An Exploratory Study on Machine-Learning-Based Hyper-heuristics for the Knapsack Problem

José Eduardo Zárate-Aranda[1 [0000−0001−7322−4126]] and
José C. Ortiz-Bayliss[1 [0000−0003−3408−2166]]

Tecnologico de Monterrey, School of Engineering and Sciences,
Monterrey 64849, Mexico
{A01630299, jcobayliss}@tec.mx

Abstract. Hyper-heuristics have risen as a recurrent method to solve combinatorial optimization problems since they use a set of heuristics selectively according to the problem state. Although many ideas have been developed to produce hyper-heuristics, a recent trend involves treating the heuristic selection problem as a classification one. This allows the introduction of machine learning elements into the hyper-heuristic process. This work explores creating hyper-heuristics using Machine Learning classifiers to solve the Knapsack Problem, a fascinating and well-studied combinatorial problem. We propose two approaches to these hyper-heuristics: a dynamical approach, where the hyper-heuristic may change heuristics throughout the whole solving process, and a static approach, where the hyper-heuristic makes one initial choice of heuristic and no further changes are allowed. Our results confirm that hyper-heuristics powered by machine learning techniques can deal with the Knapsack problem and obtain competent results. Besides, we also observed a clear superiority in the performance of the hyper-heuristics running under the static approach concerning the dynamic counterpart.

Keywords: Machine Learning · Heuristics · Hyper-heuristics · Knapsack Problem.

1 Introduction

When solving an optimization problem, we have to make choices. One primary choice concerns how much we are willing to 'invest' to solve such a problem and our expectations about the solutions. For example, we may be tempted to invest a lot and use tailor-made algorithms so that the solution for one particular instance of the problem is solved in the best possible way. In other cases, we could just apply a general solver that consumes fewer resources and be satisfied with a valid solution of acceptable quality. Although there are many scenarios to explore, in this work, we focus on the one where we aim for solutions of acceptable quality for various instances of the problem. We are not interested

in the best possible solver for one particular instance but a method capable of performing "well enough" over various types of instances of a problem.

Hyper-heuristics, or "heuristics to choose heuristics" [5], have proven suitable for the scenario described before since they create a mapping between problem states (characterized by some problem features) and suitable heuristics. In other words, they intelligently apply existing heuristics to improve the quality of the solutions. This work proposes developing hyper-heuristics to solve the Knapsack Problem. This widely studied combinatorial optimization problem remains relevant nowadays due to its many applications in different fields, such as production planning, health care, financial modeling, and computer networks [1, 4, 6]. Although many versions of the Knapsack Problem exist, this work explores its most straightforward version, the 0/1 KP, where each item has only two options: to be packed or left out of the solution set [1]. Since we will only refer to this version of the problem, we will simply refer to this problem as KP. The KP consists of (1) items, each with a profit and weight, and (2) a knapsack with a particular capacity. Solving the problem requires finding a subset of items that maximize the overall profit without breaking the knapsack's capacity.

As described before, we propose using a hyper-heuristic (HH) to solve the KP. The hyper-heuristic must learn when to apply one heuristic based on the problem state under exploration. Various strategies, such as ant colony optimization [8], genetic programming [7], fuzzy logic [14], and reinforcement learning [21], have already been used to generate hyper-heuristics for solving the KP or some of its variations. However, an almost unexplored trend in hyper-heuristics, used in other problem domains, treats the heuristic selection problem as a classification one. Under this perspective, the hyper-heuristic assigns a set of features to a suitable class, which is nothing but a heuristic [15,16]. To the authors' knowledge, this approach has yet to be explored for the KP.

The remainder of this document is organized as follows. Section 2 mentions the existing solution methods for the KP. Section 3 describes the solution model and its operation. Afterward, the experiments and results can be found in Sect. 4. Finally, Sect. 5 presents the conclusion and some ideas for future work derived.

