Recursive Hyper-heuristics for the Job Shop Scheduling Problem

Alonso Vela, Jorge M. Cruz-Duarte, José Carlos Ortiz-Bayliss, Ivan Amaya
Tecnologico de Monterrey
Monterrey, Mexico
{a00815751, jorge.cruz, jcobayliss, iamaya2}@tec.mx

***Abstract*—Hyper-heuristics are a broad topic that has drawn increasing attention because of its flexibility. This, however, implies that there are diverse models, including selection hyper-heuristics, where the idea is to derive a model that learns when to use each available solver. Nonetheless, such a learning procedure usually proves difficult and leads to non-ideal selections. Hence, in this work, we propose a recursive hyper-heuristic model allowing more complexity within the selection models. Our idea is straightforward: to have a selection hyper-heuristic to select low-level heuristics and lower-level hyper-heuristics. In doing so, one can merge the combined decisions of existing solvers. We test the feasibility of such a model through experiments on the Job Shop Scheduling Problem that cover small and large datasets of previously tailored instances. We found that increasing the order of the model leads to more stable and better-performing approaches. For example, migrating from a second-order hyper-heuristic to a fourth-order hyper-heuristic reduced the makespan by over 6%. Thus, the proposed model seems feasible and should be further tested under more varied scenarios and conditions.**

***Index Terms*—Combinatorial Optimization; Job Shop Scheduling; Hyper-heuristics; Recursive Solver.**

I. Introduction

DEVELOPING better methods for solving optimization problems is an area that attracts plenty of attention from the scientific community. In these problems, the goal is to minimize or maximize the value of an objective function. This leads to multiple applications in real-life activities [1]. Among the different classes of solvers for tackling these problems, Hyper-Heuristics (HHs) stand as a contemporary approach [2].

Although there are different kinds of HHs, the overall idea is to combine the strengths of a set of available solvers to improve one or more performance metrics [3]. This usually allows finding better solutions than using the solvers in a standalone fashion [4]. This work focuses on selection HHs.

Overall, HHs have proven useful in tackling diverse combinatorial problems. For example, Zhong *et al*. dealt with generating schedules for heat sinks in networks of wireless sensors [5]. Similarly, Toledo *et al*. targeted the Orienteering Problem with Hotel Selection, where an optimal tour including hotels and points of interest must be found [6]. Moreover, Bai *et al*. developed a general-purpose hyper-heuristic for solving bin packing and course timetabling problems [7].

This work was supported by Tecnologico de Monterrey, and the Mexican Council of Science and Technology CONACyT under grant number 287479.

The Job Shop Scheduling Problem (JSSP) is another problem of particular interest. For example, Da Col and Teppan compared two HHs for solving various types of instances with known optimal values [8]. Another example is the work from Garza-Santisteban *et al*., in which the authors trained a selection hyper-heuristic using Simulated Annealing [9]. Lara-Cárdenas *et al*. followed a different approach by combining unsupervised and reinforcement learning techniques [10].

Despite previous research efforts, it is customary for HHs to fail at solving some of the available instances. In some cases, this is due to excessively simple action regions for each low-level solver. Despite this, literature lacks a generalized model that helps to improve the generalization capabilities of HHs. In this work, we target such a gap by introducing the concept of Recursive Hyper-Heuristics (RHH) and testing its feasibility on some instances of the JSSP. Our idea is to recursively add selection layers to the model, which allows for the overlap of action regions that result in more complex interactions between low-level solvers. Additionally, this approach offers other benefits, such as the ability to reuse trained HHs. Since space is limited, we bound our scope to some initial experiments.

The remainder of this manuscript is organized as follows. Section II briefly describes fundamental ideas related to our research. Then, Section III describes the proposed recursive model. We split experiments in two sections: one for the methodology (Section IV) and one for the data (Section V). Finally, we discuss the main highlights in Section VI.

II. Fundamentals

A. Job Shop Scheduling Problem

We can split Job Shop Scheduling Problems into different categories [11]. Still, we focus on the traditional JSSP, where jobs are composed of operations. Each operation must be carried out in a specific machine, which can only perform one operation simultaneously. The goal is to find a schedule that reduces the makespan (the time taken to complete all jobs).

There are different approaches to solving JSSPs. Some methods include the combination of constraint programming and local search algorithms [12]. Dispatching rules and low-level heuristics seek to take advantage of expert knowledge to derive direct approaches that allow quick decision-making [13]. Examples of this approach include the work from Mencía *et al*. [14] and the review from Branke *et al*. [15].

In the case of metaheuristics, several works stand out, such as those related to Evolutionary Algorithms [16], Particle Swarm Optimization [17], and Ant Colony Optimization [18]. Hyper-heuristics have also been used, as we mentioned in the introduction. Additionally, we can find the work from Garza-Santisteban *et al*. on feature transformations [19], and from Bai and Blazewicz about a flexible hyper-heuristic model [7].

