A Preliminary Study on Feature-independent Hyper-heuristics for the 0/1 Knapsack Problem

Xavier F. C. 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 Science
E-mails: {sax, jcobayliss, iamaya2, jorge.cruz, sconant, terashima}@tec.mx

***Abstract*—Recent years have witnessed an escalating interest for methods that automatically adapt to different types of problems. In this regard, the term hyper-heuristics—heuristics that either select or generate new heuristics—is a relevant concept. Experimental evidence supports the idea that hyper-heuristics can outperform single, isolated heuristics. However, commonly used hyper-heuristic models require several inputs. One of them is a set of features that accurately characterize the instances, which limits their applicability. Thus, in this work, we analyze how to implement a simple evolutionary algorithm to produce feature-independent hyper-heuristics. We compare its performance against that of simple heuristics, for the domain of the knapsack problem. Our research focuses on two elements: performance and frequency. In the former, we analyze how the performance of the learning stage varies across different scenarios. In the latter, we examine how frequently heuristics interact within the hyper-heuristic. We show that the proposed hyper-heuristic model solves most of the instances considered in this work. Moreover, it does so more efficiently than isolated heuristics. At the same time, the model offers a straightforward parameter setting and requires little or no problem characterization, which simplifies its use on new problem domains.**

***Index Terms*—Heuristics, Hyper-heuristics, Knapsack problem.**

I. Introduction

Hyper-heuristics are considered high-level heuristics useful to tackle hard-to-solve problems [1], particularly the NP-hard ones [2]. Hyper-heuristics are usually classified into two main groups: selection hyper-heuristics —those that select heuristics from an available set—, and generation hyper-heuristics — those that create new heuristics using components of existing ones [3]. Although both groups have proved of great interest to the scientific community, in this work, we will only focus on the former.

Selection hyper-heuristics work on the problems indirectly since they browse a set of available heuristics, which are selectively applied to solve the problem at hand [4]. A generic selection hyper-heuristic analyzes a set of available heuristics and selects the most suitable one according to a given performance metric. Most of the current selection hyper-heuristic models include two key phases: heuristic selection and move acceptance [5]. The former represents the strategy for deciding which heuristic should be selected. Conversely, move acceptance 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.

Among the many learning methods used in the literature to produce hyper-heuristics, evolutionary computation is a recurring one. Examples of these methods include, but are not limited to Genetic Programming (GP) [4], Grammatical Evolution (GE) [6], Messy Genetic Algorithms (MGA) [7], and Artificial Immune Systems (AIS) [8], [9]. Even so, there are other proposals, such as those that create the hyper-heuristic by analyzing the set of problem instances [10].

Despite the wide use of hyper-heuristics [3], [11], only a few works explore the insights of their behavior. A few examples include the run-time analysis of selection hyper-heuristics [12], [13], the use and control of crossover operators for selection hyper-heuristics [14], and heuristic interaction of selection hyper-heuristics applied to constraint satisfaction problems [15]. Therefore, this paper conducts an exploration of heuristic interaction for the 0/1 knapsack problem using a simple evolutionary algorithm (EA). Moreover, some traditional selection hyper-heuristic models require the definition of a set of features for mapping the state of a problem instance [16]. Hence, this work proposes an alternative selection hyper-heuristic model that is feature-independent.

The remainder of this document is organized as follows. Section II defines the problem and presents some relevant related works. The rationale and details of the proposed hyper-heuristic model are explained in Section III. Section IV describes the experiments conducted and discusses the results. Finally, Section V presents the conclusions and traces some paths for future research.

II. Background

The knapsack problem, as classically defined, consists of a set of n items (each one with its profit p and weight w) and a knapsack of capacity c. Solving a knapsack problem means that we must find a subset of the items such that:

This research was partially supported by CONACyT Basic Science Project under grant 287479 and ITESM Research Group with Strategic Focus on Intelligent Systems.

978-1-7281-6929-3/20/$31.00 ©2020 IEEE

$$\text{maximize} \qquad z = \sum_{j=1}^{n} p_j x_j \tag{1}$$

$$\text{subject to} \qquad \sum_{j=1}^{n} w_j x_j \leq c, \tag{2}$$

$$\text{where} \qquad x_j = \begin{cases} 1, & \text{if item } j \text{ is selected;} \\ 0, & \text{otherwise.} \end{cases} \tag{3}$$

The knapsack problem is one of the most studied combinatorial optimization problems due to its wide range of applications, which include [17]: cargo loading, cutting stock, allocation, and cryptography. When this problem is solved through a constructive approach, the solution is built one step at a time, deciding if one particular item must be packed or ignored. For simplicity, our setting states that once a decision is made for an item, there is no way to change it. Using a constructive approach leads to different subproblems. 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 packed item is removed from the list). Since the selection of the item is closely related to the heuristic used, we can state that the choice of the heuristic leads to different subproblems throughout the solving process.

This behavior brings up 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. Literature usually refers to the problem of selecting the best algorithm for a particular situation as the algorithm selection problem [18].

