics due to their popularity: $\mathfrak{H} = \{\texttt{Def}, \texttt{MaxP}, \texttt{MaxPW}, \texttt{MinW}\}$. Before moving forward, we describe them below.

Default (`Def`) packs the items in the same order that they are contained in the instance, as long as they fit in the knapsack.

Maximum profit (`MaxP`) sorts the items in descending order based on their profit and packs them in that order as long as they fit in the knapsack.

Maximum profit per weight (`MaxPW`) calculates profit-over-weight ratio for each item and sorts them in descending order. Then, `MaxPW` follows this ordering until the knapsack is full or no more items are left to be packed.

Minimum weight (`MinW`) favors lighter items, so it sorts items in ascending order based on their weight and packs them by following such order until they no longer fit.

Should there be a tie, all heuristics will choose the first conflicting item. Among these heuristics, `Def` is the fastest one to execute as it involves no additional operations. On the contrary, `MaxP`, `MaxPW`, and `MinW` take longer to compute but usually yield better results than using no order at all (`Def`). We are aware that more complex heuristics are available in the literature. Still, at this point in the investigation, we consider it suitable to test the proposed HH approach on this set of simple heuristics.

In our model, each HH represents a sequence of heuristics to apply to the problem in one specific order. For simplicity, we refer to its length as the cardinality of the HH. Let us consider the HH with cardinality of five ($\boldsymbol{h} \in \mathfrak{H}^5$) given by $\boldsymbol{h} = \{\texttt{Def}, \texttt{MaxP}, \texttt{MaxP}, \texttt{Def}, \texttt{MinW}\}$ (Figure 1). Please note that one of the available heuristics is not contained within the HH, `MaxPW`. In this HH, `Def` occupies positions 0 and 3, `MaxP` occupies positions 1 and 2, and `MinW` occupies position 4. Let us assume that $\boldsymbol{h}$ will solve a 12-item knapsack instance. In this case, there are 12 decisions to be made for solving this instance (d_1 to d_{12}). While some of the decisions will add an item to the knapsack, others will discard it. As the HH represents a sequence of five heuristics, the first five decisions will be made in the same order in which the heuristics are presented in $\boldsymbol{h}$: `Def`, `MaxP`, `MaxP`, `Def`, and `MinW`. After that, there are no more heuristics to choose to make further decisions. Then, the HH is used again, but now backwards: `MinW`, `Def`, `MaxP`, `MaxP`, and `Def`. The process repeats by inverting the sequence every time we reach the end of the HH. Although this scheme seems disruptive, we consider it feasible to favor the changes in heuristics throughout the solving process.

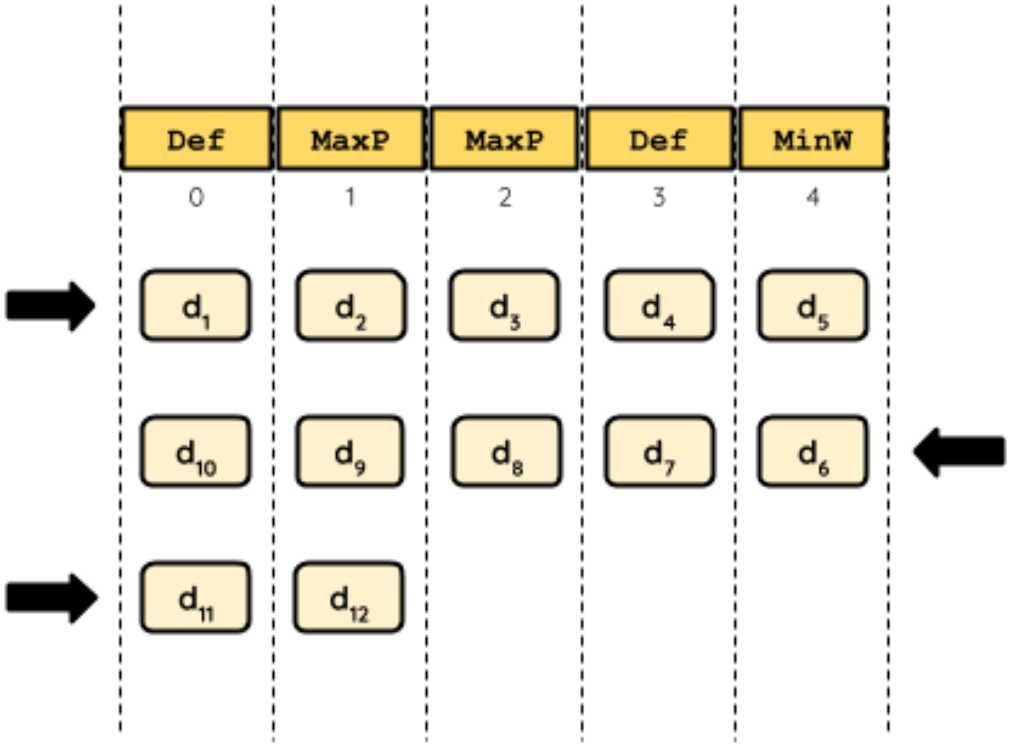

Figure 1. An example of a hyper-heuristic with a cardinality of five, and the way it solves an instance with 12 items. Black arrows indicate the direction in which heuristics are selected to make a decision.

3.1. Hyper-Heuristic Training

The learning mechanism within the HH model works as follows. The HH, $\boldsymbol{h} \in \mathfrak{H}^{\varpi}$, is randomly initialized (with a sequence of ϖ randomly selected heuristics), and it is used to solve the set of training instances. We register the profit of the solution obtained by $\boldsymbol{h}$ for each instance. The performance of the HH, $F(\boldsymbol{h}, \mathfrak{X})$, is then calculated as the average of all the profits obtained for the training set. In our EA implementation, a chromosome represents a HH that codes a sequence of heuristics to apply, i.e., $\boldsymbol{h}$. At each iteration, the process randomly chooses one mutation operator o_m from the available operators $\mathfrak{O}$. To do so, a uniform probability distribution is employed. Thus, EA applies it on a copy of the current HH, which produces a candidate HH, according to (2). The model evaluates the candidate HH on the set of training instances. If its performance is greater or equal (to favor diversity) than the current HH, this candidate takes its place. The process is repeated until it reaches the maximum number of allowed iterations $I_{\max}$. Pseudocode 1 details this training procedure.

The main goal of the learning mechanism is to produce a HH (an ordered sequence of heuristics) that maximizes the profit of an instance by packing appropriate items into the knapsack. In other words, it solves (1). Thus, the EA does not operate on the solution space $\mathfrak{X}$, but on the heuristic space $\mathfrak{H}$.

Pseudocode 1 Hyper-heuristic trainer

input:
Set of heuristics $\mathfrak{H}$,
Set of mutation operators $\mathfrak{O}$,
Set of instances to train the hyper-heuristic $\mathfrak{X}$,
Initial cardinality ω,
Performance function $F(\boldsymbol{h}, \mathfrak{X})$, and
Maximum number of iterations $I_{\max}$
output: Best selection hyper-heuristic $\boldsymbol{h}_*$

$\boldsymbol{h} \leftarrow$ INITIALIZERANDOMLY($\mathfrak{H}$, ω)
$F_h \leftarrow F(\boldsymbol{h}, \mathfrak{X})$ ▷ Evaluate the current HH
for i = 0 **to** $I_{\max}$ **do**
  $\boldsymbol{h}' \leftarrow$GENERATECANDIDATE($\boldsymbol{h}$)
  $F_{h'} \leftarrow F(\boldsymbol{h}', \mathfrak{X})$ ▷ Evaluate the candidate HH
  if $F_{h'} \geq F_h$ **then**
    $\boldsymbol{h} \leftarrow \boldsymbol{h}'$ and $F_h \leftarrow F_{h'}$
  end if
end for
return $\boldsymbol{h}_* \leftarrow \boldsymbol{h}$

procedure GENERATECANDIDATE($\boldsymbol{h}$)
  $\boldsymbol{h}' \leftarrow$ CLONE($\boldsymbol{h}$)
  $o_m \leftarrow$ CHOOSERANDOMLY($\mathfrak{O}$)
  $\boldsymbol{h}' \leftarrow o_m\{\boldsymbol{h}'\}$ ▷ Mutate $\boldsymbol{h}'$ by applying o_m
  return $\boldsymbol{h}'$
end procedure

3.2. Mutation Operators

The evolutionary process dynamically chooses among eight available mutation operators to alter the candidate hyper-heuristic $\boldsymbol{h}'$. Operators contained within set $\mathfrak{O}$ are described below. To clarify the behavior of such operators, we also provide a brief example of their behavior. Bear in mind that, in all the examples, we always consider the HH depicted in Figure 1 as the HH to mutate, i.e., $\boldsymbol{h}' = \{\texttt{Def}, \texttt{MaxP}, \texttt{MaxP}, \texttt{Def}, \texttt{MinW}\}$.

