Discovering Action Regions for Solving the Bin Packing Problem through Hyper-heuristics

Arturo Silva-Gálvez*, Jorge Orozco-Sanchez*, Erick Lara-Cárdenas*, José Carlos Ortiz-Bayliss*,
Ivan Amaya*, Jorge M. Cruz-Duarte*, and Hugo Terashima-Marín*
*Tecnologico de Monterrey, School of Engineering and Sciences
Email: artsg130994@gmail.com, jorgeorozcosanchez@gmail.com, a00398510@itesm.mx,
{jcobayliss, iamaya2, jorge.cruz, terashima}@tec.mx

***Abstract*—Hyper-heuristics represent an innovative method for solving hard combinatorial problems such as the Bin Packing Problem. In this work, we propose a solution model that incorporates insights from unsupervised learning to produce solvers that base their decisions on maximizing a reward. The solution model takes the form of a hyper-heuristic. As the search progresses, this strategy chooses among different heuristics, adapting the solution process to the current problem state under exploration. The proposed model relies on the k-means clustering algorithm to identify the centroids of what we define as "action regions" (regions where we can recommend one particular heuristic). To recommend a heuristic in such action regions, we use a simple yet useful reward-based approach that analyzes the performance of individual heuristics. Then, the hyper-heuristic (a collection of action regions) decides which heuristic is the most suitable one to apply given one specific problem state. The experimental setup was carried out on a total of 580 bin packing instances with promising results.**

***Index Terms*—Bin packing problem, Hyper-heuristics, Unsupervised learning.**

I. Introduction

Optimizing a problem implies finding the best solution based on a performance metric. Such a problem is usually represented through a mathematical model that relates variables whose optimum values must be found. It is conventional to express optimization problems as minimization ones since maximization problems can be usually transformed into the former. Moreover, and based on the nature of the variables, optimization problems can be of a continuous, discrete, or mixed nature. Also, as the problem involves more variables or constraints, it becomes more complex and harder to solve.

A combinatorial optimization problem that is commonly tackled in literature corresponds to the Bin Packing Problem (BPP). The chief reason for this is its inherent industrial applicability [1]–[4]. Naturally, this has led to plenty variants of this problem [5]–[7]. One of those is the well-known one-dimensional online version (1D-BPP), which belongs to the category of Cutting and Packing Problems [8]. A particular scenario where the 1D-BPP appears is in a production line where a fixed robot arm packages items into containers. In such a case, the items must be packed as they arrive, even if the whole production schedule is known in advance. In other words, the robot only knows the properties that describe each item (*e.g.,* the length), but of course, it has no chance to sort them before they reach the end of the line. Thus, items must be packed in the exact order given by the production schedule. This case of 1D-BPP is not trivial. Also, BPPs are considered as NP-Hard. Because of this, they have attracted the interest of scientists and practitioners in recent years [9]. One of the charms of BPPs is their versatility in describing many other radically different optimization problems from industry and academia [10].

In a similar fashion to the number of variants, the literature contains a plethora of methods for solving BPPs. Despite that, they can be organized into two broad categories. The first one focuses on producing new solving methods from scratch [11], [12], while the other centers on modifying or combining existing techniques to enhance performance [13], [14]. Even so, the idea of combining solvers for tackling a BPP is not new [15], [16]. For instance, Hyper-Heuristics (HHs) are methods that learn how to solve problems by managing a set of heuristics, or simple procedures [17], [18]. They have been successfully implemented in multiple combinatorial problems [19] and, particularly, in BPPs [20]–[22]. However, HHs require a proper training stage to ensure their robustness. For that purpose, Machine Learning (ML) and Evolutionary Algorithms (EAs) have been implemented with HHs [3], [19], [23]. Within them, reinforcement learning represents a set of strategies that have proved successful at improving hyper-heuristic models [24], [25]. For example, Özcan *et al.* used an approach based on reward and punishment for improving the performance of the great deluge HH on examination timetabling problems [26]. In fact, it is interesting that Alanazi and Per Lehre found that if the probability of improving a heuristic is less than 50%, this type of additive RL performs similar to a random mechanism [21]. Another example rests on the work of Cao and Tang, where the authors used probability matching. In doing so, they defined the reward assignment for the RL hyper-heuristic, improving the results of a multi-objective algorithm [27].

Hyper-heuristics work by mapping similar problem instances to one suitable heuristic. Regardless of the popularity

This work was supported in part by Consejo Nacional de Ciencia y Tecnología (CONACyT) Basic Science Project [Grant number 287479], and by ITESM Research Group with Strategic Focus in Intelligent Systems.