Evolutionary algorithms are common learning mechanisms employed in hyper-heuristics. They have been used to tackle packing problems in the past. For example, Hart and Sim [8], [9] employed GP trees as learning mechanisms for the bin packing (BPP) and job shop scheduling (JSP) problems. Falkenauer [19] proposed a GA hyper-heuristic model for the BPP, and Hyde [20], Burke et al. [21] and Drake et al. [22] studied GP rules for bin packing. These findings support the idea of using an evolutionary strategy as a learning mechanism for hyper-heuristics, one which improves via crossover or mutation. This work focuses on the latter.

III. The Hyper-heuristic Model

The hyper-heuristic used in this research is classified as an offline selection hyper-heuristic [23]. The learning mechanism relies on a variant of the original $(1 + 1)$ Evolutionary Algorithm (EA). The original algorithm [24] used a single mutation operator, which flipped each bit with probability $1/n$, where n is the length of the chromosome. The EA version used in this work extends the algorithm proposed by Lehre and Özcan [13], which chooses one among the available mutation operators based on a uniform random probability distribution.

The learning mechanism within the hyper-heuristic works as follows. The model starts by generating a random sequence of heuristics (the current hyper-heuristic) and calculates its fitness as the average performance on a set of training instances. In this case, the performance is measured in terms of the profit of the items contained in the knapsack. In our EA implementation, the chromosome itself represents, by using a vector of integers, a hyper-heuristic that codes a sequence of heuristics to apply. At each iteration, the process randomly chooses one of the available mutation operators and applies it on a copy of the current hyper-heuristic, which produces a candidate hyper-heuristic. Each mutation operator has the same probability of being selected. The model evaluates the candidate hyper-heuristic on the set of training instances and, if its fitness is larger or equal (to favor diversity) than the current one, this candidate takes its place. The process is repeated until a maximum number of iterations is reached.

Pseudocode 1 presents this learning mechanism.

Pseudocode 1 Learning mechanism to produce selection hyper-heuristics

```
Inputs:
  heuristics – Set of heuristics
  operators – Set of mutation operators
  maxIterations – Maximum number of iterations
  trainingSet – Set of instances to train the hyper-heuristic
Output:
  A selection hyper-heuristic

  current ← INITIALIZE(heuristics)
  currentEval ← EVALUATE(current, trainingSet)
  for i = 0 to maxIterations do
    candidate ← CLONE(current)
    operator ← SELECT(operators)
    candidate ← MUTATE(candidate, operator)
    candidateEval ← EVALUATE(candidate, trainingSet)
    if candidateEval ≥ currentEval then
      current ← candidate
      currentEval ← candidateEval
    end if
  end for
  return current
```

The main goal of the learning mechanism is to produce a hyper-heuristic (an ordered sequence of heuristics) that, once used to solve an instance, the solution obtained maximizes the profit of the items packed within the knapsack. Thus, the evolutionary algorithm does not operate on the solution space, but on the heuristic space.

A. Heuristics

Four simple packing heuristics were considered for this research due to its popularity and performance in similar works.

- **Default (Def)** packs the items in the same order 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 unit (MaxPW)** calculates the profit-over-weight ratio for each item and sorts them in descending order. Then, MaxPW follows such an ordering until the knapsack is full or no more items are left to be packed.
- **Minimum weight (MinW)** favors lighter items, so it sorts the items in ascending order based on their weight and packs them by following such an order and as long as they fit.

In the case of ties, all the heuristics will prefer the first item among the tied ones. From these heuristics, Def is the fastest one to execute, which runs in $O(n)$. On the contrary, MaxP, MaxPW, and MinW run in $O(n \log n)$. While these three packing heuristics take longer to compute, they 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 stage, we consider suitable to test the proposed hyper-heuristic approach on this set of simpler heuristics.

B. Mutation operators

The evolutionary process uses one of eight available mutation operators to perturbate the candidate hyper-heuristic. These operators are described below:

- **AddGene** inserts a random gene at a randomly selected position.
- **FlipOneGene** takes a randomly selected gene and assigns a random packing operator.
- **FlipTwoGenes** works in the same fashion as FlipOneGene but choosing two genes at random.
- **AddGeneNeigh** selects a random location in the hyper-heuristic and inserts a gene based on its neighbors.
- **FlipOneGeneNeigh** is a neighborhood variant of FlipOneGene, in which the randomly chosen gene is replaced by one of its neighbors.
- **FlipTwoGenesNeigh** acts like FlipTwoGenes, but the packing operators are chosen at random from the neighborhood of each selected gene.
- **SwapGenes** interchanges the positions of two randomly selected genes.
- **RemoveGene** selects a random gene and removes it from the hyper-heuristic.