Add inserts a randomly chosen heuristic at a random position $i \sim \mathcal{U}\{0, \omega - 1\}$ into $\boldsymbol{h}'$. In doing so, cardinality ($\omega = \#\boldsymbol{h}'$) increases by one and existing heuristics at this position onward are shifted to the right. For the example, let us assume that $i = 3$ and that the random selection chooses `MaxPW`. Thus, the resulting HH has a cardinality of six and is given by $\boldsymbol{h}' = \{\texttt{Def}, \texttt{MaxP}, \texttt{MaxP}, \texttt{MaxPW}, \texttt{Def}, \texttt{MinW}\}$.

Single-point Flip selects a position $i \sim \mathcal{U}\{0, \omega - 1\}$ at random from $\boldsymbol{h}'$ and changes its heuristic $h_i \in \boldsymbol{h}$ to a different one (selected at random). Thus, cardinality is preserved. Let us suppose that $i = 0$. As `Def` is the current heuristic at this position, it cannot be chosen. Therefore, let us assume that the new heuristic is `MinW`. Then, the resulting HH is $\boldsymbol{h}' = \{\texttt{MinW}, \texttt{MaxP}, \texttt{MaxP}, \texttt{Def}, \texttt{MinW}\}$.

Two-point Flip is a more disruptive version of the previous operator. This time, heuristics at two different positions ($i, j \sim \mathcal{U}\{0, \omega - 1\}$) are renewed. Let us suppose that $i = 2$ and $j = 4$. In the first case, the available heuristics are `Def`, `MaxPW`, and `MinW`. In the second one, they are `Def`, `MaxP`, and `MaxPW`. Imagine that `MaxPW` and `Def` are selected, respectively. Then, the resulting HH still preserves cardinality and is given by $\boldsymbol{h}' = \{\texttt{Def}, \texttt{MaxP}, \texttt{MaxPW}, \texttt{Def}, \texttt{Def}\}$.

Neighbor-based Add is similar to `Add` as it also inserts a heuristic at a random position $k \sim \mathcal{U}\{0, \omega - 1\}$ within $\boldsymbol{h}'$. However, the heuristic to insert is randomly selected among neighboring heuristics (positions $j = i - 1$ and $k = i + 1$), as long as they

are valid positions. Should i correspond to an edge, the only neighbor is copied. Let us suppose that $i = 1$. In this case, the heuristic to insert would be either `Def` ($j = 0$) or `MaxP` ($k = 2$), with the same probability (50%). Imagine that `MaxP` is selected. Then, and as with the `Add` operator, cardinality grows to six and the resulting HH is $\boldsymbol{h}' = \{$`Def`, `MaxP`, `MaxP`, `MaxP`, `Def`, `MinW`$\}$.

Neighbor-based Single-point Flip is a variant of `Single-Point Flip` where the heuristic at $i \sim \mathcal{U}\{0, \varpi - 1\}$ changes into one of its neighbors (positions $j = i - 1$ and $k = i + 1$, if they exist). For example, suppose that $i = 2$. In this case, candidate replacements would be either `MaxP` ($j = 1$) or `Def` ($k = 3$), each one with a 50% probability. Let us suppose that `Def` is selected. Then, the resulting HH preserves cardinality and becomes $\boldsymbol{h}' = \{$`Def`, `MaxP`, `Def`, `Def`, `MinW`$\}$.

Neighbor-based Two-point Flip likewise, this is a variant of `Two-Point Flip` in which heuristics are chosen at random from neighboring ones. Let us suppose that $i = 2$ and $j = 4$. In the first case, available heuristics are `MaxP` and `Def`; in the second one, it is only `Def` since the position corresponds to an edge of the HH. Let us suppose that, in the first case, `Def` is selected. Therefore, the resulting HH is $\boldsymbol{h}' = \{$`Def`, `MaxP`, `Def`, `Def`, `Def`$\}$.

Swap interchanges heuristics at two randomly selected locations, $i, j \sim \mathcal{U}\{0, \varpi - 1\}$, and so preserves cardinality. For example, assume that $i = 1$ and $j = 3$. Thus, the resulting HH becomes $\boldsymbol{h}' = \{$`Def`, `Def`, `MaxP`, `MaxP`, `MinW`$\}$.

Remove randomly selects one position $i \sim \mathcal{U}\{0, \varpi - 1\}$ within the HH and removes the heuristic at that position. Therefore, cardinality is reduced by one. After removing the heuristic, upcoming ones are shifted to the left. For example, imagine that $i = 0$. Then, the resulting HH has a cardinality of four and is given by: $\boldsymbol{h}' = \{$`MaxP`, `MaxP`, `Def`, `MinW`$\}$.

Note that neighbor-based operators are allowed to replace the corresponding heuristic with the same one since it is determined by the neighbors. Certainly, other operators can be used but, for the sake of simplicity, we limit ourselves to these eight to formulate $\mathfrak{O}$. Finally, consider that there is only one operator that reduces cardinality (`Remove`), while two of them increase it (`Add` and `Neighbor-based Add`), and the remaining five preserve it (`Single-point Flip`, `Two-point Flip`, `Single-point Neighbor-based Flip`, `Two-point Neighbor-based Flip`, and `Swap`). Therefore, it is rather expected that trained HHs exhibit a higher cardinality than untrained ones. Because of this, a HH may end up choosing a different amount of items with each heuristic. We believe such flexibility in the learning process may favor more complex interactions between the available heuristics.

3.3. The Knapsack Problem Instances

In this investigation, we consider four datasets: α_1, α_2, β_1, and β_2. Dataset α_1 is used for training purposes and comprises 400 knapsack problem instances. We synthetically generated such instances for this work by using the algorithm proposed by Plata et al. [57]. This dataset contains a balanced mixture of problem instances, where each heuristic has an equal probability of performing best. Therefore, we are sure that no single heuristic outperforms others on all instances. We replicated the process to produce 400 additional problem instances for testing purposes (dataset α_2). These two datasets consider instances of 50 items and a knapsack capacity of 20 units. Dataset β_1 contains 600 hard instances proposed by Pisinger [45] and featuring 20 items but at different capacities. Finally, dataset β_2 consists of 600 additional hard instances (also proposed by Pisinger [45]). In this case, instances have 50 items and, again, exhibit different capacities. A summary of the datasets is provided in Table 1. Bear in mind that for the confirmatory experiments, we analyze the effect of changing the training dataset, but we shall discuss it in more detail further on.

Table 1. Datasets used in this work.

ID	Type	Features		
		Instances	Items	Capacity
α_1	Balanced	400	50	50
α_2	Balanced	400	50	50
β_1	Hard	600	20	Variable
β_2	Hard	600	50	Variable

3.4. Performance Metrics

We evaluate performance by considering both absolute and relative performances. All approaches are evaluated based on the profits obtained by their solutions (the sum of the profits of the items packed within the knapsack). The overall performance of a method on a particular set is calculated as the sum of the profits of the solutions produced for all the instances in such a set. However, it is also useful to estimate the relative performance, i.e., w.r.t. the other methods. For this purpose, we also include a performance ranking and the success rate (ρ).

The performance ranking is calculated as follows. All methods solve every instance in the set. Then, for each instance, the methods are sorted based on the profit of their solutions. The best one receives a ranking of 1, the second one a ranking of 2, and so on. In the case of ties, the ranking is the average of tied positions. This metric is helpful to identify ties that indicate a similar performance of two or more methods. Conversely, the success rate is calculated as the percentage of instances in a set where a particular method reaches or surpasses a threshold. In this investigation, such a threshold is given per instance and corresponds to the best result among those achieved by the heuristics, i.e., the best profit we can get from using each heuristic individually. For the sake of clarity, we present the success rate as a vector of two components: one where the threshold is achieved and one where it is surpassed. Bear in mind that the second component in the vector can only be achieved by mixing heuristics throughout the solving process, and so it only applies to HHs.

4. Experiments and Results

We organized our experiments in three stages: preliminary, exploratory, and confirmatory experiments. For the sake of clarity, these stages are further divided into categories. Figure 2 provides an overview of our experimental methodology. In the following lines, we carefully describe each stage, the corresponding experiments, and the results we obtained.

4.1. Preliminary Experiments

The preliminary experiments were designed first to justify the need for an intelligent way to combine the heuristics (the HHs) and, second, to determine a suitable set of parameters for running the EA to produce such HHs.