978-1-7281-2547-3/20/$31.00 ©2020 IEEE

of this approach, it is inherently limited by the way the HH defines the points in the problem state that will be used to produce a HH. In this regard, it seems worth exploring alternative approaches such as unsupervised learning (UL) and Reinforcement Learning (RL) algorithms, which may help overcome this limitation. So, this work focuses on HH methods that learn how to solve problems by managing a set of heuristics through a novel combination of UL and RL. Such heuristics are used based on some features that characterize the problem state. The objective of this model is to define a collection of action regions where one heuristic should be preferred over the others. So, we develop a procedure to discover those regions. Hence, this information is used to switch heuristics as the search progresses, thus enhancing the solving process. We implement this approach with 580 BPP instances, finding promising results.

The remainder of the paper is organized as follows. Section II presents the basic concept employed in this work. Then, Sect. III describes the proposed model. Section IV details and discusses the experiments conducted. Finally, Sect. V highlights the most relevant conclusions and future trends derived from this work.

II. Background

A. The Bin Packing Problem

The Bin Packing Problem (BPP) has been extensively studied in the literature [28]. A BPP requires finding assigning a list of items, with specific characteristics, into a minimum number of containers (bins) with defined constraints (see Sect. III-C) [29]. This model can represent other problems, such as cutting stock, knapsack, memory allocation, and vehicle loading [30].

B. Unsupervised and reinforcement learning

Machine Learning (ML) studies the techniques capable of enhancing a model based on their own experience [31]. It contains two categories that interest us: Unsupervised Learning (UL) and Reinforcement Learning (RL).

On the one hand, UL algorithms require no previous information to detect patterns from a dataset. Particularly, clustering represents one of the most popular approaches in UL. Clustering deals with the task of grouping items by common properties. Among their algorithms, k-means is probably the most popular one. It creates clusters and assigns the items to such groups based on a distance metric [32]. This method requires no explicit examples of what to learn. Instead, it needs features of the items that will be clustered. This algorithm aims to minimize the overall distance between items within a cluster and its centroid. This minimization process uses random initial locations for the centroids. Then, it iterates to improve them [33].

On the other hand, RL defines a strategy for learning how to behave through trial-and-error interactions with the environment. In other words, when learning with RL, the system perceives the problem state and takes action based on a policy. Using statistical techniques, RL estimates the usefulness of such an action in affecting the current environment state of the world where the agent is interacting.

C. Hyper-heuristics

Most simplistically, hyper-heuristics (HHs) can be defined as "heuristics to choose heuristics" [34]. HHs solve problems indirectly by working on the heuristic space rather than in the solution domain. When working with HHs, the problem state is mapped through a set of features, so the most suitable heuristic can be applied. One of the main challenges for selection HH models is to obtain a proper characterization of such a state. There are different ways to get such a mapping. Some include ML [35] and EAs [36]. However, the HH performance dramatically depends on the predictive power of the features considered. Although most of the studies on hyper-heuristics are empirical, there are some recent theoretical efforts that show the validate the effectiveness of hyper-heuristics [37]–[39].

III. Solution Model

Before describing our model, it is essential to detail the heuristics and instances we employ, as well as the characterization of such instances into action regions.

A. Heuristics

For this work, we have selected a set of useful yet straightforward heuristics [40]. All of them operate by identifying a suitable bin for packing the first item within a list. Whenever the heuristic packs an item, it is removed from the list. So, a new item takes the first position. Should a bin be full after packing the item, it is closed (it cannot accept more items due to its capacity). The process is repeated until packing all items. We now describe how these heuristics work.

- **First Fit (FF)** looks for the first open bin where the item fits. As soon as the bin is found, the item is packed there.
- **Best Fit (BF)** analyzes all open bins where the item fits, and packs it in the one that leaves the least free space (the bin where the wasted space is minimum).
- **Worst Fit (WF)** begins in the same fashion as BF, but it packs the item in the bin that leaves the most free space (the bin where the wasted space is maximum).
- **Almost Worst Fit (AWF)** is similar to WF, but it uses the bin where the wasted space is next to maximum.

It is important to remark that, should there be no open bins capable of storing the item, a new one is open and selected. Also, should there be ties, the lowest-numbered bin is preferred.

B. BPP Instances

All the instances considered for this work are synthetic. We briefly describe these instances below.

- **Training set** contains 100 instances of 20 items with sizes between 1 and 32 units. The bin capacity in these instances is 64 units. The instances in the training set are classified into four classes of 25 instances each. Each class represents a challenge for one particular heuristic.

All the heuristics will then experience problems when solving at least 25% of the instances in the training set. To generate this set, we employed the evolutionary-based BPP generator proposed by Amaya *et al.* [40].

- **Test set A** contains 400 instances similar to the ones used for training: 20 items with sizes between 1 and 32 units. The bin capacity of the instances in the test set is also 64 units. As in the training set, these instances are also classified into four classes. However, this time each class contains 100 instances. Again, each class represents a challenge for one particular heuristic. Then, the instances in the test set are classified as difficult cases (prefix 'Hard/') for BF, FF, WF, and AWF. To generate this set, we used the same generator from the training set.
- **Test set B** has the first class of instances introduced by Falkenauer [41]. These instances have uniformly distributed item sizes in the range from 120 to 1000, and a bin capacity of 150. These instances are included in this investigation to assess the performance of the hyper-heuristics on completely different instances than the ones contained in the training set and the test set A.