The EA starts with a hyper-heuristic of a fixed size: four genes. We use the integer division of the number of items in the instance over the number of genes in the hyper-heuristic to calculate the number of items to be packed by using each gene. In this way, when the hyper-heuristic is initialized, each gene packs "a quarter" of the instance. Since some mutation operators can add or remove genes throughout the evolutionary process, the proportion of items packed by each gene will change accordingly to the number of genes in the chromosome. In case the integer division leaves a remainder, the additional items are added sequentially and backward, from the last gene back to the first one. For example, given an instance with 20 items and a hyper-heuristic coded within a five-gene chromosome, the first three genes will pack four items, while the fourth and fifth will pack six items. The graphical representation of this hyper-heuristic and the items packed is depicted in Fig. 1.

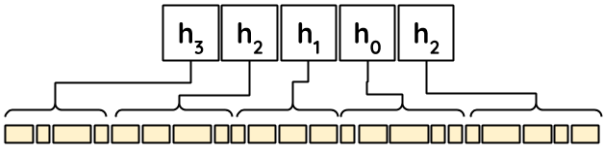

Fig. 1. An example hyper-heuristic coded by using a five-gene chromosome. This hyper-heuristic is used to solve an instance of 22 items. As depicted, the first three genes pack four items each. Since the integer division of 22 over five produces a remainder of two, those two items must be distributed backward among the genes. In this particular example, those two items are assigned to the two last genes, one item per gene. The hyper-heuristic will start solving the instance by using h_3 but will end using h_2.

Over time, a hyper-heuristic may end up with different distributions of items per gene: some genes could potentially contain more than one item, and sometimes a gene may contain a single item. These strategies generate more interactions between packing heuristics.

IV. Experiments and Results

To test our model, we considered a reduced set of 100 knapsack instances of 20 items and 50 units of capacity. Although the collection seems small, it contains enough diversity to support our conclusions. The instances used in this work are available upon request. To obtain such instances, we used an evolutionary process that tailors the instances to specific solvers, as depicted in [25]. We stated that the set is balanced in the sense that each heuristic represents the best option for solving around 25% of the instances. The rationale behind this distribution is to guarantee that no single heuristic represents the best option for solving the whole set of instances. Therefore, we expect that hyper-heuristics outperform individual heuristics by adequately identifying the cases where one of them should be used. The collection of instances was sampled to generate the training and test sets. We tested different configurations for training and test sets, and we will detail each configuration within the descriptions of the experiments. In all the cases, hyper-heuristics were produced using the entire training set. Moreover, results always refer to the test set, where hyper-heuristics were compared against single heuristics. Throughout the experimental setting, two aspects of the hyper-heuristic were analyzed: the performance of the learning mechanism and the interactions between the heuristics.

A. Analyzing the Performance of the Learning Mechanism

In this first experiment, we used 60% of the available instances for training and the remaining 40% for testing. Ten different training and test sets were generated to minimize the effect of the instances used for training in the resulting

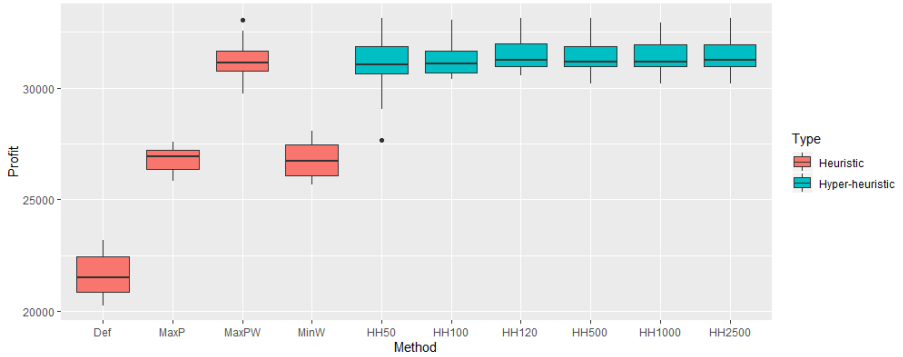

Fig. 2. Distribution of the profits achieved by the heuristics and the hyper-heuristics produced under different scenarios (50, 100, 120, 500, 1000 and 2500 iterations) with 60% of the instances used for training.

hyper-heuristic. We conducted ten different runs of the training process, under six different scenarios: 50, 100, 120, 500, 1000, and 2500 iterations. These values were chosen based on preliminary experiments. For each scenario, ten hyper-heuristics were produced, one per each training set.

The performance of each heuristic, as well as the performance of the hyper-heuristics produced, are depicted in Fig. 2. The performance of these methods is calculated over the remaining instances kept for testing for each particular run. Then, only unseen instances are considered for measuring the performance of the methods on the following tests.

We can observe that MaxPW provides outstanding results for the instances under analysis, clearly overcoming the remaining heuristics. Regarding the hyper-heuristics, their performance overcomes three heuristics: Def, MaxP, and MinW. When compared against MaxPW, their performance is somewhat similar, at least in the distribution of the profits achieved. Based on the results shown in Fig. 2, it seems that using 50-iteration training is a poor decision since it introduces a high variation in the performance of the resulting hyper-heuristics. As the number of iterations increases, this range seems to shorten.