4.1.1. Preliminary Experiment I

The first experiment conducted in this work aims to prove that we need an intelligent approach to define the sequences of heuristics. In other words, we justify that it is unlikely for a random sequence of heuristics to achieve competent results. For this purpose, we generated 30 random HHs. The length of these HHs was set to 16 heuristics for empirical reasons. As in this experiment, we were only interested in the behavior of randomly generated HHs, the EA was not used to improve the initial HHs. Then, we used the 30 randomly generated HHs to solve set α_2, and we compared their results against the ones obtained by the available heuristics (see Section 3 for more details). For the sake of clarity, our comparisons include the results of three representative HHs from the perspective of total profit on set α_2: the best, the median, and the worst hyper-heuristics.

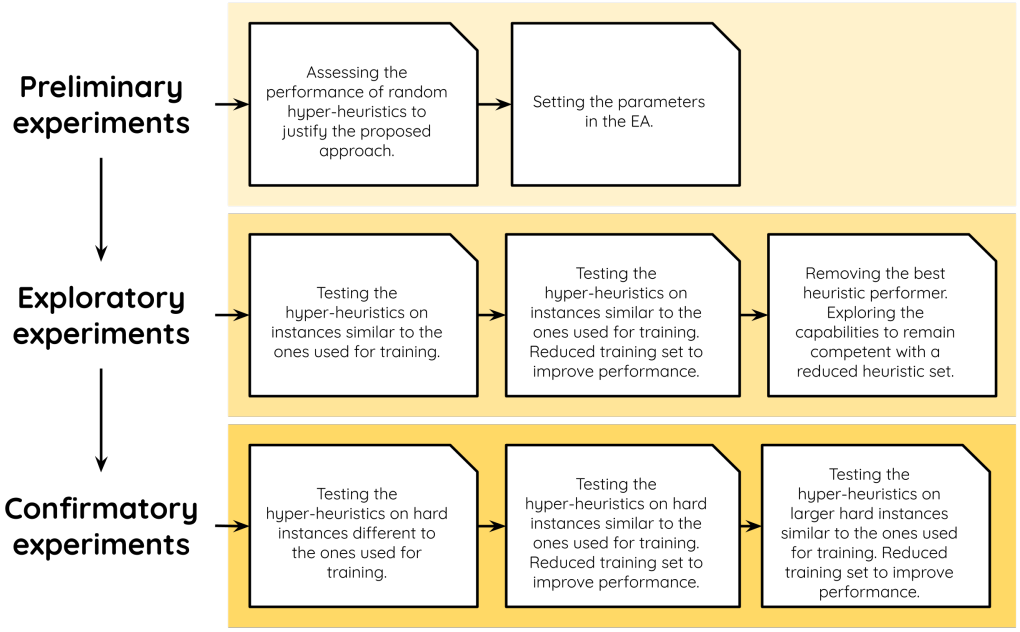

Figure 2. Three-stage methodology followed in this work.

One way to analyze the performance of the methods relies on ranking the resulting data. Table 2 shows the ranks obtained by the four heuristics, as well as the three representative HHs, when solving set α_2. As one may observe, in most of the cases, the heuristics obtain the best result in isolation (ranks close to 1), as it was expected as set α_2 is balanced. Although the best HH never occupies the last positions in the rank, the median performance hints that the good behavior of `Best-HH` is more likely to be due to chance. Furthermore, by randomly generating the sequences of heuristics, we risk producing something as bad as `Worst-HH`, where it replicated the worst heuristic for this set, i.e., `Def`.

Table 2 also shows the total profit obtained for each method on set α_2, which serves as evidence that it is indeed possible to produce one good HH at random, such that it outperforms several heuristics. However, this seems unlikely to happen by chance since at least half the HHs perform worse than `MinW`.

Regarding the success rate, the results confirm that generating sequences of heuristics at random is not a good idea. The success rate of the best randomly generated HH was $(0.0500, 0.0025)$, which means that only in 5% of the instances in set α_2 this HH produced a solution as competent as the best one among the four heuristics, while in 0.25% of the instances for the same set, the HH improved upon this result.

4.1.2. Preliminary Experiment II

Before moving further in this investigation, we needed to fix the number of iterations for the EA. Therefore, we generated 30 HHs by running the EA for 1000 iterations each time. In all the cases, the initial HH contained sequences of 12 heuristics, and we used α_1 as the training set. This time, the cardinality of the HHs was reduced by $1/4$ concerning the previous experiment since the mutation operators allow shortening and extending the sequences. Then, we no longer need long initial sequences as with the first preliminary experiment. For each HH, we recorded the Stagnation Point (SP). We define SP as the iteration at which the best solution was first encountered and for which the profit showed no improvement in subsequent iterations. Table 3 shows the stagnation points of the 30 runs of the EA.

Table 2. Ranks and total profit obtained by the four heuristics and the best, median, and worst randomly generated hyper-heuristics when tested on set α_2. The best result, measured in terms of total profit, is highlighted in bold.

	Heuristics				Hyper-Heuristics		
Rank	`Def`	`MaxP`	`MaxPW`	`MinW`	`Best-HH`	`Median-HH`	`Worst-HH`
1.0	1	100	102	78	1	0	0
1.5	80	0	9	22	12	20	79
2.0	4	0	111	0	107	1	4
2.5	6	0	59	0	64	5	6
3.0	0	13	20	35	83	65	1
3.5	9	1	82	4	99	21	10
4.0	0	27	15	90	29	80	0
4.5	2	0	2	43	3	46	2
5.0	0	165	0	28	1	103	0
5.5	17	0	0	34	0	34	17
6.0	0	12	0	32	1	6	0
6.5	281	0	0	16	0	16	281
7.0	0	82	0	18	0	3	0
Profit	241,081	343,872	**467,145**	376,148	436,393	361,284	241,025

Table 3. Stagnation points for the first 30 runs of the EA-based hyper-heuristic model.

Run	**SP**	**Run**	**SP**	**Run**	**SP**
1	162	11	46	21	675
2	127	12	51	22	41
3	743	13	114	23	30
4	39	14	38	24	47
5	682	15	18	25	40
6	38	16	317	26	28
7	68	17	56	27	152
8	21	18	146	28	78
9	405	19	176	29	37
10	64	20	30	30	34

We used these stagnation points to estimate the maximum number of iterations for running the EA. The average stagnation point among the 30 runs was 150.1, so we rounded it down to 150 iterations. Fifty additional iterations were added just as a precaution to minimize the risk of not reaching a good result. Thus, we set the maximum number of iterations to 200 for the rest of the experiments.

4.2. Exploratory Experiments

The exploratory experiments comprise a series of tests that cover general aspects of the proposed model, particularly those regarding how it copes with single heuristics on the balanced instance sets (sets α_1 and α_2).

4.2.1. Exploratory Experiment I

In this experiment, the rationale was to test the performance of HHs on similar instances to those they were trained on. Therefore, we produced 30 HHs with an initial cardinality of 12 heuristics each, as in preliminary experiment 2 (Section 4.1.2). Each HH was trained using set α_1 for 200 iterations. Afterward, we tested the resulting HHs on set α_2 and we compared the data against those obtained by heuristics applied in isolation.

Table 4 presents the ranking of the four heuristics as well as the best, median, and worst HHs produced in this experiment and tested on set α_2. This table also shows the total profit obtained by each of these methods on the same set. Based on the results obtained (both on ranks and total profit), we observe that the process is forcing the HHs to replicate the

behavior of the best performing heuristic for these types of instances (`MaxPW`). This also means that the HHs are, most of the time, ignoring the remaining heuristics. Although this may seem like a good choice as `MaxPW` is the best individual performer, the HHs are sacrificing the opportunity to improve their overall performance and outperform the best individual heuristic.

Table 4. Ranks and total profit obtained by the four heuristics and the best, median, and worst hyper-heuristics trained on set α_1 and tested on set α_2. The best result, measured in terms of total profit, is highlighted in bold.

	Heuristics				**Hyper-Heuristics**		
Rank	`Def`	`MaxP`	`MaxPW`	`MinW`	`Best-HH`	`Median-HH`	`Worst-HH`
1.0	85	100	0	87	0	7	0
1.5	0	0	6	8	6	0	8
2.0	1	1	5	4	5	16	1
2.5	0	0	100	1	100	99	100
3.0	2	0	3	0	3	10	5
3.5	1	0	264	1	264	258	258
4.0	0	0	16	1	16	4	25
4.5	0	0	6	0	6	1	1
5.0	11	21	0	78	0	5	2
5.5	0	1	0	1	0	0	0
6.0	18	195	0	183	0	0	0
7.0	282	82	0	36	0	0	0
Profit	241,081	343,872	**467,145**	376,148	**467,145**	467,103	466,711

4.2.2. Exploratory Experiment 2

In the previous experiment, the EA forced the HHs into replicating the behavior of one heuristic, `MaxPW`. In this experiment, we try to overcome this situation by reducing the number of instances in the training set. Then, for this experiment, 30 new HHs were produced, but this time, only 60% of the instances in set α_1 were used for training purposes. These 60% instances were randomly selected once and used for producing the 30 HHs. As in the previous experiment, the maximum number of iterations for the EA was set to 200 and all the HHs were initialized by using a random sequence of 12 heuristics. We used the 30 HHs to solve set α_2 and summarized the results through the rankings and total profit (Table 5).