2 Background and Related Work

We start this section by briefly defining the KP. The KP contains a set of items. Each item is associated with a weight and profit. When solving the KP, the objective is to select a subset of items that maximizes the total profit without exceeding the knapsack's capacity. The KP is challenging and recurrent in Combinatorial Optimization (CO) [19], and the literature is rich in methods for solving such a problem. We can broadly classify the solving methods as exact and approximation ones. Exact methods guarantee to find the optimal solution if enough time is provided. However, this is only possible in some practical (and rather) cases. Some examples of exact methods include: linear programming [2, 12], branch and bound approach [10, 23], dynamic programming [3, 9]. As mentioned earlier, these methods can offer optimal solutions only

to small-scale instances, which limits their application to practical cases. Regarding approximation methods, we can mention heuristics, metaheuristics, and hyper-heuristics. Heuristics are usually known as "rules of thumb" that use little computational resources to find an acceptable solution. Heuristics are problem-dependent; they cannot be used generically for different problems. Metaheuristics work on a higher level than simple heuristics in different problem domains, unlike simple heuristics [24]. Hyper-heuristics, on the other hand, work at an even higher level [14, 17, 20]. Instead of exploring the solution space, they select or generate low-level heuristics suitable for the current problem being solved. Those heuristics will be responsible for solving the problem.

3 Solution Approach

As mentioned before, our solution proposal uses some popular classifiers to produce HHs that map the problem state of a KP instance to a suitable heuristic to apply. Before describing how the hyper-heuristic model works, we introduce some vital elements of such a proposal.

3.1 The Features

This work characterizes the instances using three straightforward features. It is relevant to note that these features are dynamic. Every time an item is packed, the values for these features change for what remains of the instance.

Profit (P). This is the average profit of all the remaining items in the instance divided by the maximum profit among all the remaining items.

Weight (W). This represents the average weight of all the remaining items in the instance divided by the maximum weight among all the remaining items.

Correlation (C). It estimates the correlation between the profits and weights of the remaining items in the instance. Since correlation is calculated using the Pearson correlation coefficient, which lies in the range $[-1, 1]$, we divide it by two and add 0.5. This adjusts the range of this feature to $[0, 1]$, as with the first two features described.

3.2 The Heuristics

As in other heuristic-based works that solve the KP, deciding which item to pack next is done by applying heuristics iteratively, one item at a time. For this purpose, we have considered four heuristics, which are briefly described as follows:

Default (DEF). DEF packs the items following the order established in which they appear in the instance (no additional ordering is conducted on the items).

Minimum Weight (MINW). MINW prefers the item with the smallest weight. Then, MinW prioritizes lighter items, allowing the selection of the most lightweight items that still fit within the knapsack's capacity.

Maximum Profit (MAXP). MAXP chooses the item with the largest profit. It uses a greedy approach to fill the knapsack.

Maximum Profit per Weight (MAXPW). MAXPW prioritizes the profit-to-weight ratio. It computes each item's profit-to-weight ratio and selects the items in decreasing order. So, the objects with the highest profit-to-weight ratio are packed first.

3.3 The KP instances

In this work, we used synthetic KP instances produced with the algorithm proposed by Plata et al. [18]. The instances are grouped into two sets: training and testing [1]. The training set contains 100 instances, while the testing set has 400. All the instances in this work have 100 items and a maximum capacity of 64 weight units. We acknowledge that these sets may seem arbitrarily chosen or small to achieve conclusive results. However, the main characteristic of the sets produced for this work is that they are 'balanced.' Thus, no heuristic is the best option when considering all the instances throughout the sets. Each heuristic is the best performer in 25% of the instances of each set. So, in each set, the instances are also grouped in four subsets: SET_DEF, SET_MAXP, SET_MAXPW, and SET_MINW, where the best performers are DEF, MAXP, MAXPW, and MINW, respectively. The rationale behind this distribution is that no single heuristic outperforms the other when considering the whole training or testing set. This makes this scenario suitable to test hyper-heuristic performance.

3.4 Performance Metrics

As in other studies where the KP is studied, the profit of the solution is used as a quality metric. The larger the profit (without exceeding the knapsack's capacity), the better the solution. To allow the comparison of various models, and address the fact that some instances may result in larger profits than others, we have normalized the results per instance. So, for each instance, the best method obtains a normalized profit of 1 and the worst method, 0. Normalization is calculated as $\vec{z} = \frac{\vec{x} - \min(\vec{x})}{\max(\vec{x}) - \min(\vec{x})}$, where $\vec{x}$ is a vector that contains the profits obtained by different methods for a particular KP instance.