Trying to further analyze the behavior of heuristics and the hyper-heuristics produced, we analyzed the profit distribution of each of these methods in more detail. Table I shows the profit of all single heuristics and all hyper-heuristics, for every scenario and on their respective test set.

From Table I, we can observe that, even for the 50-iteration scenario—which was considered the one with the highest variation—, hyper-heuristics are the best-solving strategy in 70% of the cases. Only in 30% of the cases, MaxPW outperformed HH50. In the remaining cases, all the hyper-heuristics outperformed MaxPW in, at least, 80% of the cases. Thus, it would seem that increasing the number of training iterations translates into getting better results more consistently. This behavior is bolstered by the 120-iteration scenario, where hyper-heuristics always represented the best choice. However, it is undermined by the 500-iteration situation. In general, using 120 iterations in the EA seems to produce the best scenario (profit-wise).

The next step of the analysis evaluated the results from Table I by using hypothesis tests on the means of the profits. We used a two-tail t-test, where H_0 stands for "the mean profit of both methods is equal" and H_1 for " the mean profit of both methods is not equal". The statistical evidence suggests that some of these hyper-heuristics are better than MaxPW (ranked as the best performing heuristic when applied in isolation). The t-tests conducted on the pairs (MaxPW, HH50), (MaxP, HH100), (MaxPW, HH120), (MaxWP, HH500), (MaxPW, HH1000) and (MaxPW, HH2500) reported p-values of 0.511, 0.961, 0.003, 0.001, 0.018, and 0.001, respectively. Then, the statistical evidence supports the idea that HH120, HH500, HH1000, and HH2500 are, in general, better than MaxPW for the type of instances studied in this work (assuming a standard value of $\alpha = 0.05$).

Although the previous scenarios yielded good results, they drift away from the optimal. The reason for this is that the exploration strategy relies on the probability distribution of the mutation operators. For example, there are two mutation operators for adding genes but only one for removing them. Since mutation operators are chosen using a uniform probability model, it is more likely to end up with longer hyper-heuristics. Including acceptance operators, as suggested in [13], may benefit the hyper-heuristic performance to find the optimal solution at large-iteration scenarios.

Furthermore, it is necessary to notice that the profit achieved by some hyper-heuristics did not vary across scenarios. We think that this phenomenon occurs because the initial solution is always four genes long, meaning that more than one item is packed using the same heuristic at the beginning of the evolution process. It is also worth remarking that the performance of the hyper-heuristic is derived exclusively from evolution since no feature analysis is involved. We consider this among the most reliable features of our approach. Furthermore, training a hyper-heuristic through our model is quite affordable since evolving a single individual is more straightforward (and faster) than dealing with crossover of moderately populated models.

Striving to test the proposed hyper-heuristic model further, we analyzed four additional scenarios. This time around,

TABLE I

RESULTS ACHIEVED BY EACH HEURISTIC (DEF, MAXP, MAXPW, AND MINW) AND THE HYPER-HEURISTICS (HH) AFTER 50, 100, 120, 500, 1000, AND 2500 ITERATIONS ON EACH OF THE TEST SETS PRODUCED. RESULTS IN BOLD INDICATE THAT THE HYPER-HEURISTIC OUTPERFORMED THE FOUR SINGLE HEURISTICS IN THAT PARTICULAR TEST SET. THE LAST ROW SUMMARIZES THE PERCENTAGE OF CASES WHERE EACH HYPER-HEURISTIC OBTAINED BETTER RESULTS THAN THE FOUR HEURISTICS FOR EACH PARTICULAR TEST SET.

Test set	Def	MaxP	MaxPW	MinW	HH50	HH100	HH120	HH500	HH1000	HH2500
1	21582	27590	33052	28074	**33133**	33038	**33133**	**33133**	32916	**33133**
2	20626	26315	30883	25736	**31174**	**31174**	**31174**	**31174**	**31174**	**31174**
3	22752	26483	31655	26210	**31828**	**31828**	**31828**	**31828**	**31957**	**31957**
4	23186	27209	31593	27614	**31857**	**31857**	**32052**	**31857**	**31857**	**31857**
5	20495	27160	29748	26803	29058	**30382**	**30551**	**30193**	**30193**	**30193**
6	21511	27098	30949	26958	27650	**31163**	**31163**	**31163**	**31187**	**31163**
7	20235	25824	30697	26010	**30898**	**30898**	**30898**	**30923**	**30923**	**30898**
8	23098	27527	32538	28032	**32856**	30492	**32856**	**32761**	**32676**	**32678**
9	21527	25940	31275	25661	30822	31035	**31297**	31050	31089	**31297**
10	21487	26753	29934	26653	**30581**	**30581**	**30581**	**30581**	**30543**	**30581**
	0%	0%	0%	0%	70%	70%	100%	90%	80%	100%

however, we changed the ratio of instances used for training, while analyzing only the 50 and 120-iteration scenarios. In the first two cases, we considered a 50% train ratio (Table II). In the remaining two, we reduced it to 30% (Table III). Also, the reader may find useful to look at Fig. 3 and 4 for a graphical representation of the data, where the distribution of the profits obtained for each method over the ten runs is depicted.