Based on the results shown in Table 5, we observe a small improvement in `Best-HH` with respect to `MaxPW`. Although this supports our initial idea that we can improve the results obtained by the heuristics with a hyper-heuristic, the improvement is rather small and insignificant in practice. Furthermore, the success rate of `Best-HH` is $(0.285, 0.0)$, which is exactly the same as `MaxPW`.

For a more in-depth analysis about the performance of the best HH in this experiment, we plotted the performance of `Best-HH`, as well as of the best and worst heuristic for each particular instance in set α_2. For clarity, the results are sorted (from larger to smaller) on the profit obtained by `Best-HH` on each instance in the set. Figure 3 depicts the profit of each method across all instances. As we can observe, there are many cases where `Best-HH` obtains a smaller profit than the best heuristic for each particular instance.

Table 5. Ranks and total profit obtained by the four heuristics and the best, median, and worst hyper-heuristics trained on 60% of set α_1 and tested on set α_2. The best result, measured in terms of total profit, is highlighted in bold.

	Heuristics				Hyper-Heuristics		
Rank	`Def`	`MaxP`	`MaxPW`	`MinW`	`Best-HH`	`Median-HH`	`Worst-HH`
1.0	86	100	0	91	0	0	0
1.5	0	0	5	4	0	5	4
2.0	0	1	6	5	11	6	5
2.5	0	0	100	0	100	100	100
3.0	2	0	4	0	9	4	2
3.5	1	0	274	1	274	274	274
4.0	0	0	5	1	5	5	12
4.5	0	0	6	0	1	6	1
5.0	11	21	0	78	0	0	2
5.5	0	1	0	1	0	0	0
6.0	18	195	0	183	0	0	0
7.0	282	82	0	36	0	0	0
Profit	241,081	343,872	467,145	376,148	**467,182**	467,145	466,711

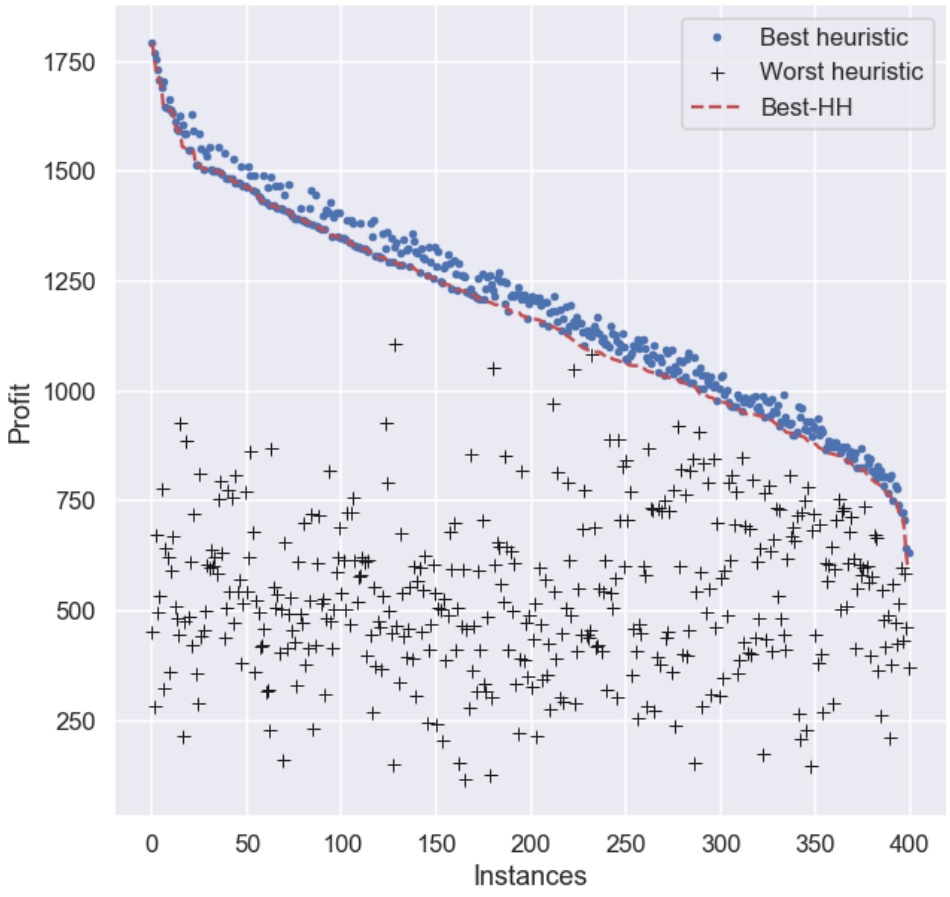

Figure 3. Profit per instance obtained by the best hyper-heuristic trained on 60% of set α_1 and the best and worst heuristic per instance on set α_2.

4.2.3. Exploratory Experiment III

So far, we have observed that, even though it is possible to overcome the best individual performer for each instance in some specific cases, oftentimes the HHs tend to replicate the behavior of the overall best heuristic (`MaxPW` for sets α_1 and α_2). Although this is not a bad result—the model produces very competent HHs, it is difficult to justify the additional

time devoted to producing such HHs given the small gains with respect to `MaxPW`. For this reason, in this experiment we wanted to explore the case where the HHs can only choose among `Def`, `MaxP`, and `MinW` (all the available heuristics but `MaxPW`) and evaluate if the HHs produced may still be considered competent. As in the previous experiments, we generated 30 HHs by training on set α_1 for 200 iterations each, and testing on set α_2. In all the cases, the HHs have an initial cardinality of 12 heuristics.

A comparison between the total profits of `Best-HH` and `MaxPW` (Table 6) reveals that a significant efficiency was lost due to the removal of `MaxPW` from the pool of heuristics (Figure 4). However, all three representative HHs (best, median, and worst) obtained the second rank in terms of total profit. The profit of `Best-HH` shows an improvement of nearly 6% and over 64% with respect to the `MinW` and `Def` heuristics, respectively. Therefore, the model can learn in harsh conditions and thus obtains promising results regardless of whether it was given a pool of heuristics with limited quality.

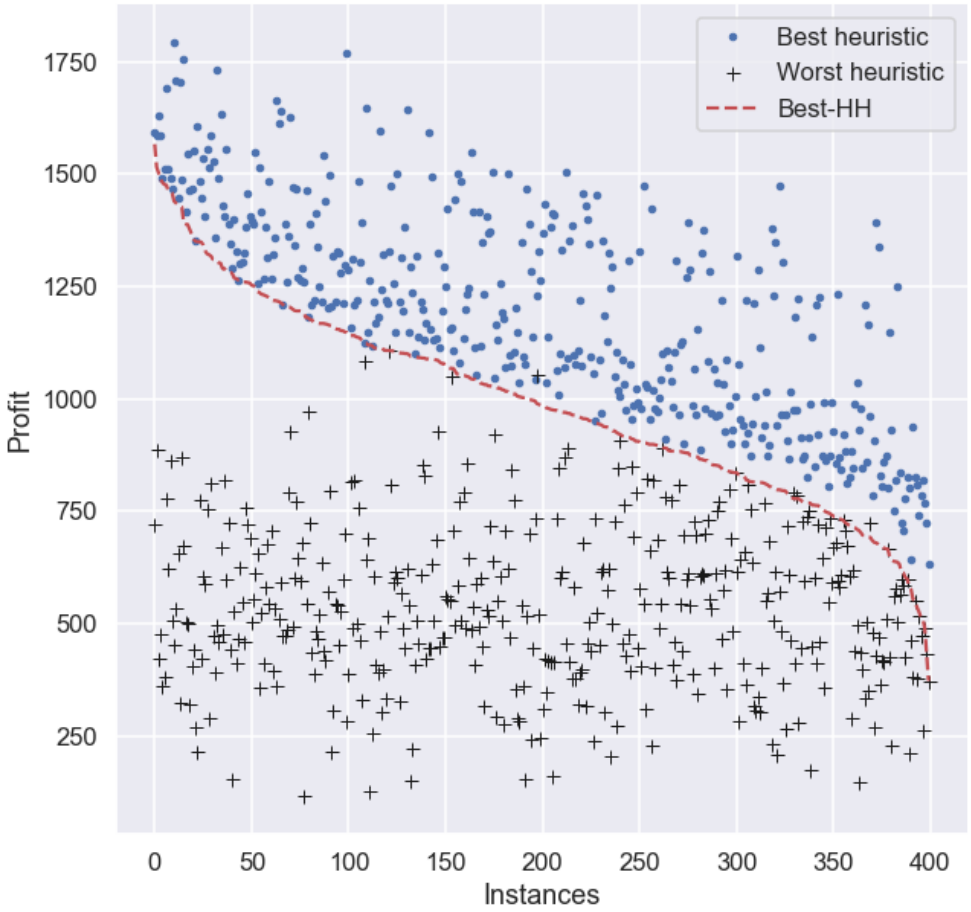

Figure 4. Profit per instance obtained by the best hyper-heuristic trained on set α_1 (removing `MaxPW` from the pool of available heuristics) and the best and worst heuristic per instance on set α_2 (including `MaxPW` in the comparison).