We will also use the success rate to evaluate the methods' performance along with the normalized profit. The success rate indicates the percentage of instances where a method obtains the best possible result among all analyzed methods. Although this seems similar to the concept of accuracy, commonly used in classification scenarios, the rationale behind them is different. For example, we cannot

[1] These instances are publicly available at `https://bit.ly/3wvxPly`

use the accuracy on our four heuristics since they do not perform any classification process. Then, we calculate the proportion of instances where these heuristics obtain the best result, and we use it to compare its performance against the remaining methods.

3.5 Using Machine Learning to Power Hyper-heuristics

This work explores using ML techniques to produce hyper-heuristics that solve the KP. To do so, we assume that choosing the most suitable heuristic at a given moment can be seen as a classification problem. Under this perspective, the hyper-heuristic is a classifier that chooses the correct class (a heuristic) given a particular input vector (the normalized features that characterize the problem state).

We propose two ways to apply such a classifier in this context. We will refer to them as static and dynamic, and they work as follows.

Dynamic. A hyper-heuristic implemented under the dynamic approach can use different heuristics when solving an instance. So, for each item, the hyper-heuristic uses the problem characterization to choose a heuristic. When the hyper-heuristic applies a heuristic to the instance (an item is packed), the number of items reduces. As a consequence, the problem characterization also changes. The hyper-heuristic process repeats until it packs the last item, deciding which heuristic to apply for each item to pack throughout the solving process. The dynamic approach is the most common way hyper-heuristics have been implemented in the literature [11, 13, 22].

Static. A hyper-heuristic from the static approach decides which heuristic to use only once per instance when it selects the first item to pack. The selected heuristic is used repeatedly until the instance is solved (no further changes in the heuristic are allowed). This means that when the hyper-heuristic faces an instance, it only uses the problem characterization to decide which heuristic to apply for the initial state. Later, the problem characterization becomes useless since the hyper-heuristic will ignore it. Most of the works that have used ML algorithms to produce hyper-heuristics implement this approach [16].

4 Experimental Results

We used the training set described in Section 3.3 to train the classifiers (the hyper-heuristics). We produced five hyper-heuristics using five different machine learning techniques: k Nearest Neighbors (KNN), Logistic Regression (LR), Multi-layer Perceptrons (MLP), Random Forests (RF), and Support Vector Machines (SVM). Thus, the five hyper-heuristics considered for the analysis are KNN-HH, LR-HH, MLPC-HH, RF-HH, and SVM-HH, where the prefix indicates the ML classifier used in each case. In all cases, we used Python's Scikit-Learn to implement the classifiers. For simplicity, we used the default configuration for each algorithm.

Once the hyper-heuristics were trained, we used them to solve the instances in the test set. Thus, all the results presented from this point on correspond exclusively to the testing set.

4.1 Analysis of the Success Rate

Before analyzing the success rate, imagine a hypothetical solver who always makes the right heuristic choice. Let this method be called ORACLE. As expected, the ORACLE will always make the right choice, and its success rate will be 100%. Of course, constructing the ORACLE is only possible after running all the solvers on a particular instance and selecting the best outcome. In this case, we have solved the instances in the test set using all the heuristics. Hence, we can construct the ORACLE and use it for comparison purposes.

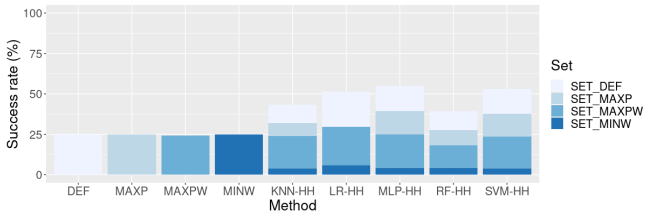

Fig. 1: Success rate of the methods under study in the testing set (the hyper-heuristics work using the dynamic approach).