In some HH approaches, we may need to map the current state of a problem instance by using a set of features, which can be diverse. Mirshekarian and Šormaz provided a comprehensive review of features for the JSSP, where they exposed a set of 380 features divided into eight categories [20]. Some of these categories relate to the overall components of a problem instance, such as the number of jobs and machines it contains. Others relate to statistical data about these components, such as mean and median values. Additional categories relate to more complex features, *i.e.*, those requiring calculating several intermediate parameters. There is also a category for the performance of low-level heuristics over the problem instance.

Finally, it is essential to comment on the sets of instances that we may find in the literature. Traditional instances include those from Taillard [21] and Demirkol [22], as well as those available in the JSPLIB[1]. These instances are usually generated through random procedures disregarding each solver's nature. That is why Vela *et al*. recently proposed a different approach using a metaheuristic for tailoring easy and hard instances for specific heuristics of the JSSP [23]. To do so, they defined an objective function based on the difference between the performance of the target solver and the remaining ones.

B. Hyper-heuristics

Hyper-heuristics has been an active research area for the last two decades. The term was coined during 2000 [24]. Although initially described as "heuristics to choose heuristics," this approach has been used to tackle various problems through selection and generation strategies [4]. For example, Nguyen *et al*. proposed a genetic programming-based hyper-heuristic approach for three combinatorial and optimization problems: Max-SAT, one-dimensional bin packing, and permutation flow shop [25]. Similarly, Sim and Hart described an immune-inspired hyper-heuristic system that produces new heuristics for the bin-packing and job-shop scheduling problems [26]. Later, Sabar and Kendall described a Monte Carlo tree-search hyper-heuristic, which evolved heuristics for five problems: Max-SAT, one-dimensional bin packing, permutation flow shop, traveling salesman, and personnel scheduling [27].

More recent works include the one from Ya Su *et al*., who presented a model based on the Whale Optimization Algorithm (WOA) [28]. Furthermore, the work from Sánchez-Díaz *et al*. introduced a feature-independent hyper-heuristic for solving the Knapsack Problem, powered by an evolutionary algorithm [29]. Later on, Fakhrud Din and Kamal Z. Zamli presented a hyper-heuristic strategy for input-output-based interaction testing [30]. Such a strategy used an Exponential Monte Carlo with Counter (EMCQ) approach as its high-level heuristic and three metaheuristics as low-level solvers: Genetic Algorithm, Teaching Learning-Based Optimization, and Flower Pollination Algorithm (FPA). Recently, Rodríguez *et al*. tackled the Traveling Thief Problem through a sequence-based selection hyper-heuristic [3]. Hyper-heuristics have also been used for the automatic design of metaheuristics [31].

[1] https://github.com/tamy0612/JSPLIB

III. Proposed Approach

In this work, we propose a recursive hyper-heuristic model in which the order of the hyper-heuristic indicates the depth of the recursive process. At every step of the recursion, our proposed model contains a selection layer akin to traditional rule-based selection hyper-heuristics. Hence, whenever the model must decide, it begins at the outermost layer and works through the layers until it arrives at a low-level heuristic (*e.g.*, Fig. 1). Throughout this process, a different subset of features can be used at each layer, for added flexibility.

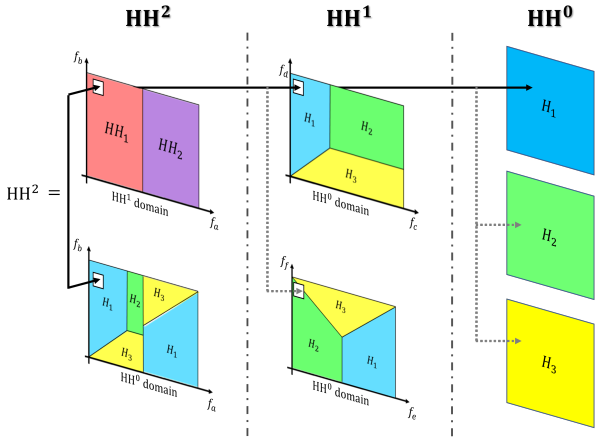


Fig. 1: An example of the mapping from features to actions in our proposed recursive model. Note that f_a and f_b can be the same or different features from f_c, f_d, f_e, and f_f.

Let us analyze some quick examples, beginning with low-level heuristics. Since these solvers do not select other heuristics (*i.e.*, they attack the problem directly), they can be regarded as zeroth-order-hyper-heuristics (HH^0). A traditional selection hyper-heuristic, in contrast, chooses a solver from a pool of low-level heuristics. Hence, this model can be regarded as a first-order-hyper-heuristic (HH^1). Then we have the model proposed by Vela *et al*., which the authors called *squared hyper-heuristic* [32]. This model contains two decision layers. The first one selects a solver from a pool of traditional hyper-heuristics. The second one links those hyper-heuristics with the base low-level heuristics. Therefore, it can be regarded as a second-order-hyper-heuristic (HH^2).