C. BPP Features

To characterize these instances, we used some straightforward features:

- **AVGL**. The average of the lengths of items yet to be packed.
- **STDL**. The standard deviation of the lengths of items yet to be packed.
- **VSMLR**. The proportion of "very small" items left to pack, *i.e.,* those which length is smaller than 25% of the maximum capacity of the bins.
- **SMLR**. The proportion of "small" items *i.e.,* those which length is larger than 25% but smaller than 50% of the maximum capacity of the bins.

The range of values AVGL and STDL can take lies in a different range compared to VSMLR and SMLR. Then, we decided to normalize AVGL and STDL, so all the features lie in the range 0 to 1, inclusive. To normalize the average and standard deviations of the length, we divided the values by 40 and 10, respectively. To exemplify how these features work, let us use the instance depicted in Fig. 1, with nine items of different sizes and a bin capacity of 12. We can calculate AVGL as the average length of all items divided by 40 (the maximum length considered in this work). Similarly, we can calculate STDL as the standard deviation of item lengths, divided by ten. Then, AVGL = 0.075 and STDL = 0.173. The value for SMLR = 0.333 since three out of nine items have a length in the range $(3, 6]$. Finally, six out of nine items present a length of, at most, three units. Hence, VSMLR = 0.666. We are aware that the numbers used for feature normalization are tailored for the instances in this work, and then, they are likely to change when dealing with other types of instances.

It is important to state that all these features are dynamic since they change as the search progresses. To estimate the quality of a solution, we use the average waste (AVGW) among all the open bins when all the items have been packed. Please note that we do not include features for "large" and "very large" items since instances within the training set lack these types of items. Then, we deem it unwise to include features missing from the training process.

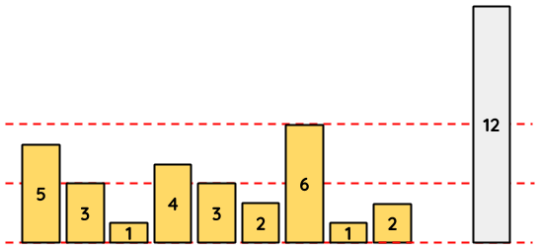

Fig. 1. Illustrative example of an instance with nine items of different sizes (in yellow) and a bin capacity of 12 (in gray).

D. Training phase

The model proposed in this document goes through two broad stages to produce a hyper-heuristic. First, it splits the problem space into action regions; then, it assigns one suitable heuristic to each region. The goal is to reduce the average waste of solutions. We now describe each stage in more detail.

In the first stage, the values of each feature are calculated for each instance in the training set. Then, k-means is used to cluster the instances based on the previously determined features. In doing so, the solution model generates a collection of action regions with no associated heuristics. Up to this point, the number of clusters is defined by the user. However, bear in mind that the solution model must include at least two clusters to avoid producing hyper-heuristics that inevitably mimic the behavior of standalone heuristics. The k-means algorithm relies on the Euclidean distance as a distance measure. In all cases, k-means runs for 100 iterations.

After the system generates the action regions, it calculates a score-matrix, which is the core of our proposal. Such a matrix contains one row per action region (*i.e.,* clusters from the previous step) and as many columns as heuristics are available. All the cells in the score-matrix are randomly initialized with small values in the range $[0.0, 0.1]$. Then, the solution model iterates to update the scores based on the performance of heuristics over the training set. Before explaining how scores are updated, we will first explain how to interpret a hyper-heuristic in our model.

As we stated before, the score-matrix will become a hyper-heuristic at the end of the training process. Hence, we can analyze how it works by studying an example. Figure 2 depicts a score-matrix with three action regions (characterized by the four features described in Sect. III-C). This score-matrix can choose one among the four heuristics described in Sect. III-A.

Let us consider that the next instance to be solved is currently characterized as [0.40, 0.58, 0.47, 0.53]. Since the action regions were calculated by using the Euclidean distance, we again use such a distance to identify the action region closest to said instance. In this case, the closest action region

Fig. 2. Illustrative example of the score-matrix with three action regions characterized by the four features described in Sect. III-C.