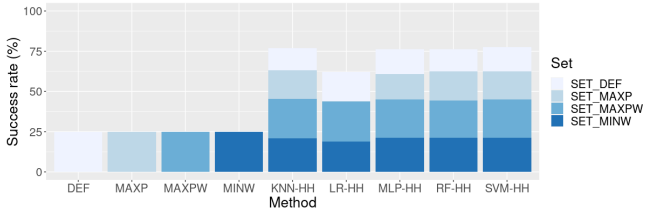

Fig. 2: Success rate of the methods under study in the testing set (the hyper-heuristics work using the static approach).

Figures 1 and 2 show the success rate of the methods studied in this work for the dynamic and static approaches, respectively. For a deeper analysis, we have shown each set's contribution to each method's success rate.

As mentioned in Section 3.3, the training and testing sets are composed of subsets where one heuristic is the best performer. This explains why the success rate of each of the heuristics is 25% since they obtain the best result

only in their corresponding set. Regarding the dynamic approach (Figure 1), we observed that, as expected, the hyper-heuristics obtain a better success rate than the heuristics. The best hyper-heuristic MLP-HH obtains the best result in a little more than half the instances in the test set. These results support the idea that using hyper-heuristics to solve the KP is good. When analyzing the hyper-heuristics under the static approach (Figure 2), we observed a significant improvement in terms of the success rate concerning the hyper-heuristics running under the dynamic approach. Of course, the success rate of the heuristics remains the same since it does not depend on the approach used by the hyper-heuristic. Regardless of the outstanding behavior of the hyper-heuristics running under the static approach, they all obtain a success rate below 80%. Then, there is plenty of room for improvement in this regard.

4.2 Analysis of the Normalized Profit

In this section, we compare the performance of the methods considering the normalized profit. The higher the normalized profit, the better the method's performance. Figure 3 depicts the distribution of normalized profits of the heuristics and hyper-heuristics produced when used to solve the testing set under the dynamic approach.

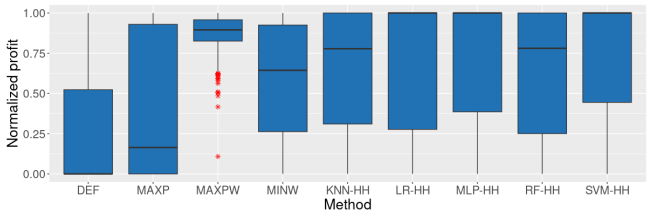

Fig. 3: Normalized profit of the methods under study in the testing set (the hyper-heuristics work using the dynamic approach).

The hyper-heuristics under the dynamic approach reduce their performance significantly. Although the medians of LR-HH, MLP-HH, and SVM-HH are larger than that of MAXPW (the best heuristic in this set), the variance in the results is larger than that of MAXPW. Comparing the means using a one-tail Wilcoxon's test fails to find enough evidence that supports the real median of any of these three hyper-heuristics (LR-HH, MLP-HH, and SVM-HH) is larger than MAXPW. In these tests, H_0 states that the real median of the normalized profits of MAXPW is larger or equal to that of the hyper-heuristic in turn. Conversely, H_1 states that the real median of the normalized profit of MAXPW is smaller than the one of the heuristics in turn. So, small p-values suggest that the corresponding hyper-heuristic outperforms MAXPW regarding the normalized profit. So, the statistical evidence is overwhelming against the idea that

any of these hyper-heuristics has a real median larger than the one of MAXPW. In all the cases, the p-values of the tests for LR-HH, MLP-HH, and SVM-HH were 0.6313, 0.1791, and 0.2796, respectively. In simpler words, although LR-HH, MLP-HH, and SVM-HH are better than MAXPW in the testing set (based on the median normalized profit), there is no statistical evidence that supports that such hyper-heuristics are actually better than MAXPW.

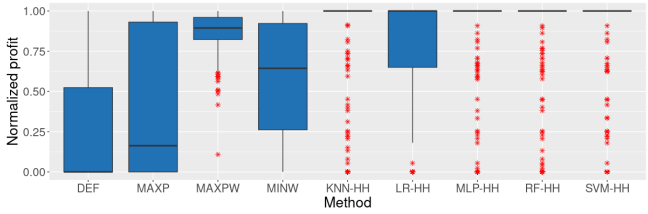

Fig. 4: Normalized profit of the methods under study in the testing set (the hyper-heuristics work using the static approach).