Building upon these examples, a cubic hyper-heuristic (CHH) would be equivalent to a third-order-hyper-heuristic (HH^3), which would be able to choose second-order solvers. Hence, an n^{th}-order-hyper-heuristic (HH^n) is a solver that selects hyper-heuristics of $(n-1)^{\text{th}}$ order. Also, bear in mind that this definition depends upon the inner solver with the

highest order. So, a given recursive hyper-heuristic can contain solvers from its previous order or even lower ones (up to the base heuristics themselves). This adds extra flexibility to the model and allows it to create complex action regions for each heuristic, as shown in Fig. 2. Moreover, this flexibility may be of benefit for reusing previously trained models. Imagine that we have a competent HH^1 for a given subset of instances, but must expand it to cover a new instance subset. Hence, one could train a new HH^1 for the new subset and then train an HH^2 that learns to differentiate between both instance subsets. This can also go as far as to incorporate a different subset of features for each layer, should that benefit the overall performance.

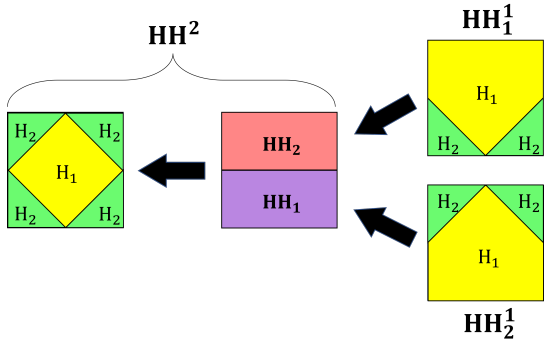


Fig. 2: An example on how recursive hyper-heuristics integrate solvers from different layers to create more complex regions.

We summarize our proposal in Fig. 3 for any given hyper-heuristic of order n. Each column represents a given feature for the model ($\forall\, j = 1, \ldots, f$), whereas each row represents a given rule ($\forall\, i = 1, \ldots, r$). Such a rule includes a solver s_i that will be applied whenever that rule is selected. Up to this point, the proposal is akin to a traditional hyper-heuristic. However, the set of solvers S comprises all the available solvers at lower levels, *i.e.*, $s_i \in S = \{\text{HH}^0, \text{HH}^1, \ldots, \text{HH}^{n-1}\}$, instead of exclusively containing low-level heuristics. Remember that each inner hyper-heuristic, except for HH^0, is also represented by this same model. Hence, the recursive term. This means that each inner solver also has a given set of features and a fixed number of rules, which may differ from those used in the outermost model. Whenever a decision is requested, the outermost layer (n) selects a solver from the next layer ($n-1$) and then uses the model of this solver to choose a solver from the next layer ($n-2$). This process continues until we return to the low-level heuristics, *i.e.*, at layer 0.

A. Heuristics

In this work, we consider four heuristics for the JSSP. All of them select an activity that complies with the problem constraints. Hence, any solution, partial or complete, is valid and feasible. This indicates that for each job, heuristics only choose from among the available activities.

We include two heuristics based on the processing times of activities. One prioritizes scheduling jobs with longer processing times, the Longest Processing Time (LPT) heuristic, while

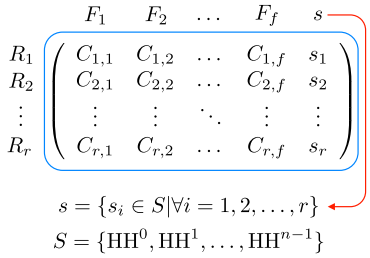


Fig. 3: Overview of the rule-based recursive hyper-heuristic model. R_i: Rule of the model. F_j: Feature that the model incorporates. s_i: Solver associated with rule R_i. $C_{i,j}$: Coordinate of rule R_i for feature F_j. HH^k: Solvers available at depth k, where 0 indicates low-level heuristics.

the other prioritizes scheduling smaller jobs first, the Shortest Processing Time (SPT) heuristic. Similarly, we include two heuristics that aim to schedule a whole job quickly or evenly schedule them all. To achieve this, we target each job's pending activities. Thus, MPA intends a breadth-first approach since it selects the job with the Most Pending Activities. In contrast, LPA employs a depth-first strategy, aiming to finish jobs with the fewest pending activities. If two or more jobs are valid options for any heuristic, it selects the one with the lowest ID.

B. Features

For the HH models created in this article, we used four features proposed by Mirshekarian *et al.* [20] and the naming convention proposed by Vela *et al.* [32]. Hence, we use a five-letter code associated with the first author and an ID extracted from the list of features that the authors used. Two of these features are given by ratios between the standard deviation and the mean of a given set of processing times. In this sense, Mirsh015 and Mirsh029 consider those values related to the jobs and the machines, respectively. The third feature, *i.e.*, Mirsh222, relates upcoming activities within the instance with processing times and machine usage. Finally, feature Mirsh282 targets a ratio related to the vacancy associated with machines. Please refer to [20], [32] for more details about calculating such features.

C. Instances

Throughout this research, we considered two sets of instances. Table I summarizes the information from the first dataset. We include two *toy* instances, which are small, simple problems that are easily solved by hand. Similarly, Table II provides information about the second dataset, which contains 240 instances. Because of the size of this dataset, it is impossible to provide a complete description of each instance, as with the first one. Nonetheless, the dataset can be organized into eight batches of 30 instances, as the table indicates. More information about the instances can be found in [23]. Additionally, we have procured such instances and may provide them upon request.