is R_2. So, we evaluate the second row of the score-matrix (the scores of the heuristics for that action region). Such scores are -0.044, 1.153, 0.048, and 0.016 for BF, FF, WF, and AWF, respectively. Since the largest score is 1.153, the BF heuristic is chosen. Because of this, the next item in the instance will be packed following BF. In packing an item, feature values change, and then, the system calculates the new distance to each action region. The process is repeated until the instance is solved, *i.e.,* no items remain for packing. At this point, we can calculate the average waste associated with the solution of the instance by using the hyper-heuristic (AVGWHH), which we use to estimate the reward. To do so, we also consider the average waste of a reference heuristic. We considered BF as such a baseline in this work, mainly because of its good overall performance. Therefore, the instance is solved anew with BF, and an average waste (AVGWBF) is calculated as a reference. We then calculate their difference, as $\Delta w = \text{AVGWBF} - \text{AVGWHH}$. The reward function in (1) uses the parameter α (set to a small value), and the Kronecker's Unit Sample Function $\delta[\cdot]$, to determine training speed. In this investigation, we set α to 0.01 based on preliminary tests.

$$r = \alpha \times (\delta[\Delta w] + \Delta w) \tag{1}$$

Equation (1) works on the idea of pushing the hyper-heuristic towards reducing the average waste concerning the reference heuristic (*i.e.,* BF). When Δw is negative, it means that the HH performed worse than BF. As a consequence, the decisions made by the hyper-heuristic should be corrected. Conversely, if Δw is positive, it means that the HH outperformed BF on the instance and that its decisions must be strengthened. The idea of setting the reward to α when the performance of both the HH and the heuristic is equal lies in the intuition that the reference heuristic is a competent one. Thus, matching its performance should also be considered a good result, and the decisions made by the HH should be rewarded accordingly.

Once the reward has been calculated, the system estimates the contribution of each action region to solve the instance. The HH keeps a record of the frequency of selection of the action regions. Then, the system multiplies the frequency of using each action region by the reward and adds it to the selected heuristic score for the corresponding region. By continuing our previous example, imagine that the HH depicted in Fig. 2 solved the first instance, obtaining an average waste of 1.23, while the average waste of BF was 1.02. So, Δw and r would be -0.21 and -0.0021, respectively. We need to analyze the frequencies of the rule used to determine the proportional effect of the update. Let us assume that R_2 and R_3 were employed in 65% and 35% of the decisions made by the hyper-heuristic, recommending BF and FF, respectively. The scores for BF in R_2 and FF in R_3 will be updated by adding -0.001365 and -0.000735 to these scores, respectively.

When the training set has been solved in a given number of epochs, the training is over, and the HH is ready to be used to solve unseen instances. From this point onward, no further updates are made to the score-matrix.

IV. Experiments and Results

In this investigation, we conducted three experiments. The first one deals with producing various hyper-heuristics and assessing their overall performance against single heuristics. The second experiment deepens into the decisions made by some selected hyper-heuristics by analyzing their behavior on the four classes of instances identified in the test set A. The last experiment evaluates the generalization power of the hyper-heuristics by solving unseen instances with properties different from the ones used for training.

A. Overall Hyper-heuristic Performance

In this experiment, we evaluated the hyper-heuristic performance when varying the number of action regions used for discriminating instances and choosing a suitable heuristic. For this purpose, we defined four configurations (*i.e.,* three, five, seven, and nine regions), and produced ten HHs for each case. Besides, we employed the four available heuristics described in Sect. III-A. The scores in the HH were updated for five epochs to produce each hyper-heuristic. In all cases, the performance metric used was the average waste (AVGW) on the test instances. Fig. 3 depicts the results of this experiment.

Before analyzing the hyper-heuristics produced, we need to describe our naming conventions. Each hyper-heuristic produced in this work contains the prefix 'HH', followed by two digits that indicate the number of action regions in the hyper-heuristic. After a dash ('-'), we also include two digits to indicate the replica number. For example, HH05-02 indicates the second hyper-heuristic produced by using five action regions.

By using three action regions, the solution model produced hyper-heuristics that replicate the exact behavior of BF. Then,

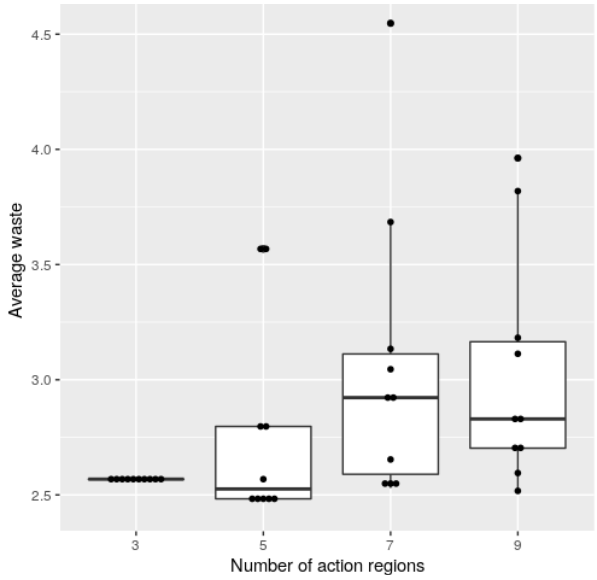

Fig. 3. Average waste distribution (10 runs) achieved by our proposed hyper-heuristic model when considering four different numbers of action regions for discriminating instances.