As in the previous case, when the statistical evidence is analyzed, we observe a competent behavior of hyper-heuristics. The p-values for the comparison of the pairs (MaxPW, HH50) and (MaxPW, HH120) when 50% of the data is used for training are 0.005 and 0.001, respectively. Although the hyper-heuristics seemed to reduce its performance when we drastically reduced the size of the training set, they remained competitive for some configurations. The p-values for the case where only 30% of the instances were used for training are 0.53 and 0.024, for (MaxWP, HH50) and (MaxPW, HH120), respectively. It is interesting to note that, although the statistical evidence is not as reliable as in the previous cases, the hyper-heuristics performed surprisingly well also on this set, ranking first for almost all the cases.

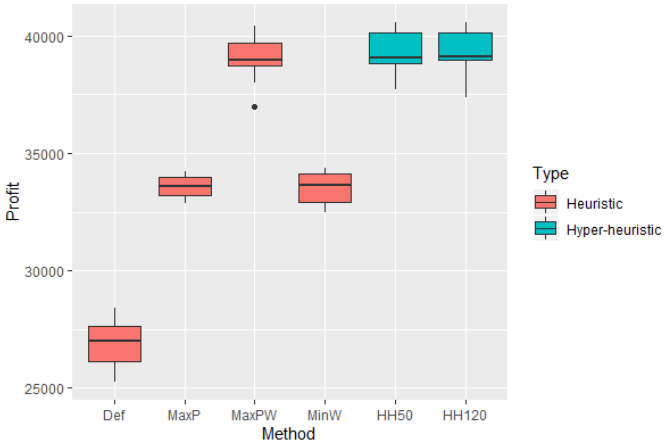

Fig. 3. Distribution of the profits achieved by the heuristics and the hyper-heuristics produced with 50 and 120 iterations and with 50% of the instances used for training.

B. Analyzing Heuristic Interactions

The previous stage showed that a featureless approach to hyper-heuristics could yield good results. Thus, during this stage, we analyze the nature of the generated hyper-heuristics. To do so, we carried out a frequency analysis for all combinations of two-heuristic sequences. We trained a new hyper-heuristic for each one of the training subsets and for 120 iterations each (since this scenario yielded the best results in the previous experimental stage).

Throughout the process, we stored each hyper-heuristic (the list of heuristics applied to each instance, and for each iteration). Afterwards, we calculated the frequency of all two-heuristic sequences on each run. Table IV summarizes these results (numbers in bold highlight the most common heuristic sequence for the run).

As can be seen from Table IV, MaxPW + MaxPW was the dominating sequence. In nine out of ten runs, hyper-

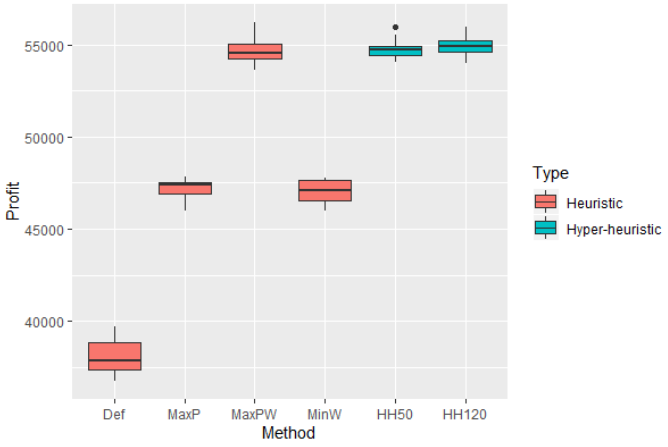

Fig. 4. Distribution of the profits achieved by the heuristics and the hyper-heuristics produced with 50 and 120 iterations and with 30% of the instances used for training.

TABLE II

RESULTS ACHIEVED BY EACH BASE HEURISTIC (DEF, MAXP, MAXPW, AND MINW), AND BY THE HYPER-HEURISTICS (HH) TRAINED WITH 50% OF THE INSTANCES, AND FOR 50 AND 120 ITERATIONS. RESULTS IN BOLD INDICATE THAT THE HYPER-HEURISTIC OUTPERFORMED THE FOUR SINGLE HEURISTICS IN THAT PARTICULAR TEST SET. THE LAST ROW SUMMARIZES THE PERCENTAGE OF CASES WHERE EACH HYPER-HEURISTIC OBTAINED BETTER RESULTS THAN THE FOUR HEURISTICS FOR EACH PARTICULAR TEST SET.