TABLE I: Instances for preliminary tests and their initial feature values. The number of operations for each instance equals the number of machines. Toy instance 1 and 2 were handcrafted, and the remaining instances were taken from the literature [23].

Name	Type	No. Jobs	No. Machines	Mirsh222	Mirsh015	Mirsh029	Mirsh282
Toy Instance 1	Handcrafted	2	2	0.0000	0.3270	0.0000	0.0000
Toy Instance 2	Handcrafted	2	2	0.0556	0.0303	0.0262	0.1481
Instance 1	LPTvsAll	3	4	0.2812	0.4380	0.5413	0.6667
Instance 2	SPTvsAll	3	4	0.0749	0.1805	0.1765	0.4074
Instance 3	MPAvsAll	3	4	0.0751	0.0615	0.3064	0.4074
Instance 4	SPTvsAll	3	4	0.2544	0.3334	0.3767	0.6667
Instance 5	LPAvsAll	3	4	0.0000	0.4923	0.4015	0.2963
Instance 6	MPAvsAll	3	4	0.1481	0.1609	0.5301	0.625
Instance 7	LPAvsAll	3	4	0.0973	0.1164	0.3128	0.4074

TABLE II: Instance subsets associated with the main testing stage and their average initial feature values. The dataset contains four types of instances, which were generated for favoring a particular heuristic (indicated in its name) [23].

Type	Batch	Instance IDs	Mirsh222	Mirsh015	Mirsh029	Mirsh282
LPAvsAll	1, 5	1:30, 121:150	0.1480	0.1979	0.3809	0.4914
MPAvsAll	2, 6	31:60, 151:180	0.0807	0.1309	0.3519	0.4321
SPTvsAll	3, 7	61:90, 181:210	0.1776	0.1976	0.3427	0.5321
LPTvsAll	4, 8	91:120, 211:240	0.1726	0.2631	0.4161	0.5257

D. Computational cost

We want to wrap this section up by commenting on the computational cost of our proposal. As Vela *et al.* showed, the difference between using a traditional HH and a second-order HH is a constant value of two [32]. This, of course, is assuming that the second-order HH contains previously trained HHs. Otherwise, the cost would quickly escalate since every candidate HH^2 would require the training of several independent HH^1. Moreover, if we extend their analysis, we arrive at a computing cost that increases linearly w.r.t. the number of layers, *i.e.*,:

$$\mathcal{O}(N_{tr} * N_a * N_j * N_m * (F_W + n * N_r * N_f + n * N_r + H)) \quad (1)$$

where n is the number of layers in the model. Additionally, N_{tr} is the number of training instances. At the same time, N_a, N_j, N_m, N_r, and N_f are the number of search agents used during the training process; jobs and machines within the instance; and rules and features within the model, respectively. Finally, F_W and H stand for the highest cost of the features and heuristics, respectively.

IV. Methodology

In this work, we analyze the feasibility of using a recursive hyper-heuristic model for improving rule-based selection hyper-heuristics. We target the Job Shop Scheduling Problem (JSSP). Additionally, we propose a two-fold experimental methodology using the MatHH framework [33].

A. Preliminary tests

Table III summarizes our first approach. We included an experiment identified as zero that uses toy instances. This allows us to glimpse at possible patterns that may arise further on. Most of the remaining experiments consist of defining a second-order hyper-heuristic through different combinations of base hyper-heuristics. The only exceptions are experiments one and four, where we also include a low-level heuristic to validate that the model can handle them adequately. The last experiment from this stage focuses on a third-order hyper-heuristic formed by a second-order hyper-heuristic and a low-level heuristic. Our goal is to analyze whether it is possible to have models with arbitrary combinations of hyper-heuristics that properly tackle different subsets of instances. Note that each experiment only uses a subset of instances, given by the ones that the inner solvers can tackle adequately. We do this to validate whether the proposed models can replicate the process defined by each inner solver.

B. Main tests

Our main testing stage revolves around five experiments, as shown in Table IV. We evaluated each experiment with a set of 240 instances. The first experiment intends to test the solvers from the previous section. The second one evaluates the performance of a single randomly generated second-order hyper-heuristic. Each of the remaining experiments evaluates the performance of 100 random recursive hyper-heuristics of different orders. In doing so, we hope to analyze the models' performance tailored to a few instances, as well as how the variability of the models evolves as we add layers.

V. Results

We now present the most relevant data associated with our work. For clarity, we preserve the same structure from Section IV. Plus, data can be freely accessed at https://github.com/iamaya2/rhh_jssp_cec23.

A. Preliminary tests

For experiment zero, we were able to replicate the oracle, thus outperforming the low-level heuristics. This, of course, is only sometimes possible, especially when the problem grows. Still, this idea shall apply recursively, *i.e.*, to models that select other high-level models. Table V presents the performance of

TABLE III: Description of the experiments performed in the preliminary testing stage. All experiments consider a single run and relate to handcrafted models. T: Toy instance. I: Instance from the literature. See Table I for more details about instances.