In the static approach, as described before, a decision is made only when packing the first item in the instance. The heuristic chosen to pack the first item is used to pack all the remaining items in the instance. Figure 4 depicts the distribution of normalized profits of the heuristics and hyper-heuristics produced when used to solve the testing set under the static approach. Among the four heuristics, the best individual performer is MAXPW. However, the five hyper-heuristics exhibit an outstanding behavior. We observe that the medians of the five hyper-heuristics are larger than that of MAXPW. When comparing the means using one-tail Wilcoxon's tests against MAXPW, similar to the one conducted for the dynamic scenario, the p-values obtained from the Wilcoxon's tests between MAXPW and each of the hyper-heuristics were 2.31×10^{-16} for LR-HH and 2.2×10^{-16} for the other hyper-heuristics. As observed, in all cases, the statistical evidence supports the idea that using ML techniques (such as the ones considered in this work) to produce hyper-heuristics for solving the KP is suitable and improves the results obtained by single heuristics.

5 Conclusion and Future Work

This work explores using ML methods to produce hyper-heuristics for solving the KP. These hyper-heuristics work in two different modes: dynamic and static. As previously stated, when using the dynamic approach, the solver selects a heuristic whenever the hyper-heuristic must pack an item. Then, we obtain a solution by applying a sequence of heuristics, not a single one. This is different from what happens in the static approach, where only one choice is made. Although the dynamic approach remains the most used regarding hyper-heuristics, our

results suggest that using the static approach may be a better idea since hyper-heuristics working on such an approach (only choosing a heuristic when the search starts) obtained the best results on a sample level and a statistical one (with 5% of significance). Besides, the computational effort derived from the dynamic approach is more significant since it implies invoking the hyper-heuristic for every decision. In contrast, the static approach only requires invoking the hyper-heuristic once per instance.

Regarding future work, it seems interesting to explore other ML methods and compare their performance against other hyper-heuristic approaches, and not only against the heuristics. Besides, our results on the differences between the static and dynamic approaches should be verified in other problem domains with larger instance sets.

References

1. Assi, M., Haraty, R.A.: A survey of the knapsack problem. pp. 1–6. IEEE (11 2018). https://doi.org/10.1109/ACIT.2018.8672677
2. Best, M.J., Ritter, K.: Linear Programming Active Set Analysis and Computer Programs. Prentice Hall Englewood Cliffs, N.J. (1 1985)
3. Bhargava, A.Y.: Dynamic Programming, vol. 1. Manning Publications, 1 edn. (5 2016)
4. Bretthauer, K.M., Shetty, B.: The nonlinear knapsack problem – algorithms and applications. European Journal of Operational Research **138**, 459–472 (5 2002). https://doi.org/10.1016/S0377-2217(01)00179-5
5. 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. Journal of the Operational Research Society **64**(12), 1695–1724 (December 2013)
6. Cacchiani, V., Iori, M., Locatelli, A., Martello, S.: Knapsack problems — an overview of recent advances. part i: Single knapsack problems. Computers & Operations Research **143**, 105692 (2022). https://doi.org/https://doi.org/10.1016/j.cor.2021.105692, `https://www.sciencedirect.com/science/article/pii/S0305054821003877`
7. Drake, J.H., Hyde, M., Ibrahim, K., Ozcan, E.: A genetic programming hyper-heuristic for the multidimensional knapsack problem. Kybernetes **43**, 1500–1511 (11 2014). https://doi.org/10.1108/K-09-2013-0201
8. 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: Martínez-Trinidad, J.F., Carrasco-Ochoa, J.A., Olvera-López, J.A., Sarkar, S. (eds.) Pattern Recognition. pp. 62–71. Springer International Publishing, Cham (2018)
9. Eddy, S.R.: What is dynamic programming? Nature Biotechnology **22**, 909–910 (7 2004). https://doi.org/10.1038/nbt0704-909
10. Kellerer, H., Pferschy, U., Pisinger, D.: Basic Algorithm Concepts, pp. 27–29. Springer Berlin Heidelberg, 1 edn. (10 2004). https://doi.org/10.1007/978-3-540-24777-7
11. Mischek, F., Musliu, N.: Reinforcement learning for cross-domain hyper-heuristics. In: Raedt, L.D. (ed.) Proceedings of the Thirty-First International