Test set	Def	MaxP	MaxPW	MinW	HH50	HH120
1	26055	33967	40419	34221	**40585**	**40585**
2	25250	33437	38773	33147	38796	**39129**
3	27694	32857	39362	32659	**39626**	**39626**
4	28418	33717	39828	34145	**40389**	**40389**
5	26296	33929	37991	34074	**38184**	**38184**
6	27345	34196	38730	33571	**38987**	**38987**
7	27308	33047	38912	33668	**39097**	**39097**
8	28407	33968	39875	34342	**40316**	**40316**
9	25682	33168	**38992**	32466	38947	38947
10	26728	33324	37003	32826	**37732**	37384
	0%	0%	10%	0%	80%	80%

TABLE III

RESULTS ACHIEVED BY EACH BASE HEURISTIC (DEF, MAXP, MAXPW, AND MINW), AND BY A HYPER-HEURISTIC (HH) TRAINED WITH 30% OF THE INSTANCES, AND FOR 50 AND 120 ITERATIONS. RESULTS IN BOLD INDICATE THAT THE HYPER-HEURISTIC OUTPERFORMED THE FOUR SINGLE HEURISTICS IN THAT PARTICULAR TEST SET. THE LAST ROW SUMMARIZES THE PERCENTAGE OF CASES WHERE EACH HYPER-HEURISTIC OBTAINED BETTER RESULTS THAN THE FOUR HEURISTICS FOR EACH PARTICULAR TEST SET.

Test set	Def	MaxP	MaxPW	MinW	HH50	HH120
1	37581	47836	**56227**	47750	55968	55968
2	36734	46832	54461	45950	54526	**54952**
3	38564	46475	55088	46286	**55545**	55274
4	39705	47412	**54977**	47782	**54977**	54948
5	37306	47451	54106	47448	54106	**54554**
6	37400	47564	54323	46907	**54828**	**54828**
7	38133	45950	53645	46429	**54042**	53967
8	39271	47033	**54668**	47680	**54668**	**54668**
9	36853	47393	55418	47060	54834	**55820**
10	38940	47615	54226	47127	**54372**	**54372**
	0%	0%	30%	0%	60%	60%

TABLE IV

FREQUENCY OF TWO-SEGMENT LOW-LEVEL HEURISTIC SEQUENCES. RESULTS IN BOLD INDICATE THE SEQUENCE THAT WAS USED THE MOST FOR EACH PARTICULAR TEST SET.

Sequence/Test set	1	2	3	4	5	6	7	8	9	10
Def + Def	5	14	0	23	0	6	64	**150**	5	151
Def + MaxP	13	24	0	14	0	30	38	25	0	0
Def + MaxPW	7	17	8	2	5	11	21	4	8	6
Def + MinW	0	3	10	37	0	15	22	6	9	0
MaxP + Def	11	32	8	26	3	18	53	110	0	48
MaxP + MaxP	61	63	58	115	80	62	61	59	9	35
MaxP + MaxPW	139	70	131	28	76	80	82	79	90	67
MaxP + MinW	0	21	56	31	1	18	0	17	9	29
MaxPW + Def	23	29	0	10	2	12	34	14	11	43
MaxPW + MaxP	179	115	180	91	101	24	104	120	106	102
MaxPW + MaxPW	**217**	**270**	**341**	**226**	**374**	**292**	**295**	55	**334**	**261**
MaxPW + MinW	3	31	124	40	18	79	13	4	28	14
MinW + Def	6	15	13	17	0	26	0	5	0	4
MinW + MaxP	0	37	15	39	0	97	9	55	2	18
MinW + MaxPW	7	27	58	47	19	33	22	55	48	117
MinW + MinW	0	8	88	21	8	106	4	106	24	29

heuristics evolved into a high use of MaxPW only. This phenomenon follows the behavior exhibited by standalone heuristics: MaxPW outperforms the remaining heuristics in most of the cases and by a wide enough gap. Our proposed model can recognize this behavior and seeks to use MaxPW most of the time. Nonetheless, it also detects an opportunity for improvement and learns that MaxPW should be combined with others to yield better results, for example, with MaxP, and with MinW. In the remaining 10% of the runs, the single-heuristic sequence Def + Def was the most commonly used, though

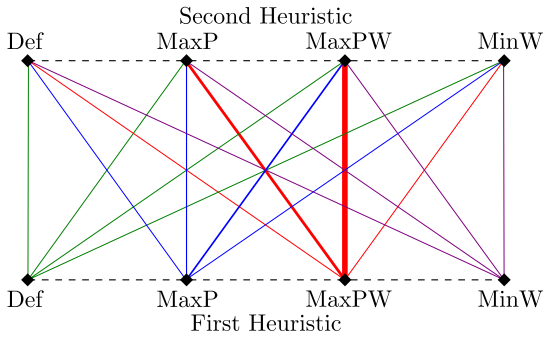

Fig. 5. Heuristic interaction across all runs. Line thickness is proportional to the number of times that a sequence appears.

its frequency is less than other scenarios. In the remaining cases, i.e. for MaxP-MaxP and MinW-MinW, interaction loses importance (Table IV). In the first case, it ranks between third and fourth place across the runs. But, in the second case, the ranking oscillates between the fourth and last places.