Experiment	Model	Inner Solvers	Rules	Features	Depth	Instance solved
0	HH0	LPT, SPT	2	Mirsh222	1	T1, T2
1	SHH1	HH1, MPA	4	Mirsh282	2	I1:I3
2	SHH2	HH1, HH2	3	Mirsh282	2	I1:I4
3	SHH3	HH1, HH3	5	Mirsh029, Mirsh015	2	I1:I3, I5
4	SHH4	HH1, HH3, MPA	6	Mirsh029, Mirsh015	2	I1:I3, I5, I6
5	SHH5	HH1, HH2, HH3	3	Mirsh222, Mirsh282, Mirsh029	2	I1:I5
6	CHH1	SHH5, LPA	3	Mirsh015, Mirsh029	3	I1:I5, I7

TABLE IV: Description of the experiments performed in the main testing stage, with recursive hyper-heuristics at different depth levels. Superscripts indicate the depth of the model. All experiments target the whole dataset from Table II. Moreover, experiments 2-5 correspond to randomly generated models using feature Mirsh222.

Experiment	Model	Inner Solvers	Rules	Runs	Depth
1	SHH1:SHH5, CHH1	See Table III			
2	SHH6	HH1:HH3	3	1	2
3	HH^2	HH1:HH3	3	100	2
4	HH^3	SHH1:SHH5	5	100	3
5	HH^4	CHH1, Best HH^4	2	100	4

different baseline approaches. Do note that we include low-level heuristics, and a few hyper-heuristics since we are interested in analyzing performance at a higher level. Additionally, we provide the performance of a synthetic oracle. A word of caution: such an oracle is useful for comparison purposes, but unfeasible in practice (it brute-forces the solution). It is interesting to see that all the solvers work well over a few instances and poorly over others, which indicates that this subset is varied enough. Even so, these base hyper-heuristics fail to find the best solution for three of the instances. Hence, there is still room for improvement.

Table VI summarizes the performance of all the models associated with the experiments from this stage and across their corresponding instances. In virtually all cases, it was possible to replicate the oracle. The only exception was instance 1, where the difference remained at a single time unit. We deem this noteworthy since, in the end, all of them end up using the same four base low-level heuristics. What does change is that the interaction between solvers becomes more complex, which allows for a better adaptation of the model to the instances. However, this should be taken with caution, as it could lead to an overfitting of the model.

Fig. 4 shows the action regions of the solvers within each model for all six experiments. Note that the resulting models have a single feature in the first two cases, so we opted to repeat it in both axes for visualization purposes. Also, it is interesting that SHH3 and SHH4 exhibit virtually the same regions. The only difference is a zone (in SHH4) that targets the MPA heuristic. This addition allows the model to solve an additional instance (*i.e.*, instance 6). Additionally, the model for SHH5 may look simple, but it requires three features (Mirsh029, Mirsh222, and Mirsh282). Finally, the last model (CHH1) looks like a single solver. Nonetheless, it is a third-order hyper-heuristic that contains the second-order model from experiment five (SHH5) and a low-level heuristic (LPA). Hence, at a second-order level, the action regions would look like those from Fig. 4e with the addition of the small blue chunk shown in Fig. 4f. Thus, our proposed recursive model easily incorporates existing solvers for targeting specific cases.

B. Main tests

Fig. 5 summarizes the main testing stage. Here, we use horizontal dashed lines to indicate the performance of those experiments incorporating a single run, *i.e.*, the first two experiments. It is worth clarifying that we have six models (SHH1–SHH5, and CHH1) in the first case. So, we only display the performance of the best (SHH4) and worst (SHH2) models, for readability. We also include a black horizontal dashed line to indicate the performance of the oracle. Additionally, we use violins to show the distribution of the average performance of randomly generated models (experiments three to five).

As the model goes deeper (*i.e.*, order increases), its performance improves, and its variability diminishes. In experiment five (fourth-order hyper-heuristic model), most runs can be sorted into two groups: one with a performance of about 46 and one with about 50. This change is more evident than the one that occurs when migrating from a second-order model to a third-order one (experiments three and four, respectively). Nonetheless, the performance of these models still lags far from the oracle. Let us remind the reader that these models are randomly generated. Hence, one would expect that the models perform better after a proper training scheme. Another element worth noticing is that the performance of models from experiment one falls within that of experiments three and four. This is somewhat expected, and it indicates that the human designer from experiment one performed averagely.