HH03-01 to HH03-10 were discarded from further analysis since they replicate BF. Among the remaining hyper-heuristics, the ones generated with five action regions show the most promising performance. The fact that increasing the number of action regions decreases the performance of the HHs makes sense since it is more difficult for the model to adjust all the scores and produce a reliable method capable of generalizing to unseen instances. Although this claim seems valid at this point, we are aware that testing more configurations regarding the number of action regions is required to validate it properly.

When we compare the hyper-heuristics produced by using five action regions against the performance of the standalone heuristics, we observe that the ten HHs reduce the average waste produced by FF, WF, and AWF. When dealing with BF, only half of the HHs reduced the average waste produced by BF. These hyper-heuristics, HH05-02, HH05-04, HH05-06, HH05-09, and HH05-10, improved the results of BF by 3.32%. This percentage represents an overall saving of around 34 units of capacity to BF, the best heuristic in this case. Although these reductions in the average waste are valuable in practice, we found no statistical evidence that supports that any of the hyper-heuristics is better than BF. By taking a more in-depth look at these five HHs, we observed that they have different values in their score-matrices, although they exhibit the same behavior on the test set A. This situation suggests different ways of combining heuristics, which may lead to similar solving processes.

B. Switching Heuristics to Improve the Performance

Aiming to analyze the behavior of the hyper-heuristics produced with five action regions in more detail, we studied the performance of two competent HHs by the class of instance in the test set A. We selected HH05-02 (which also represents the behavior of HH05-04, HH05-06, HH05-09, and HH05-10) and HH05-03 (which also represents the behavior of HH05-05) for this analysis. Fig. 4 illustrates the results of this comparison.

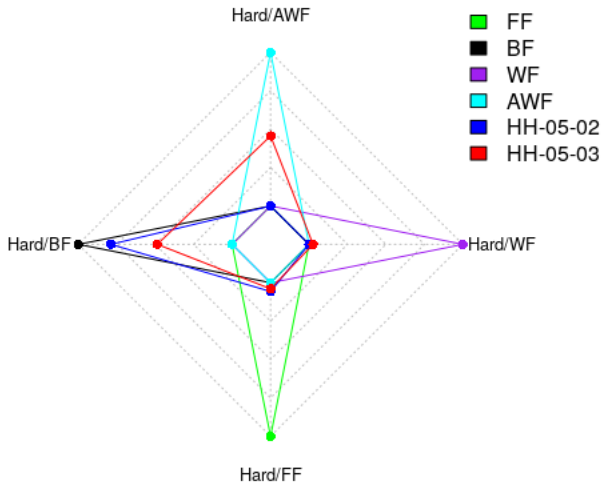

Fig. 4. Average waste distribution given by heuristics and hyper-heuristics HH05-02 and HH05-03 on test set A. Data are grouped by instance class.

From Fig. 4, we can observe that all the heuristics have one class that is difficult to solve, as indicated in Sec. III-B. This behavior is alleviated by using the hyper-heuristics. On the one hand, HH05-02, as well as the other four hyper-heuristics with the same behavior, is capable of reducing the average waste produced by BF in the test set due to its outstanding performance in the classes Hard/FF, Hard/WF, and Hard/AWF. At the same time, HH05-02 also reduces the waste in class Hard/BF, which is possible because it introduces different heuristics throughout the solving process. On the other hand, HH05-03 is a much better performer in class Hard/BF than HH05-02. Unfortunately, it pays the price of some poor results in the class Hard/AWF, which decreases its overall performance. This is the reason why HH05-03 cannot reduce the average waste per instance that BF obtains in this set.

Regarding some statistics of the internal behavior of these two hyper-heuristics, we can analyze the frequency in which the HHs use each heuristic. To solve the test set A, HH05-02 utilizes only FF and BF, in 13.56% and 86.4% of the decisions, respectively. We can then assume that by combining only FF and BF, it is likely to reduce the average waste in the class Hard/BF without affecting the performance in other classes. Otherwise, HH05-03 avoids using FF and chooses BF, WF, and AWF in 48.11%, 13.58%, and 38.31% of the decisions. Although HH05-03 performs much better in the class Hard/BF, the excessive use of AWF pays the price when solving the instances from the class Hard/AWF.