4.3. Confirmatory Experiments

Seeking to test the generalization properties of the proposed HH model, we conducted three additional experiments that now incorporate problem instances taken from the literature. These experiments were conducted in a similar way to the exploratory ones: each one involves the generation of 30 HHs with an initial cardinality of 12, which were trained for 200 iterations. However, this time we tried some combinations of the instance sets for training and testing. The rationale behind these experiments is to observe how well the HHs adapt to changes in the properties of the instances being solved with respect to the instances used for producing such HHs.

Table 6. Ranks and total profit obtained by the four heuristics and the best, median, and worst hyper-heuristics trained on set α_1 and tested on set α_2, but `MaxPW` is not available for the hyper-heuristics. The best result, measured in terms of total profit, is highlighted in bold.

	Heuristics				Hyper-Heuristics		
Rank	`Def`	`MaxP`	`MaxPW`	`MinW`	`Best-HH`	`Median-HH`	`Worst-HH`
1.0	83	99	111	96	0	0	0
1.5	3	0	2	2	1	2	2
2.0	13	2	219	1	49	39	8
2.5	0	0	5	3	19	32	31
3.0	1	14	30	19	82	61	37
3.5	0	0	5	1	49	87	106
4.0	1	4	14	16	70	51	59
4.5	0	0	1	3	33	67	84
5.0	0	26	13	91	53	33	42
5.5	0	1	0	12	17	18	22
6.0	17	176	0	129	18	6	5
6.5	0	0	0	6	7	2	3
7.0	282	78	0	21	2	2	1
Profit	241,081	343,872	**467,145**	376,148	397,373	395,586	391,940

4.3.1. Confirmatory Experiment I

So far, we have evaluated the performance of HHs on instances similar to the ones used for training, under various scenarios. In the first confirmatory experiment, we use all the instances from set α_1 to train the HHs, but test their performance on set β_1. Let us recall that set β_1 contains 600 hard instances proposed by Pisinger [45] and feature 20 items and different capacities. The relevant issue regarding set β_1 is that such instances are considered hard to solve.

Table 7 shows that `MaxPW` is, once again, the best individual performer in this experiment. Although all the produced HHs perform similarly (cf. `Worst-HH`), they are unable to improve upon the results obtained by the best heuristic. Nonetheless, all the HHs produced obtained a total profit larger than those obtained by the remaining heuristics.

Table 7. Ranks and total profit obtained by the four heuristics and the best, median, and worst hyper-heuristics trained on set α_1 and tested on set β_1. The best result, measured in terms of total profit, is highlighted in bold.

	Heuristics				Hyper-Heuristics		
Rank	`Def`	`MaxP`	`MaxPW`	`MinW`	`Best-HH`	`Median-HH`	`Worst-HH`
1.0	24	116	32	10	42	39	31
1.5	16	20	71	11	60	39	25
2.0	32	41	111	79	124	57	33
2.5	14	19	80	27	56	54	54
3.0	54	47	63	38	59	84	77
3.5	18	4	62	16	48	47	43
4.0	69	33	47	35	53	98	79
4.5	30	3	42	14	32	44	47
5.0	63	31	26	48	46	73	94
5.5	22	4	31	17	21	13	24
6.0	190	43	20	64	36	42	75
6.5	15	6	13	14	5	2	3
7.0	53	233	2	227	18	8	15
Profit	3,804,271	3,724,588	**4,039,708**	3,867,345	4,031,981	3,958,061	3,905,734

Even if the HHs were incapable of overcoming MaxPW in this experiment, it is interesting to see how close their performance is, especially as these HHs were trained with instances of a different type than those used for testing.

4.3.2. Confirmatory Experiment II

In the previous experiment, we evaluated the performance of HHs trained on balanced sets of instances when tested on hard instances, and we observed a limited capacity to deal with such instances. In this experiment, we show that this behaviour is not due to the hardness of the instances, but because of the discrepancy between the instances used for training and the ones used for testing. For this reason, this experiment is twofold: (1) we test how HHs behave when trained and tested on hard instances (set β_1) and (2) we try to reduce the effect of MaxPW in the resulting HHs by reducing the number of instances in the training set. Then, we produced 30 new HHs using only 60% of the instances in set β_1 for training purposes. As in previous experiments, the 60% instances were randomly selected once and used for producing the 30 HHs. For consistency, the maximum number of iterations for the EA was set to 200 and all the HHs were initialized to a cardinality of 12. The HHs produced in this experiment were evaluated on the remaining 40% of the instances in set β_1. The results for rankings and total profit derived from this experiment are summarized in Table 8.

Table 8. Ranks and total profit obtained by the four heuristics and the best, median, and worst hyper-heuristics trained on 60% of set β_1 and tested on the remaining 40% of set β_1. The best result, measured in terms of total profit, is highlighted in bold.

	Heuristics				**Hyper-Heuristics**		
Rank	Def	MaxP	MaxPW	MinW	Best-HH	Median-HH	Worst-HH
1.0	16	54	3	3	12	4	2
1.5	1	0	6	5	13	3	16
2.0	14	8	10	9	29	28	20
2.5	2	5	47	3	51	59	59
3.0	14	16	54	30	67	64	56
3.5	3	3	20	0	19	25	22
4.0	19	13	24	12	16	21	13
4.5	4	1	29	11	18	20	21
5.0	30	19	10	19	10	8	11
5.5	9	3	21	13	4	6	6
6.0	101	16	9	28	1	0	10
6.5	6	4	6	10	0	2	2
7.0	21	98	1	97	0	0	2
Profit	1,584,007	1,540,100	1,673,387	1,600,015	**1,681,403**	1,681,127	1,678,375

This experiment confirms the importance of the instances used for training and their similarity with those solved during the test process. This time, the performance of most of the hyper-heuristics produced using 60% of set β_1 when tested on the remaining 40% of set β_1 improve on MaxPW (the best heuristic for this set). In fact, the performance of Worst-HH is practically the same as MaxPW (Figure 5). Although the results suggest that an improvement in the total profit can be obtained by using the proposed hyper-heuristic approach, the gain in profit derived from using Best-HH instead of MaxPW is rather small (<0.5%). However, this result should not be interpreted as an indication that Best-HH is replicating the behavior of MaxPW. In fact, their overall profit is similar (with a small gain for Best-HH but their specific performance is not. As depicted in Figure 6, when the ranks are analyzed, we observe that Best-HH obtains better results in more cases than the best individual heuristic, MaxPW.

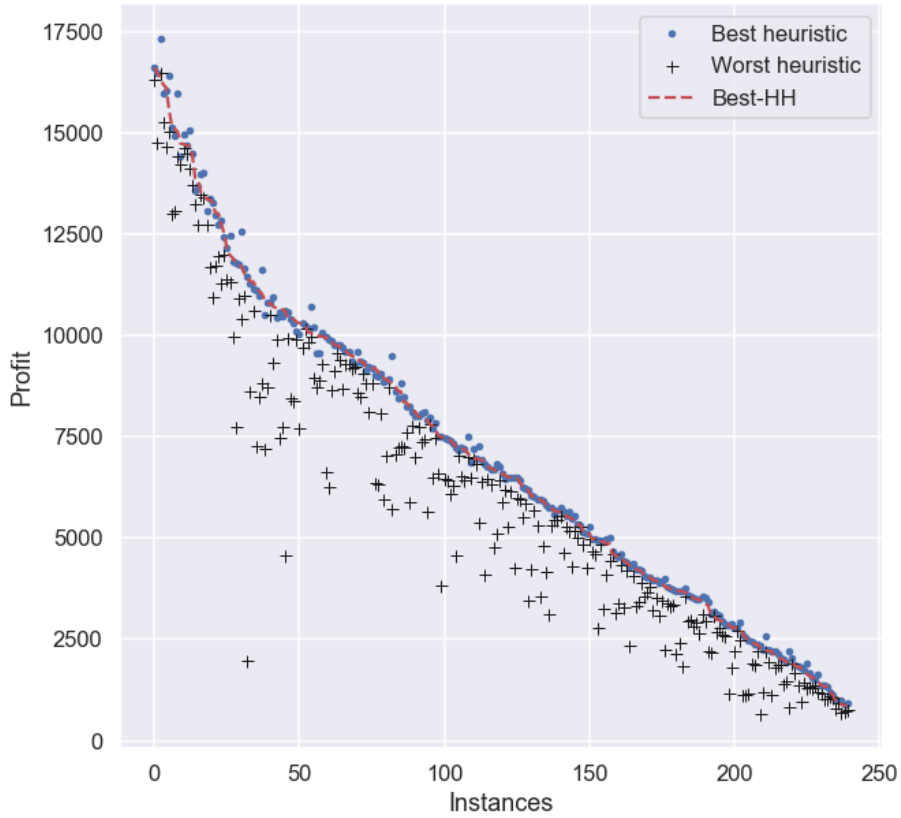

Figure 5. Profit per instance obtained by the best hyper-heuristic trained on 60% of set β_1, and the best and worst heuristic per instance on the remaining 40% of set β_1.