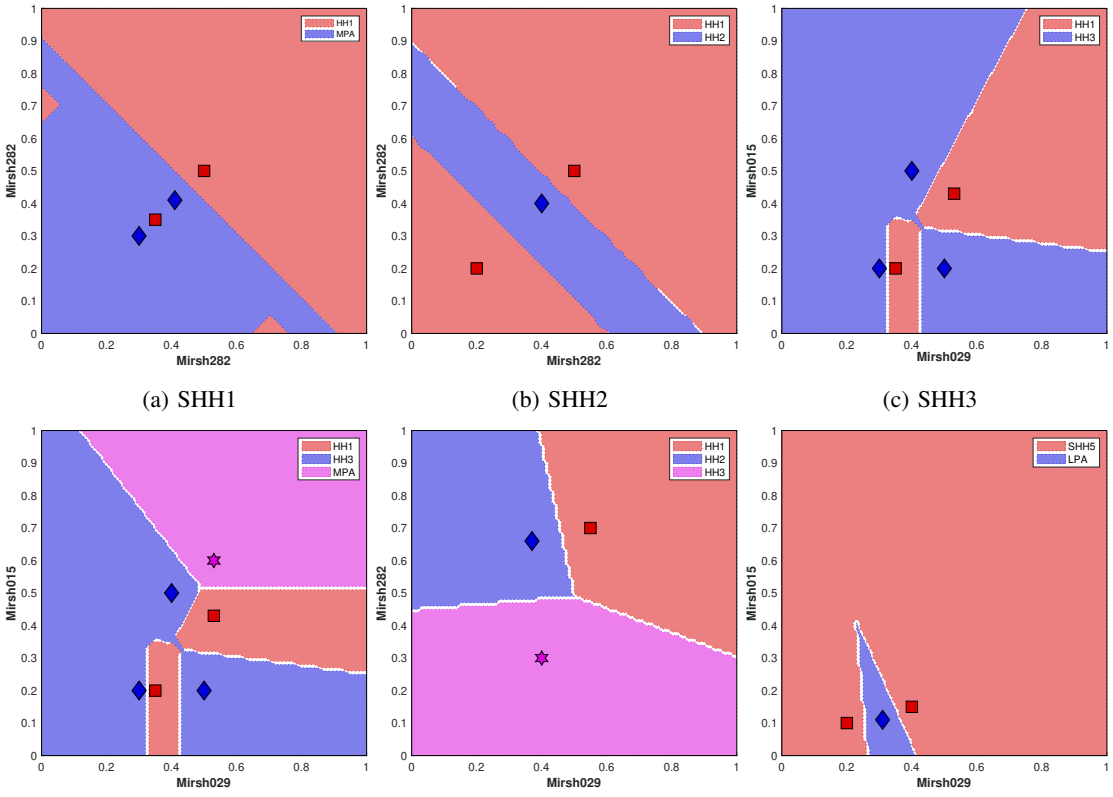
Fig. 6 shows a heatmap with the normalized (per column) performance for each available solver across every instance. Since experiments 3–5 consider the generation of 100 random

TABLE V: Performance of the base solvers on the instance dataset associated with preliminary tests, and the resulting synthetic oracle. Values in red and green represent the worst and best performance for the instance (column), respectively.

Type	Solver	Instance 1	Instance 2	Instance 3	Instance 4	Instance 5	Instance 6	Instance 7
Heuristic	LPT	**41.0000**	**41.0665**	72.4897	48.1692	**40.0000**	80.0000	48.9463
	SPT	**61.0000**	**21.0665**	**73.0239**	**28.1692**	**40.0000**	**81.3152**	48.9464
	MPA	58.0000	**41.0665**	**33.0303**	48.1692	**40.0000**	**50.0000**	48.9463
	LPA	**41.0000**	**41.0665**	72.4882	**50.8974**	**20.0000**	**81.3152**	**28.9472**
Traditional hyper-heuristic	HH1	42.0000	**21.0665**	**73.0239**	50.8885	**40.0000**	**81.3152**	48.9464
	HH2	**61.0000**	**21.0665**	**33.0303**	**28.1692**	**40.0000**	61.3152	**48.9465**
	HH3	49.0000	**41.0665**	**33.0303**	**50.8974**	**20.0000**	70.0000	48.9463
Synthetic	Oracle	**41.0000**	**21.0665**	**33.0303**	**28.1692**	**20.0000**	**50.0000**	**28.9472**

TABLE VI: Performance of the recursive solvers and a synthetic oracle on the preliminary tests. Red: Experiments that failed to replicate the synthetic oracle. Dashed values: Instances that were not part of the corresponding experiment (see Table III).

Experiment ID	Solver	Instance 1	Instance 2	Instance 3	Instance 4	Instance 5	Instance 6	Instance 7
–	Oracle	41.0000	21.0665	33.0303	28.1692	20.0000	50.0000	28.9472
1	SHH1	**42.0000**	21.0665	33.0303	–	–	–	–
2	SHH2	**42.0000**	21.0665	33.0303	28.1692	–	–	–
3	SHH3	**42.0000**	21.0665	33.0303	–	20.0000	–	–
4	SHH4	**42.0000**	21.0665	33.0303	–	20.0000	50.0000	–
5	SHH5	**42.0000**	21.0665	33.0303	28.1692	20.0000	–	–
6	CHH1	**42.0000**	21.0665	33.0303	28.1692	20.0000	–	28.9472



(a) SHH1 (b) SHH2 (c) SHH3

(d) SHH4 (e) SHH5 (f) CHH1

Fig. 4: Overview of the action regions yielded by each hyper-heuristic model from the preliminary testing stage. Each marker represents a rule of the model and the shaded regions show their corresponding zone of effect.