C. Testing on Instances Unrelated to the Training

We employed the ten hyper-heuristics produced with five action regions to solve the test set B. Test set B is not balanced,

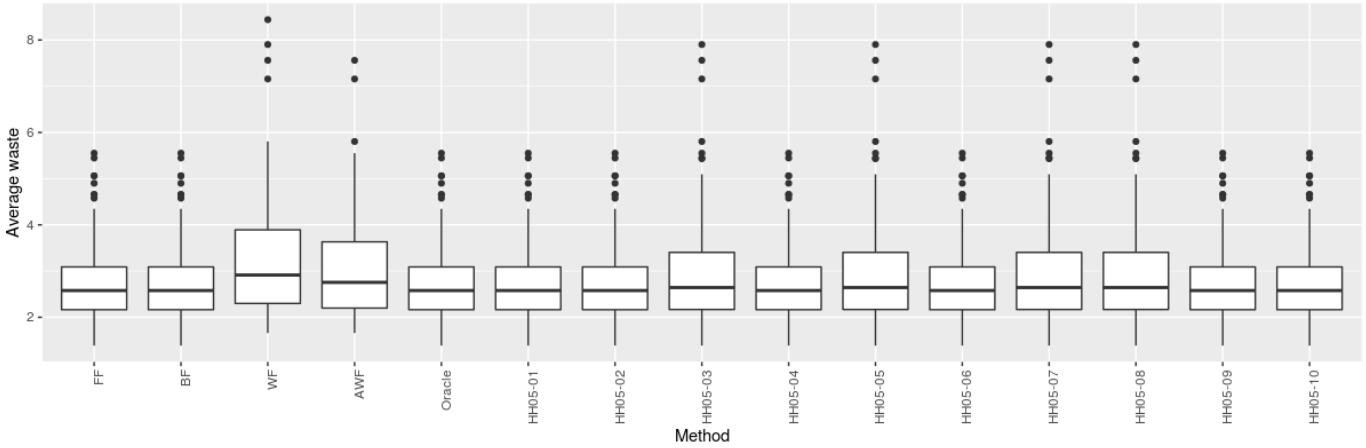

Fig. 5. Waste distribution given by heuristics and hyper-heuristics on test set B.

and the instances are larger than the ones used for training and testing in the previous experiments. Hence, this represents an opportunity to assess the performance of these HHs in a more realistic scenario.

By analyzing the test set B, we observed that the best standalone heuristics are FF and BF, replicating the Oracle. These heuristics, as well as the Oracle, produce an average waste of 2.813. WF and AWF are no match for them since they produce an average waste of 3.337 and 3.145, respectively. When HH05-01 to HH05-10 are used to solve the test set B, their performance is quite acceptable since they were not trained for these new types of instances. In the test set B, six out of ten hyper-heuristics behave as competent as the Oracle (Fig. 5). In fact, those hyper-heuristics replicate the exact behavior of the Oracle; which is also the behavior of FF and BF in this case. The four remaining hyper-heuristics are not as good as FF and BF, but they are better than WF and AWF. Then, we evidenced that the HHs produced with the proposed approach may be useful for solving instances in which properties differ from those used for training them.

V. Conclusion

This study proposed a Hyper-Heuristic (HH) model based on a combination of Unsupervised Learning (particularly, k-means) and a reward-based strategy. Our model generates action regions whose centroids are found by k-means. Then, it assigns one suitable heuristic to each region. When the hyper-heuristic solves one instance, it identifies the action region that influences such an instance and uses the corresponding heuristic. Although simple, this method proved useful. The resulting HHs were robust enough to behave competitively, even on unseen instances of a different nature.

Nevertheless, the scope of this study was set to exploring the feasibility of the model and its behavior when tackling the 1D-BPP. It shall be interesting to delve more deeply into the effect of instance features and select the best ones for the model. Moreover, further improvements can be made to this model by choosing a more robust reward function that better guides the model and delivers better labeling of the action regions.

An advantage of this model is its limited consumption of resources. This model requires a few iterations to produce a HH. In fact, the solution model needed but a few seconds to produce a HH in this investigation. Of course, increasing the number of instances, or the number of items within them will surely increase the training time. We are aware that a more detailed study on the running time analysis of the model would be advisable in the future. Regarding the execution time of the HHs, once they are produced, their times are slightly longer than the ones of the single heuristics. The additional time is because the HHs must execute some additional calculations to determine the heuristic to apply. In the end, this additional time is negligible.

An essential part of our model is that it can be easily extended to other problem domains with few modifications. Specifically, to adapt this model to another domain, we would need to define a feasible reward function and to select both a set of heuristics and features to characterize the instances. We think that, by studying the performance of the model on other domains, we could learn more about the benefits and drawbacks of the model and validate its effectiveness.

Finally, we are interested in taking this solution model to areas where HHs have been scarcely explored. For example, a compelling application rests in the adaptation of the model for improving the automatic customization of metaheuristics used for solving continuous engineering problems. Here, search operators can be extracted from such metaheuristics and be regarded as elements, and features such as computational burden and average performance may be considered for characterizing them.

References

[1] M. Delorme, M. Iori, and S. Martello, "Bin packing and cutting stock problems: Mathematical models and exact algorithms," *European Journal of Operational Research*, vol. 255, no. 1, pp. 1 – 20, 2016.