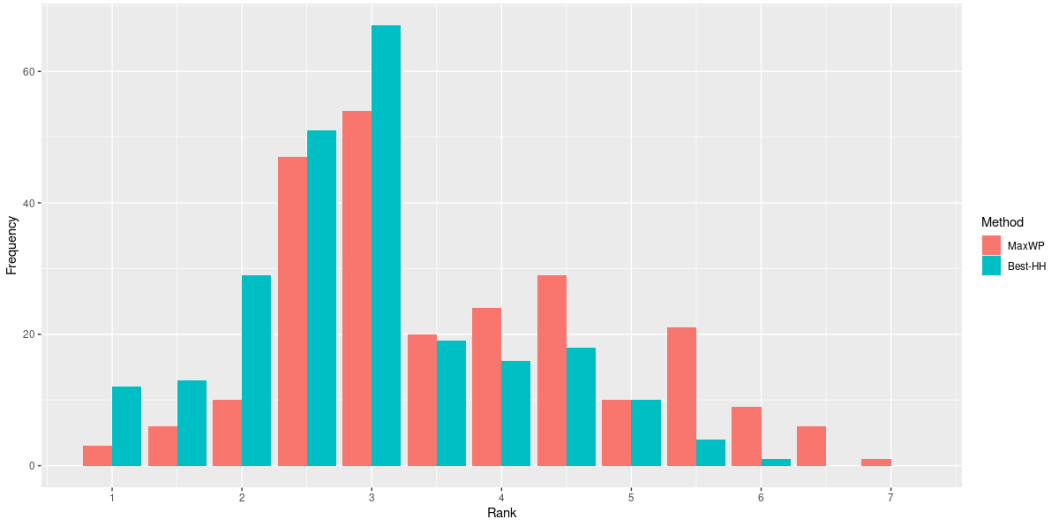

Figure 6. Frequency of the ranks obtained by `MaxPW` and `Best-HH` (trained on 60% of set β_1) the remaining 40% of set β_1.

4.3.3. Confirmatory Experiment III

In this final experiment, we extend our analysis to hard and larger instances. This time, we produced 30 HHs by using 60% of set β_2 and tested them on the remaining 40% of the same set. With this experiment, we validate that the learning method is quite stable as it still produces competent hyper-heuristics. Despite the fact that the set comprised instances with different features, all three cases beat the best operator (`MaxPW`) in isolation. Additionally, setting a good training set seems to impact the efficiency of the HH model.

Training done on α_1 seemed to negatively affect the results, as seen on exploratory experiment I and confirmatory experiment I. It is important to note that sets α_1 and α_2 are balanced and synthetically produced. Thus, 25% of the problem instances are designed to maximize the performance of a single low-level heuristic. This pattern is repeated for all the heuristics in this set. Conversely, hard problem instances from sets β_1 and β_2 are randomly sampled (without replacement) using three different seeds. This training scheme is more representative of real-life applications, where often no balanced or ideal conditions are met.

The model's behavior is similar to what we observed in previous cases. The hyper-heuristics trained on hard and larger instances are still competitive regarding the single heuristics applied in isolation. This time, the improvement on `MaxPW` is smaller than in previous cases ($<$0.01%). Table 9 shows that, as in previous cases, there is a difference in the behavior of the heuristics and HHs in terms of ranks, but it does not necessarily represent a significant difference in terms of total profit. However, when we analyze the success rate, we observe that `Best-HH` obtains promising results, $(0.6542, 0.0958)$. This indicates that in little more of 65% of the instances in the 40% of set β_2 used for testing, `Best-HH` was equal in performance to the best out of the four heuristics. Besides, in almost 10% of the instances for the same set, `Best-HH` improved upon this result.

Table 9. Ranks and total profit obtained by the four heuristics and the best, median, and worst hyper-heuristics trained on 60% of set β_2 and tested on the remaining 40% of set β_2. The best result, measured in terms of total profit, is highlighted in bold.

	Heuristics				**Hyper-Heuristics**		
Rank	`Def`	`MaxP`	`MaxPW`	`MinW`	`Best-HH`	`Median-HH`	`Worst-HH`
1.0	10	20	0	15	10	4	15
1.5	1	1	4	2	8	6	2
2.0	10	12	14	2	23	19	7
2.5	0	1	81	2	80	82	80
3.0	14	25	62	33	62	71	63
3.5	1	2	31	3	29	30	26
4.0	3	5	9	9	8	6	16
4.5	0	0	17	6	10	11	12
5.0	25	27	4	44	2	3	4
5.5	3	2	15	8	7	8	9
6.0	140	34	2	27	1	0	5
6.5	1	1	1	0	0	0	1
7.0	32	110	0	89	0	0	0
Profit	3,752,072	3,631,322	4,046,715	3,930,092	**4,052,501**	4,052,039	4,045,794

4.4. Discussion

We would like to discuss the rationale behind the proposed model and why it may be useful for other problem domains besides the one studied in this document.

As with other packing problems, the iterative nature of KP makes it a great candidate for a hyper-heuristic approach. The inherent mapping of problem states to decisions (or, in this case, packing heuristics) can lead to an optimal item selection by overfitting if the relationship were to be searched exhaustively. A generalization of this item selection is the basis of the rationale behind this approach. Furthermore, the advantages of mixing heuristics have previously been discussed in detail for various optimization and search problems [27,58].

The evidence obtained from the experiments confirms that it is possible to produce a sequence of heuristics that provides an acceptable performance when tested on a set of instances. Allow us to explain this in more detail. Let us assume that a hyper-heuristic HH_1 is to be produced for solving only one KP instance with n items, KP_1. If the cardinality of the hyper-heuristic is equal to the number of items to pack ($\varpi = n$), then HH_1 can

decide which heuristic to use for packing each item. Among all the possible HHs that could be produced, there is one where all the decisions are correct, HH_1^*, which represents the optimal sequence to pack the items in KP_1, given the available heuristics. Now, let us produce a new hyper-heuristic HH_2^*, the best sequence of heuristics for solving a second KP instance, KP_2, also containing n items (the simplest case for the analysis). There is no guarantee that the previous hyper-heuristic, HH_1^*, will also represent the optimal sequence of heuristics for solving KP_2. If we keep the idea of having one individual decision per item, only a few of these decisions will be the same for both sequences, HH_1^* and HH_2^*. In order to merge the two sequences, some errors must be accepted. The task of the evolutionary framework is to decide which errors to accept so that the performance is the best among all the instances in the training set. Overall, the model is not looking for the best sequence of heuristics for each particular instance but the best sequence to solve, acceptably, all the instances in the training set.

5. Conclusions and Future Work

In this work, we analyzed the efficiency of a selection HH which does not depend on problem characterization. The analysis showed that the learning mechanism deals very well on all instances despite its simple parameters. For small instances like the ones in sets α_1 and α_2, the `MaxPW` heuristic seemed like the most suitable heuristic in isolation. The instances used for training were generated with one packing heuristic in mind. Instead, instances in the literature are considered harder and represent a challenge for a single heuristic in isolation. It is also important to note that the instance sets may be considered small, so the learning process was not computationally expensive. For larger datasets with more instances to solve, a HH could perform better if one has adequate resources for the learning process.

Our work proves the feasibility of a feature-independent hyper-heuristic approach powered by an evolutionary algorithm. The results confirm our idea that it is possible to generate generic sequences of heuristics that perform better than individual heuristics for specific sets of instances. Our results also demonstrate that the similarity between the training and testing instances influences the model's performance. In other words, the model can generalize to unseen types of instances, but the ideal scenario would be to train the hyper-heuristics on instances with similar features to the ones to be solved when training is over. At this point, we consider it fair to mention that we are aware of the diversity of the optimization problems. We have selected the KP because it was convenient for our study, as we can easily develop and test hyper-heuristics for this particular problem. However, many other exciting problems could have also been considered in this regard. Our interest was to propose a new hyper-heuristic method to deal with instances that are hard to solve by exact methods.