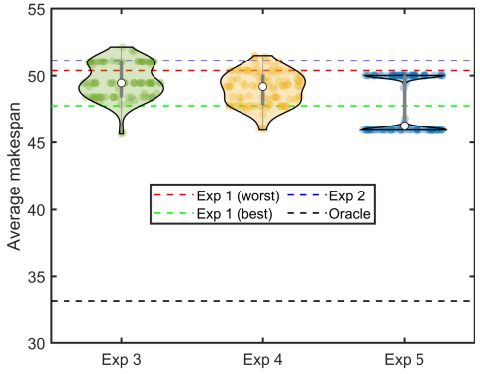


Fig. 5: Performance distribution of experiments (Exp) with randomly generated recursive hyper-heuristics (100 runs each) and their comparison against single-run models from previous experiments. See Table IV for details about experiments.

models, we opt for using the average performance across all runs. First, we can corroborate that different batches of instances are best solved with different heuristics. This is expected since it reflects the nature of the instances [23]. Even so, for some cases, the remaining heuristics tend to perform at a mid-level, *e.g.*, the region near instance 140. For others, such as near instance 170, all the remaining heuristics perform poorly.

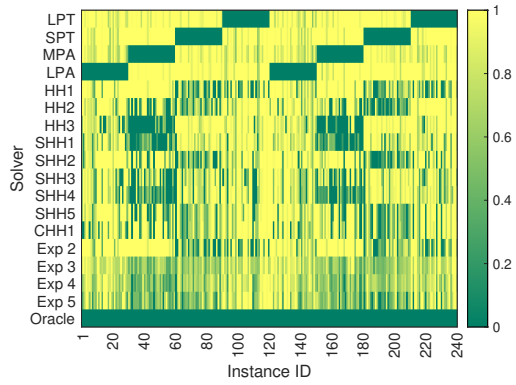


Fig. 6: Performance of all solvers and a synthetic oracle, when solving a dataset with 240 instances. Data are normalized per instance (column). For experiments (Exp) 3-5, data correspond to the average performance of 100 randomly generated models over each instance. See Table IV for details about experiments.

Traditional hyper-heuristics focus on a few instance batches. For example, HH1 performs great in some instances of batches 3, 4, 6, and 7 but poorly in most instances of batch 2. Similarly, HH3 performs well in the second and sixth batches, *i.e.*, instances 30–59 and 150–179. We can see that the best solutions from the squared models (SHH1–SHH5) are achieved on instances from different sets. The cubic model (CHH1) performs quite similarly to its base solver (SHH5) with some improvements, especially for instances from the first batch. This is due to the addition of the LPA heuristic, which added some flexibility (*cf.* Figs. 4e and 4f).

Experiments three to five are the core of this testing stage. As we have already mentioned, such experiments contain 100 hyper-heuristics each. Although performance on a single instance is not as good as for the previous solvers, the overall performance is more evenly distributed. Even so, specific instances remain poorly solved, such as instances with an ID near 120. Additionally, the overall performance seems to improve as the model grows in complexity. Such is the case with instances with IDs between 60–80 and 160–200. In these cases, the model from experiment five (a fourth-order hyper-heuristic) performs better than those from experiments two and three (second and third-order hyper-heuristics, respectively). Overall, the fourth-order model yields a better average performance across the dataset (47.8854) than the one yielded by the second and third-order models (49.6589 and 49.0445, respectively).

VI. Conclusions

In this work, we analyzed the feasibility of a recursive hyper-heuristic model. We considered two batches of tests: a reduced one (nine instances) and an extended one (240 instances).

We found interesting patterns. For starters, increasing the order of the hyper-heuristic model seems useful. This was especially marked for the fourth-order models, where data was distributed around two values. Additionally, there is a small performance gain derived from the ability to combine solvers in more complex patterns. For example, the randomly generated hyper-heuristics exhibited a median makespan of 49.46, 49.19, and 46.23 for the second, third- and fourth-order models. This represents a reduction of over 6% from simply using a deeper model. We are confident that such a reduction can be further improved by following a more robust approach.

However, we have yet to fully grasp the extent of the benefits related to our proposed approach. Hence, we cannot comment on the maximum value of n that one should use, and such a value likely depends on the problem. In some scenarios, it may suffice to stop at $n = 2$. In others, one may require going up to $n = 5$. Still, we have yet to devise an approach for estimating a proper value of n for a given problem. Thus, we opted for a flexible enough approach that allows the incorporation of as many layers as required. Also, the extra layers offer other benefits. A scenario that comes to mind is when accommodating previously trained models to tackle a new set of instances or to consider a different set of features.

Plenty of work lies ahead, which can be split into diverse paths. The most evident one is a proper training scheme for the recursive model. For simplicity, here we used randomly generated models. However, this leads to a variable performance. Hence, implementing a training scheme *e.g.*, based on metaheuristics should help. Nevertheless, the recursive nature of the proposed model imposes new challenges. For example, hyper-heuristics must also learn the most suitable depth during training. This requires an approach in which the number of

design variables can change across iterations, since each layer requires a new selector. This, in turn, requires several variables per decision rule.