[2] J. H. Drake, J. Swan, G. Neumann, and E. Özcan, "Sparse, Continuous Policy Representations for Uniform Online Bin Packing via Regression of Interpolants," in *Evolutionary Computation in Combinatorial Optimization. EvoCOP 2017. Lecture Notes in Computer Science*, pp. 189–200, Springer, 2017.

[3] J. C. Gomez and H. Terashima-Marín, "Evolutionary hyper-heuristics for tackling bi-objective 2D bin packing problems," *Genetic Programming and Evolvable Machines*, mar 2017.

[4] H. Hu, X. Zhang, X. Yan, L. Wang, and Y. Xu, "Solving a new 3d bin packing problem with deep reinforcement learning method," *CoRR*, vol. abs/1708.05930, 2017.

[5] H. I. Christensen, A. Khan, S. Pokutta, and P. Tetali, "Multidimensional bin packing and other related problems: A survey," 2016.

[6] M. Delorme, M. Iori, and S. Martello, "Bpplib: a library for bin packing and cutting stock problems," *Optimization Letters*, vol. 12, no. 2, pp. 235–250, 2018.

[7] M. S. Levin, "Towards bin packing (preliminary problem survey, models with multiset estimates)," *arXiv preprint arXiv:1605.07574*, 2016.

[8] M. Garraffa, F. Salassa, W. Vancroonenburg, G. Vanden Berghe, and T. Wauters, "The one-dimensional cutting stock problem with sequence-dependent cut losses," *International Transactions in Operational Research*, vol. 23, no. 1-2, pp. 5–24, 2016.

[9] J. Balogh, J. Békési, G. Dósa, L. Epstein, H. Kellerer, and Z. Tuza, "Online Results for Black and White Bin Packing," *Theory of Computing Systems*, vol. 56, pp. 137–155, jan 2015.

[10] T. Koch, T. Achterberg, E. Andersen, O. Bastert, T. Berthold, R. E. Bixby, E. Danna, G. Gamrath, A. M. Gleixner, S. Heinz, A. Lodi, H. Mittelmann, T. Ralphs, D. Salvagnin, D. E. Steffy, and K. Wolter, "MIPLIB 2010," *Mathematical Programming Computation*, vol. 3, no. 2, pp. 103–163, 2011.

[11] M. Abdel-Basset, G. Manogaran, L. Abdel-Fatah, and S. Mirjalili, "An improved nature inspired meta-heuristic algorithm for 1-D bin packing problems," *Personal and Ubiquitous Computing*, vol. 22, pp. 1117–1132, oct 2018.

[12] M. Haouari and M. Serairi, "Heuristics for the variable sized bin-packing problem," *Computers & Operations Research*, vol. 36, pp. 2877–2884, oct 2009.

[13] S. Asta, E. Özcan, and A. J. Parkes, "CHAMP: Creating heuristics via many parameters for online bin packing," *Expert Systems with Applications*, vol. 63, pp. 208–221, nov 2016.

[14] K. Sim, E. Hart, and B. Paechter, "A Lifelong Learning Hyper-heuristic Method for Bin Packing," *Evolutionary Computation*, vol. 23, pp. 37–67, mar 2015.

[15] E. López-Camacho, H. Terashima-Marin, P. Ross, and G. Ochoa, "A unified hyper-heuristic framework for solving bin packing problems," *Expert Systems with Applications*, vol. 41, no. 15, pp. 6876 – 6889, 2014.

[16] K. Sim, E. Hart, and B. Paechter, "A hyper-heuristic classifier for one dimensional bin packing problems: Improving classification accuracy by attribute evolution," in *Parallel Problem Solving from Nature - PPSN XII* (C. A. C. Coello, V. Cutello, K. Deb, S. Forrest, G. Nicosia, and M. Pavone, eds.), (Berlin, Heidelberg), pp. 348–357, Springer Berlin Heidelberg, 2012.

[17] E. K. Burke, S. Petrovic, and R. Qu, "Case-based heuristic selection for timetabling problems," *Journal of Scheduling*, vol. 9, pp. 115–132, apr 2006.

[18] J. A. Soria-Alcaraz, G. Ochoa, M. A. Sotelo-Figeroa, and E. K. Burke, "A methodology for determining an effective subset of heuristics in selection hyper-heuristics," *European Journal of Operational Research*, vol. 260, no. 3, pp. 972–983, 2017.

[19] J. H. Drake, A. Kheiri, E. Özcan, and E. K. Burke, "Recent advances in selection hyper-heuristics," *European Journal of Operational Research*, 2019.

[20] N. Pillay and R. Qu, "Packing problems," in *Hyper-Heuristics: Theory and Applications*, pp. 67–73, Springer, 2018.

[21] F. Alanazi and K. P. Lehre, "Limits to learning in reinforcement learning hyper-heuristics," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 9595, pp. 170–185, Springer Verlag, 2016.

[22] 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.