Joint Conference on Artificial Intelligence, IJCAI-22. pp. 4793–4799. International Joint Conferences on Artificial Intelligence Organization (7 2022). https://doi.org/10.24963/ijcai.2022/664, main Track
12. Mougouei, D., Powers, D.M.W., Moeini, A.: An Integer Linear Programming Model for Binary Knapsack Problem with Dependent Item Values, vol. 10400, pp. 144–154. Springer International Publishing, 1 edn. (7 2017). https://doi.org/10.1007/978-3-319-63004-5_12
13. Ochoa, G., Hyde, M., Curtois, T., Vazquez-Rodriguez, J.A., Walker, J., Gendreau, M., Kendall, G., McCollum, B., Parkes, A.J., Petrovic, S., Burke, E.K.: Hyflex: A benchmark framework for cross-domain heuristic search. In: Hao, J.K., Middendorf, M. (eds.) Evolutionary Computation in Combinatorial Optimization. pp. 136–147. Springer Berlin Heidelberg, Berlin, Heidelberg (2012)
14. Olivas, F., Amaya, I., Ortiz-Bayliss, J.C., Conant-Pablos, S.E., Terashima-Marin, H.: A fuzzy hyper-heuristic approach for the 0-1 knapsack problem. pp. 1–8. IEEE (7 2020). https://doi.org/10.1109/CEC48606.2020.9185710
15. Ortiz-Bayliss, J.C., Terashima-Marín, H., Conant-Pablos, S.E.: Neural networks to guide the selection of heuristics within constraint satisfaction problems. In: Martínez-Trinidad, J.F., Carrasco-Ochoa, J.A., Ben-Youssef Brants, C., Hancock, E.R. (eds.) Pattern Recognition. pp. 250–259. Springer Berlin Heidelberg, Berlin, Heidelberg (2011)
16. Ortiz-Bayliss, J.C., Amaya, I., Cruz-Duarte, J.M., Gutierrez-Rodriguez, A.E., Conant-Pablos, S.E., Terashima-Marín, H.: A general framework based on machine learning for algorithm selection in constraint satisfaction problems. Applied Sciences **11**(6) (2021). https://doi.org/10.3390/app11062749
17. Pillay, N., Beckedahl, D.: Evohyp - a java toolkit for evolutionary algorithm hyper-heuristics. pp. 2706–2713. IEEE (6 2017). https://doi.org/10.1109/CEC.2017.7969636
18. 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 Computing **23**, 12711–12728 (12 2019). https://doi.org/10.1007/s00500-019-03822-w
19. RASHID, M.H.: A gpu accelerated parallel heuristic for the 2d knapsack problem with rectangular pieces. pp. 783–787. IEEE (11 2018). https://doi.org/10.1109/UEMCON.2018.8796818
20. 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. Applied Sciences **11**, 10209 (10 2021). https://doi.org/10.3390/app112110209
21. Tu, C., Bai, R., Aickelin, U., Zhang, Y., Du, H.: A deep reinforcement learning hyper-heuristic with feature fusion for online packing problems. Expert Systems with Applications **230** (11 2023). https://doi.org/10.1016/j.eswa.2023.120568
22. Tyasnurita, R., Özcan, E., John, R.I.: Learning heuristic selection using a time delay neural network for open vehicle routing. 2017 IEEE Congress on Evolutionary Computation (CEC) pp. 1474–1481 (2017), `https://api.semanticscholar.org/CorpusID:5959987`
23. Zeng, Z., Xiong, C., Yuan, X., Bai, Y., Jin, Y., Lu, D., Lian, L.: Information-driven path planning for hybrid aerial underwater vehicles (4 2022)
24. Žerovnik, J.: Heuristics for np-hard optimization problems - simpler is better!? Logistics & Sustainable Transport **6**, 1–10 (11 2015). https://doi.org/10.1515/jlst-2015-0006