The most used sequence of heuristics is, by far, MaxPW + MaxPW, but the hyper-heuristics also use other sequences. For example, sequences MaxPW + MaxP and MaxP + MaxPW are the second and third most used choices, respectively. Conversely, some sequences are rarely used by the hyper-heuristics. For example, sequences such as MinW + Def and Def + MinW. Thus, mixing Def and MinW seems like a bad idea (at least in the way described in this work). We can derive a similar conclusion by looking at the sequences Def + MaxP, Def + MaxPW, and MaxPW + Def. Hence, it would seem as it is only beneficial to add the Def heuristic after using MaxPW (and only in some specific cases).

Another way of analyzing the interaction between heuristics is to consider the data of all runs at once. Fig. 5 shows these data by using a line thickness proportional to the number of times that a sequence appears. Once again, it is evident that the sequence MaxPW + MaxPW is the most popular. It represents about a third of all interactions. The interaction strength between MaxPW and MaxP (in both ways) is also evident. Only these two combinations of heuristics accumulate about 24% of all the interactions. Interactions Def + MaxP, Def + MaxP, Def + MinW, and MinW + Def stand as the least used (thinnest lines), barely accumulating 5% of all the total interactions.

C. Discussion

The frequency analysis of heuristic interactions cannot guarantee, by itself, that a single heuristic is the best option for solving all instances of the problem. Since a knapsack has limited capacity and not all items from an instance are supposed to fit in, heuristic sequences at the end of longer hyper-heuristics may not impact the profit. However, they do contribute to the frequency analysis.

For a human, switching heuristics at specific decision points may seem obvious. A simple example would be the situation where only two items are left, and the knapsack has capacity for any of them but not both. Under these conditions, the best strategy is to apply MaxP instead of MaxPW. This 'small' detail, visible for humans, was —surprisingly— learned by the proposed model. This finding is the reason why the sequence MaxPW + MaxP became the second most frequent packing heuristic sequence. Then, after all, there is something to learn from studying the interactions between heuristics, even if it may go unnoticed at first glance.

V. Conclusions and Future Work

In this work, an evolutionary algorithm that evolves heuristic sequences powers a feature-independent hyper-heuristic. We selected the problem domain of binary Knapsack Problems to validate the proposed approach. We focused on two elements: performance and frequency. First, we analyzed the efficiency of the model by comparing the quality of the solutions produced by different methods. Later, we analyzed heuristic interactions as a means to understand why some hyper-heuristics work and how different heuristics can lead to improvements in the search process. Despite its simplicity, our data shows that the learning mechanism performs well on most instances. Before using our approach, MaxPW seemed like the most promising heuristic. Still, most of the hyper-heuristics generated in this work outperformed such a competent heuristic.

Here, we used a balanced set of instances. Such a set contained a similar number of instances where each solver excelled. But, instances from other sources, instead, might be harder to solve and could represent a challenge for an isolated heuristic. Moreover, the set we used was rather small, so the learning process was not computationally expensive. Even so, good results were achieved. When used on larger sets, a hyper-heuristic is expected to perform better, even if the learning process becomes computationally expensive.

Many paths could be pursued in future work. Expanding our analysis by fixing the size of hyper-heuristics and setting a single heuristic per gene in the model seems interesting, and may impact the frequency analysis. Adding more mutation operators, and tweaking their probability distribution is also something worth considering. More importantly, the effect of modifying the amount of heuristics to choose from should be analyzed. For example, it would be worth exploring the effect of removing the best performing heuristic from the pool and see how the hyper-heuristics cope with this new scenario. Though increasing the search domain, this may allow for more explicit differences between heuristic interactions. Once again, we would like to emphasize the fact that our proposed model does not require any problem characterization or feature analysis. Even so, adding problem characterization may improve the performance of the learning process even more, at the expense of some loss in generality.

References

[1] R. Bai, E. K. Burke, M. Gendreau, G. Kendall, and B. McCollum, "Memory length in hyper-heuristics: An empirical study," in *Computational Intelligence in Scheduling, 2007. SCIS '07. IEEE Symposium on*, April 2007, pp. 173–178.

[2] E. K. Burke, M. Gendreau, M. Hyde, G. Kendall, G. Ochoa, E. zcan, and R. Qu, "Hyper-heuristics: a survey of the start of the art," *Journal of the Operational Research Society*, vol. 64, no. 12, pp. 1695–1724, December 2013.

[3] N. Pillay and R. Qu, *Hyper-Heuristics: Theory and Applications*, ser. Natural Computing Series. Cham: Springer International Publishing, 2018. [Online]. Available: http://link.springer.com/10.1007/978-3-319-96514-7

[4] R. Poli and M. Graff, "There is a free lunch for hyper-heuristics, genetic programming and computer scientists," in *Genetic Programming: 12th European Conference, EuroGP 2009 Tübingen, Germany, April 15-17, 2009 Proceedings*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 195–207.

[5] E. Özcan, B. Bilgin, and E. E. Korkmaz, "A comprehensive analysis of hyper-heuristics," *Intell. Data Anal.*, vol. 12, no. 1, pp. 3–23, January 2008.