Many considerations could be taken into account to improve the analysis. Fixing the size of HHs and setting a single heuristic per gene in the model may impact the frequency analysis. Adding more mutation operators and tweaking their probability distribution is also something to consider for future work. More importantly, extending the amount of packing heuristics to choose from, though increasing the heuristic search space, may display more explicit differences between heuristic sequences. Once again, we would like to emphasize that the selection HH proposed did not deal with any problem characterization or feature analysis. Adding problem characterization may improve the performance of the learning process even more at the expense of some additional computational effort.

Although we have not tested the proposed model on other NP-hard optimization problems, we are confident that the model can work correctly on similar problems such as packing and scheduling. Based on our experience, those problems show similar properties that allow hyper-heuristics to grasp the structure of the instances and decide which heuristic to apply at certain times of the solving process. Therefore, the proposed model should perform properly. Nonetheless, testing our approach on other challenging problems is, undoubtedly, a path for future work.

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

Funding: This research was partially funded by the research group with strategic focus in intelligent systems at ITESM, grant NUA A00834075, and CONACyT Basic Science projects with grant number 287479 and fellowship 2021-000001-01NACF-00604.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Sánchez, M.; Cruz-Duarte, J.M.; Ortiz-Bayliss, J.C.; Ceballos, H.; Terashima-Marín, H.; Amaya, I. A Systematic Review of Hyper-heuristics on Combinatorial Optimization Problems. *IEEE Access* **2020**, 1–28. [CrossRef]
2. Bai, R.; Burke, E.K.; Gendreau, M.; Kendall, G.; McCollum, B. Memory Length in Hyper-heuristics: An Empirical Study. In Proceedings of the IEEE Symposium on Computational Intelligence in Scheduling, SCIS '07, Honolulu, HI, USA, 2–4 April 2007; pp. 173–178. [CrossRef]
3. Burke, E.K.; Gendreau, M.; Hyde, M.; Kendall, G.; Ochoa, G.; Özcan, E.; Qu, R. Hyper-heuristics: A survey of the state of the art. *J. Oper. Res. Soc.* **2013**, *64*, 1695–1724. [CrossRef]
4. Pillay, N.; Qu, R. *Hyper-Heuristics: Theory and Applications*; Natural Computing Series; Springer International Publishing: Cham, Switzerland, 2018; doi:10.1007/978-3-319-96514-7. [CrossRef]
5. Poli, R.; Graff, M. There is a free lunch for hyper-heuristics, genetic programming and computer scientists. In Proceedings of the 12th European Conference on Genetic Programming (EuroGP 2009), Tübingen, Germany, 15–17 April 2009; Springer: Berlin/Heidelberg, Germany, 2009; pp. 195–207._17. [CrossRef]
6. Özcan, E.; Bilgin, B.; Korkmaz, E.E. A Comprehensive Analysis of Hyper-heuristics. *Intell. Data Anal.* **2008**, *12*, 3–23. [CrossRef]
7. Hart, E.; Sim, K. A Hyper-Heuristic Ensemble Method for Static Job-Shop Scheduling. *Evol. Comput.* **2016**, *24*, 609–635. [CrossRef] [PubMed]
8. Hyde, M. A Genetic Programming Hyper-Heuristic Approach to Automated Packing. Ph.D. Thesis, University of Nottingham, Nottingham, UK, 2010.
9. Drake, J.H.; Hyde, M.; Ibrahim, K.; Özcan, E. A Genetic Programming Hyper-Heuristic for the Multidimensional Knapsack Problem. In Proceedings of the 11th IEEE International Conference on Cybernetic Intelligent Systems, Limerick, Ireland, 23–24 August 2012.
10. Lourenço, N.; Pereira, F.B.; Costa, E. The Importance of the Learning Conditions in Hyper-heuristics. In Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation, GECCO '13, Amsterdam, The Netherlands, 6–10 July 2013; ACM: New York, NY, USA, 2013; pp. 1525–1532. [CrossRef]
11. Falkenauer, E. A hybrid grouping genetic algorithm for bin packing. *J. Heuristics* **1996**, *2*, 5–30. [CrossRef]
12. Ortiz-Bayliss, J.C.; Moreno-Scott, J.H.; Terashima-Marín, H. Automatic Generation of Heuristics for Constraint Satisfaction Problems. In *Nature Inspired Cooperative Strategies for Optimization (NICSO 2013)*; Studies in Computational Intelligence; Springer: Cham, Switzerland, 2013; Volume 512, pp. 315–327._24. [CrossRef]
13. Hart, E.; Sim, K. On the Life-Long Learning Capabilities of a NELLI*: A Hyper-Heuristic Optimisation System. In Proceedings of the International Conference on Parallel Problem Solving from Nature—PPSN XIII, Ljubljana, Slovenia, 13–17 September 2014; Volume 8672, pp. 282–291.
14. Sim, K.; Hart, E. An Improved Immune Inspired Hyper-heuristic for Combinatorial Optimisation Problems. In Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation, GECCO '14, Vancouver, BC, Canada, 12–16 July 2014; ACM: New York, NY, USA, 2014; pp. 121–128. [CrossRef]
15. Amaya, I.; Ortiz-Bayliss, J.C.; Conant-Pablos, S.; Terashima-Marin, H. Hyper-heuristics Reversed: Learning to Combine Solvers by Evolving Instances. In Proceedings of the 2019 IEEE Congress on Evolutionary Computation (CEC), Wellington, New Zealand, 10–13 June 2019; pp. 1790–1797. [CrossRef]
16. Hart, E.; Sim, K. On Constructing Ensembles for Combinatorial Optimisation. In Proceedings of the Genetic and Evolutionary Computation Conference Companion, GECCO '17, Berlin, Germany, 15–19 July 2017; ACM: New York, NY, USA, 2017; pp. 5–6. [CrossRef]
17. Burke, E.K.; Hyde, M.R.; Kendall, G.; Woodward, J. Automating the Packing Heuristic Design Process with Genetic Programming. *Evol. Comput.* **2012**, *20*, 63–89._a_00044. [CrossRef]