[23] W. Li, E. Özcan, and R. John, "A learning automata-based multiobjective hyper-heuristic," *IEEE Transactions on Evolutionary Computation*, vol. 23, no. 1, pp. 59–73, 2017.

[24] S. S. Choong, L.-P. Wong, and C. P. Lim, "Automatic design of hyper-heuristic based on reinforcement learning," *Information Sciences*, vol. 436, pp. 89–107, 2018.

[25] M. Montazeri, "Hyper-heuristic image enhancement (hhie): A reinforcement learning method for image contrast," *Advanced Computing and Intelligent Engineering: Proceedings of ICACIE 2018, Volume 1*, vol. 1082, p. 363, 2020.

[26] E. Özcan, M. Misir, G. Ochoa, and E. K. Burke, "A Reinforcement Learning - Great-Deluge Hyper-Heuristic for Examination Timetabling," *International Journal of Applied Metaheuristic Computing*, vol. 1, pp. 39–59, feb 2011.

[27] P. Cao and J. Tang, "A reinforcement learning hyper-heuristic in multi-objective single point search with application to structural fault identification," *CoRR*, vol. abs/1812.07958, 2018.

[28] E. Burke, G. Kendall, J. Newall, E. Hart, P. Ross, and S. Schulenburg, "Hyper-Heuristics: An Emerging Direction in Modern Search Technology," in *Handbook of Metaheuristics*, pp. 457–474, Boston: Kluwer Academic Publishers, 2003.

[29] E. Lopez-Camacho and H. Terashima-Marin, "Evolving feature selection for characterizing and solving the 1D and 2D bin packing problem," in *2013 IEEE Congress on Evolutionary Computation, CEC 2013*, pp. 2094–2101, IEEE, jun 2013.

[30] G. Wäscher, H. Haußner, and H. Schumann, "An improved typology of cutting and packing problems," *European Journal of Operational Research*, vol. 183, pp. 1109–1130, dec 2007.

[31] E. Alpaydin, *Introduction to machine learning*. MIT press, 2020.

[32] S. Z. Selim and M. A. Ismail, "K-means-type algorithms: A generalized convergence theorem and characterization of local optimality," *IEEE Transactions on pattern analysis and machine intelligence*, vol. PAMI-6, no. 1, pp. 81–87, 1984.

[33] A. Likas, N. Vlassis, and J. J. Verbeek, "The global k-means clustering algorithm," *Pattern recognition*, vol. 36, no. 2, pp. 451–461, 2003.

[34] E. K. Burke, M. Gendreau, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, and R. Qu, "Hyper-heuristics: a survey of the state of the art," *Journal of the Operational Research Society*, vol. 64, pp. 1695–1724, 2013.

[35] J. C. Ortiz-Bayliss, H. Terashima-Marín, and S. E. Conant-Pablos, "Learning vector quantization for variable ordering in constraint satisfaction problems," *Pattern Recognition Letters*, vol. 34, pp. 423–432, 3 2013.

[36] T. N. Ferreira, J. A. P. Lima, A. Strickler, J. N. Kuk, S. R. Vergilio, and A. Pozo, "Hyper-Heuristic Based Product Selection for Software Product Line Testing," *IEEE Computational Intelligence Magazine*, vol. 12, pp. 34–45, 4 2017.

[37] C. Qian, K. Tang, and Z.-H. Zhou, "Selection hyper-heuristics can provably be helpful in evolutionary multi-objective optimization," in *Parallel Problem Solving from Nature – PPSN XIV* (J. Handl, E. Hart, P. R. Lewis, M. López-Ibáñez, G. Ochoa, and B. Paechter, eds.), (Cham), pp. 835–846, Springer International Publishing, 2016.

[38] A. Lissovoi, P. S. Oliveto, and J. A. Warwicker, "On the time complexity of algorithm selection hyper-heuristics for multimodal optimisation," in *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*, pp. 2322–2329, AAAI Press, 2019.

[39] J. C. Ortiz-Bayliss, H. Terashima-Marín, E. Ozcan, A. J. Parkes, and S. E. Conant-Pablos, "Exploring heuristic interactions in constraint satisfaction problems: A closer look at the hyper-heuristic space," in *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2013, Cancun, Mexico, June 20-23, 2013*, pp. 3307–3314, IEEE, 2013.

[40] I. Amaya, J. C. Ortiz-Bayliss, S. E. Conant-Pablos, H. Terashima-Marín, and C. A. Coello Coello, "Tailoring instances of the 1d bin packing problem for assessing strengths and weaknesses of its solvers," in *Parallel Problem Solving from Nature – PPSN XV* (A. Auger, C. M. Fonseca, N. Lourenço, P. Machado, L. Paquete, and D. Whitley, eds.), (Cham), pp. 373–384, Springer International Publishing, 2018.

[41] E. Falkenauer, "A hybrid grouping genetic algorithm for bin packing," *Journal of Heuristics*, vol. 2, no. 1, pp. 5–30, 1996.