[6] N. Lourenço, F. B. Pereira, and E. Costa, "The importance of the learning conditions in hyper-heuristics," in *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation*, ser. GECCO '13. New York, NY, USA: ACM, 2013, pp. 1525–1532.

[7] J. C. Ortiz-Bayliss, J. H. Moreno-Scott, and H. Terashima-Marn, "Automatic generation of heuristics for constraint satisfaction problems," in *Nature Inspired Cooperative Strategies for Optimization (NICSO 2013)*, ser. Studies in Computational Intelligence, vol. 512, 2013, pp. 315–327.

[8] E. Hart and K. Sim, "On the life-long learning capabilities of a NELLI*: A hyper-heuristic optimisation system," in *Parallel Problem Solving from Nature PPSN XIII*, ser. Lecture Notes in Computer Science, vol. 8672, September 2014, pp. 282–291.

[9] K. Sim and E. Hart, "An improved immune inspired hyper-heuristic for combinatorial optimisation problems," in *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation*, ser. GECCO '14. New York, NY, USA: ACM, 2014, pp. 121–128.

[10] I. Amaya, J. C. Ortiz-Bayliss, S. Conant-Pablos, and H. Terashima-Marin, "Hyper-heuristics Reversed: Learning to Combine Solvers by Evolving Instances," in *2019 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, jun 2019, pp. 1790–1797. [Online]. Available: https://ieeexplore.ieee.org/document/8789928/

[11] F. Garza-Santisteban, R. Sanchez-Pamanes, L. A. Puente-Rodriguez, I. Amaya, J. C. Ortiz-Bayliss, S. Conant-Pablos, and H. Terashima-Marin, "A Simulated Annealing Hyper-heuristic for Job Shop Scheduling Problems," in *2019 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, jun 2019, pp. 57–64. [Online]. Available: https://ieeexplore.ieee.org/document/8790296/

[12] F. Alanazi and P. K. Lehre, "Runtime analysis of selection hyper-heuristics with classical learning mechanisms," in *2014 IEEE Congress on Evolutionary Computation (CEC)*, July 2014, pp. 2515–2523.

[13] P. K. Lehre and E. Özcan, "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*, ser. FOGA XII '13. New York, NY, USA: ACM, 2013, pp. 97–104.

[14] J. H. Drake, E. Özcan, and E. K. Burke, "A case study of controlling crossover in a selection hyper-heuristic framework using the multidimensional knapsack problem," *Evolutionary Computation*, vol. 24, no. 1, pp. 113–141, March 2016.

[15] J. C. Ortiz-Bayliss, H. Terashima-Marn, E. Özcan, A. J. Parkes, and S. E. Conant-Pablos, "Exploring heuristic interactions in constraint satisfaction problems: A closer look at the hyper-heuristic space," in *2013 IEEE Congress on Evolutionary Computation*, June 2013, pp. 3307–3314.

[16] I. Amaya, J. C. Ortiz-Bayliss, A. Rosales-Pérez, A. E. Gutiérrez-Rodríguez, S. E. Conant-Pablos, H. Terashima-Marín, and C. A. Coello Coello, "Enhancing Selection Hyper-Heuristics via Feature Transformations," *IEEE Computational Intelligence Magazine*, vol. 13, no. 2, pp. 30–41, 2018.

[17] C. Wilbaut, S. Hanafi, and S. Salhi, "A survey of effective heuristics and their application to a variety of knapsack problems," *IMA Journal of Management Mathematics*, vol. 19, no. 3, p. 227, 2008.

[18] J. R. Rice, "The algorithm selection problem," *Advances in Computers*, vol. 15, pp. 65–118, 1976.

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

[20] M. Hyde, "A genetic programming hyper-heuristic approach to automated packing," Ph.D. dissertation, University of Nottingham, March 2010.

[21] E. K. Burke, M. R. Hyde, G. Kendall, and J. Woodward, "Automating the packing heuristic design process with genetic programming," *Evol. Comput.*, vol. 20, no. 1, pp. 63–89, March 2012.

[22] J. H. Drake, M. Hyde, K. Ibrahim, and E. Özcan, "A genetic programming hyper-heuristic for the multidimensional knapsack problem," in *11th IEEE International Conference on Cybernetic Intelligent Systems*, Limerick, Ireland, August 2012.

[23] E. K. Burke, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, and J. R. Woodward, "A classification of hyper-heuristic approaches," in *Handbook of Metaheuristics*. Boston, MA: Springer US, 2010, pp. 449–468.

[24] S. Droste, T. Jansen, and I. Wegener, "On the analysis of the (1+1) evolutionary algorithm," *Theoretical Computer Science*, vol. 276, no. 1, pp. 51–81, 2002.

[25] L. F. Plata-González, I. Amaya, J. C. Ortiz-Bayliss, S. E. Conant-Pablos, H. Terashima-Marín, and C. A. Coello Coello, "Evolutionary-based tailoring of synthetic instances for the knapsack problem," *Soft Computing*, vol. 23, no. 23, pp. 12 711–12 728, Dec 2019.