18. Drake, J.H.; Özcan, E.; Burke, E.K. Modified Choice Function Heuristic Selection for the Multidimensional Knapsack Problem. In *Genetic and Evolutionary Computing*; Sun, H., Yang, C.Y., Lin, C.W., Pan, J.S., Snasel, V., Abraham, A., Eds.; Springer International Publishing: Cham, Switzerland, 2015; pp. 225–234.
19. Drake, J.H.; Özcan, E.; Burke, E.K. A Case Study of Controlling Crossover in a Selection Hyper-heuristic Framework Using the Multidimensional Knapsack Problem. *Evol. Comput.* **2016**, *24*, 113–141._a_00145. [CrossRef]
20. Duhart, B.; Camarena, F.; Ortiz-Bayliss, J.C.; Amaya, I.; Terashima-Marín, H. An Experimental Study on Ant Colony Optimization Hyper-Heuristics for Solving the Knapsack Problem. In *Pattern Recognition*; Martínez-Trinidad, J.F., Carrasco-Ochoa, J.A., Olvera-López, J.A., Sarkar, S., Eds.; Springer International Publishing: Cham, Switzerland, 2018; pp. 62–71.
21. Gómez-Herrera, F.; Ramirez-Valenzuela, R.A.; Ortiz-Bayliss, J.C.; Amaya, I.; Terashima-Marín, H. A Quartile-Based Hyper-heuristic for Solving the 0/1 Knapsack Problem. In *Advances in Soft Computing*; Castro, F., Miranda-Jiménez, S., González-Mendoza, M., Eds.; Springer International Publishing: Cham, Switzerland, 2018; pp. 118–128.
22. Drake, J.H.; Kheiri, A.; Özcan, E.; Burke, E.K. Recent advances in selection hyper-heuristics. *Eur. J. Oper. Res.* **2020**, *285*, 405–428. [CrossRef]
23. Wilbaut, C.; Hanafi, S.; Salhi, S. A survey of effective heuristics and their application to a variety of knapsack problems. *IMA J. Manag. Math.* **2008**, *19*, 227. [CrossRef]
24. Rice, J.R. The Algorithm Selection Problem. *Adv. Comput.* **1976**, *15*, 65–118. [CrossRef]
25. Garza-Santisteban, F.; Sanchez-Pamanes, R.; Puente-Rodriguez, L.A.; Amaya, I.; Ortiz-Bayliss, J.C.; Conant-Pablos, S.; Terashima-Marin, H. A Simulated Annealing Hyper-heuristic for Job Shop Scheduling Problems. In Proceedings of the 2019 IEEE Congress on Evolutionary Computation (CEC), Wellington, New Zealand, 10–13 June 2019; pp. 57–64. [CrossRef]
26. Alanazi, F.; Lehre, P.K. Runtime analysis of selection hyper-heuristics with classical learning mechanisms. In Proceedings of the 2014 IEEE Congress on Evolutionary Computation (CEC), Beijing, China, 6–11 July 2014; pp. 2515–2523. [CrossRef]
27. Lehre, P.K.; Özcan, E. A Runtime Analysis of Simple Hyper-heuristics: To Mix or Not to Mix Operators. In Proceedings of the Twelfth Workshop on Foundations of Genetic Algorithms XII, FOGA XII '13, Adelaide, Australia, 16–20 January 2013; ACM: New York, NY, USA, 2013; pp. 97–104. [CrossRef]
28. Ortiz-Bayliss, J.C.; Terashima-Marín, H.; Özcan, E.; Parkes, A.J.; Conant-Pablos, S.E. Exploring heuristic interactions in constraint satisfaction problems: A closer look at the hyper-heuristic space. In Proceedings of the 2013 IEEE Congress on Evolutionary Computation, Cancun, Mexico, 20–23 June 2013; pp. 3307–3314. [CrossRef]
29. Amaya, I.; Ortiz-Bayliss, J.C.; Rosales-Pérez, A.; Gutiérrez-Rodríguez, A.E.; Conant-Pablos, S.E.; Terashima-Marín, H.; Coello, C.A. Enhancing Selection Hyper-Heuristics via Feature Transformations. *IEEE Comput. Intell. Mag.* **2018**, *13*, 30–41. [CrossRef]
30. Sánchez-Díaz, X.F.C.; Ortiz-Bayliss, J.C.; Amaya, I.; Cruz-Duarte, J.M.; Conant-Pablos, S.E.; Terashima-Marín, H. A Preliminary Study on Feature-independent Hyper-heuristics for the 0/1 Knapsack Problem. In Proceedings of the 2020 IEEE Congress on Evolutionary Computation (CEC), Glasgow, UK, 19–24 July 2020; pp. 1–8. [CrossRef]
31. Dudzinski, K.; Walukiewicz, S. Exact methods for the knapsack problem and its generalizations. *Eur. J. Oper. Res.* **1987**, *28*, 3–21. [CrossRef]
32. Martello, S.; Pisinger, D.; Toth, P. New trends in exact algorithms for the 0–1 knapsack problem. *Eur. J. Oper. Res.* **2000**, *123*, 325–332. [CrossRef]
33. Lawler, E.L. Fast Approximation Algorithms for Knapsack Problems. *Math. Oper. Res.* **1979**, *4*, 339–356. [CrossRef]
34. Lienland, B.; Zeng, L. A Review and Comparison of Genetic Algorithms for the 0–1 Multidimensional Knapsack Problem. *Int. J. Oper. Res. Inf. Syst.* **2015**, *6*, 21–31. [CrossRef]
35. Hembecker, F.; Lopes, H.; Godoy, W., Jr. Particle Swarm Optimization for the Multidimensional Knapsack Problem. In Proceedings of the International Conference on Adaptive and Natural Computing Algorithms, Warsaw, Poland, 11–14 April 2007; Volume 4331, pp. 358–365._40. [CrossRef]
36. Gómez, N.; López, L.; Albert, A. Multidimensional knapsack problem optimization using a binary particle swarm model with genetic operations. *Soft Comput.* **2018**, *22*, 2567–2582. [CrossRef]
37. Razavi, S.; Sajedi, H. Cognitive discrete gravitational search algorithm for solving 0–1 knapsack problem. *J. Intell. Fuzzy Syst.* **2015**, *29*, 2247–2258. [CrossRef]
38. Patvardhan, C.; Bansal, S.; Srivastav, A. Quantum-Inspired Evolutionary Algorithm for difficult knapsack problems. *Memetic Comput.* **2015**, *7*. [CrossRef]
39. Lv, J.; Wang, X.; Huang, M.; Cheng, H.; Li, F. Solving 0–1 knapsack problem by greedy degree and expectation efficiency. *Appl. Soft Comput. J.* **2016**, *41*, 94–103. [CrossRef]
40. Kulkarni, A.J.; Shabir, H. Solving 0–1 Knapsack Problem using Cohort Intelligence Algorithm. *Int. J. Mach. Learn. Cybern.* **2016**, *7*, 427–441. [CrossRef]
41. Banda, J.; Velasco, J.; Berrones, A. A hybrid heuristic algorithm based on Mean-Field Theory with a Simple Local Search for the Quadratic Knapsack Problem. In Proceedings of the 2017 IEEE Congress on Evolutionary Computation (CEC), San Sebastián, Spain, 5–8 June 2017; pp. 2559–2565. [CrossRef]
42. Martello, S.; Toth, P. *Knapsack Problems: Algorithms and Computer Implementations*; John Wiley & Sons, Inc.: Hoboken, NJ, USA, 1990.
43. Kellerer, H.; Pferschy, U.; Pisinger, D. *Knapsack Problems*; Springer: Berlin, Germany, 2004.
44. Furini, F.; Monaci, M.; Traversi, E. Exact approaches for the knapsack problem with setups. *Comput. Oper. Res.* **2018**, *90*. [CrossRef]

45. Pisinger, D. Where Are the Hard Knapsack Problems? *Comput. Oper. Res.* **2005**, *32*, 2271–2284. [CrossRef]
46. Sun, X.; Sheng, H.; Li, D. An exact algorithm for 0–1 polynomial knapsack problems. *J. Ind. Manag. Optim.* **2007**, *3*, 223–232. [CrossRef]
47. Burke, E.K.; Hyde, M.R.; Kendall, G.; Ochoa, G.; Özcan, E.; Woodward, J.R. A classification of hyper-heuristic approaches: Revisited. In *Handbook of Metaheuristics*; Springer: Cham, Switzerland, 2019; pp. 453–477.
48. Dokeroglu, T.; Sevinc, E.; Kucukyilmaz, T.; Cosar, A. A survey on new generation metaheuristic algorithms. *Comput. Ind. Eng.* **2019**, *137*, 106040. [CrossRef]
49. Vikhar, P.A. Evolutionary algorithms: A critical review and its future prospects. In Proceedings of the 2016 IEEE International Conference on Global Trends in Signal Processing, Information Computing and Communication (ICGTSPICC), Jalgaon, India, 22–24 December 2016; pp. 261–265.
50. Cruz-Duarte, J.M.; Amaya, I.; Ortiz-Bayliss, J.C.; Conant-Pablos, S.E.; Terashima-Marín, H. A Primary Study on Hyper-Heuristics to Customise Metaheuristics for Continuous Optimisation. In Proceedings of the 2020 IEEE Congress on Evolutionary Computation, Glasgow, UK, 19–24 July 2020.
51. Cruz-Duarte, J.M.; Ortiz-Bayliss, J.C.; Amaya, I.; Shi, Y.; Terashima-Marín, H.; Pillay, N. Towards a Generalised Metaheuristic Model for Continuous Optimisation Problems. *Mathematics* **2020**, *8*, 2046. [CrossRef]
52. Salov, V. Notation for Iteration of Functions, Iteral. *arXiv* **2012**, arXiv:1207.0152.
53. Abdel-Basset, M.; Abdel-Fatah, L.; Sangaiah, A.K. *Metaheuristic Algorithms: A Comprehensive Review*; Elsevier Inc.: London, UK, 2018; pp. 185–231. [CrossRef]
54. Opara, K.R.; Arabas, J. Differential Evolution: A survey of theoretical analyses. *Swarm Evol. Comput.* **2019**, *44*, 546–558. [CrossRef]
55. Mirjalili, S. Genetic algorithm. In *Evolutionary Algorithms and Neural Networks*; Springer: Cham, Switzerland, 2019; pp. 43–55.
56. Burke, E.K.; Hyde, M.; Kendall, G.; Ochoa, G.; Özcan, E.; Woodward, J.R. A Classification of Hyper-heuristic Approaches. In *Handbook of Metaheuristics*; Springer: Boston, MA, USA, 2010; pp. 449–468._15. [CrossRef]
57. Plata-González, L.F.; Amaya, I.; Ortiz-Bayliss, J.C.; Conant-Pablos, S.E.; Terashima-Marín, H.; Coello Coello, C.A. Evolutionary-based tailoring of synthetic instances for the Knapsack problem. *Soft Comput.* **2019**, *23*, 12711–12728. [CrossRef]
58. Epstein, S.L.; Petrovic, S. Learning a Mixture of Search Heuristics. In *Autonomous Search*; Hamadi, Y., Monfroy, É., Saubion, F., Eds.; Springer: Berlin/Heidelberg, Germany, 2012; pp. 97–127._5. [CrossRef]