Another path is the definition of more intricate models, where one may incorporate different hyper-heuristic models and even solvers of a completely different nature, *e.g.*, a neural network. Of course, it is also crucial to analyze the behavior of this recursive model on other combinatorial problems, such as timetabling. Finally, efforts could be directed towards the modularity offered by our proposed approach, seeking to benefit from the rules learned by one model to reduce the burden of training a new one. This can be achieved by creating a higher-level model that uses the existing one as a base solver, along with the new pool of solvers, and training the model to determine when to use the 'old' model and when to use the 'new' one. Indeed, there are many possible applications, and we plan to keep working on this approach.

REFERENCES

[1] A. Ben-Tal, D. Den Hertog, A. De Waegenaere, B. Melenberg, and G. Rennen, "Robust solutions of optimization problems affected by uncertain probabilities," *Management Science*, vol. 59, no. 2, pp. 341–357, 2013.

[2] J. Denzinger, M. Fuchs, and M. Fuchs, "High performance ATP systems by combining several AI methods," in *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI 97)*, (CA, USA), p. 102–107, 1997.

[3] D. Rodríguez, J. M. Cruz-Duarte, J. C. Ortiz-Bayliss, and I. Amaya, "A Sequence-Based Hyper-Heuristic for Traveling Thieves," *Applied Sciences*, vol. 13, pp. 1–23, dec 2023.

[4] M. Sanchez, J. M. Cruz-Duarte, J. C. Ortiz-Bayliss, H. Ceballos, H. Terashima-Marin, and I. Amaya, "A Systematic Review of Hyper-Heuristics on Combinatorial Optimization Problems," *IEEE Access*, vol. 8, pp. 128068–128095, 2020.

[5] J. Zhong, Z. Huang, L. Feng, W. Du, and Y. Li, "A hyper-heuristic framework for lifetime maximization in wireless sensor networks with a mobile sink," *IEEE/CAA Journal of Automatica Sinica*, vol. 7, no. 1, pp. 223–236, 2020.

[6] A. Toledo, M. C. Riff, and B. Neveu, "A Hyper-Heuristic for the Orienteering Problem with Hotel Selection," *IEEE Access*, vol. 8, pp. 1303–1313, 2020.

[7] R. Bai, J. Blazewicz, E. K. Burke, G. Kendall, and B. McCollum, "A simulated annealing hyper-heuristic methodology for flexible decision support," *4OR*, vol. 10, pp. 43–66, 3 2012.

[8] G. Da Col and E. Teppan, "Google vs IBM: A Constraint Solving Challenge on the Job-Shop Scheduling Problem," *Electronic Proceedings in Theoretical Computer Science*, vol. 306, pp. 259–265, 2019.

[9] 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)*, pp. 57–64, IEEE, 6 2019.

[10] E. Lara-Cárdenas, A. Silva-Gálvez, J. C. Ortiz-Bayliss, I. Amaya, J. M. Cruz-Duarte, and H. Terashima-Marín, "Exploring Reward-based Hyper-heuristics for the Job-shop Scheduling Problem," in *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, pp. 1–8, 2020.

[11] P. Taylor, "Addressing the gap in scheduling research : a review of optimization and heuristic methods in production scheduling Addressing the gapin scheduling research : a review of optimization and heuristic methods in production scheduling," *Manufacturing Engineering*, no. 786636650, pp. 37–41, 1993.

[12] J.-P. Watson and J. Beck, "A hybrid constraint programming / local search approach to the job-shop scheduling problem," vol. 5015, pp. 263–277, 05 2008.

[13] L. Li, S. Zijin, N. Jiacheng, and Q. Fei, "Data-based scheduling framework and adaptive dispatching rule of complex manufacturing systems," *International Journal of Advanced Manufacturing Technology*, vol. 66, no. 9-12, pp. 1891–1905, 2013.

[14] C. Mencía, M. R. Sierra, and R. Varela, "Depth-first heuristic search for the job shop scheduling problem," *Annals of Operations Research*, vol. 206, no. 1, pp. 265–296, 2013.

[15] J. Branke, S. Nguyen, C. W. Pickardt, and M. Zhang, "Automated Design of Production Scheduling Heuristics: A Review," *IEEE Transactions on Evolutionary Computation*, vol. 20, no. 1, pp. 110–124, 2016.

[16] T. C. Cheng, B. Peng, and Z. Lü, "A hybrid evolutionary algorithm to solve the job shop scheduling problem," *Annals of Operations Research*, vol. 242, no. 2, pp. 223–237, 2016.

[17] L. Gao, X. Li, X. Wen, C. Lu, and F. Wen, "A hybrid algorithm based on a new neighborhood structure evaluation method for job shop scheduling problem," *Computers and Industrial Engineering*, vol. 88, pp. 417–429, 2015.

[18] M. Seo and D. Kim, "Ant colony optimisation with parameterised search space for the job shop scheduling problem," *International Journal of Production Research*, vol. 48, no. 4, pp. 1143–1154, 2010.

[19] F. Garza-Santisteban, I. Amaya, J. Cruz-Duarte, J. C. Ortiz-Bayliss, E. Ozcan, and H. Terashima-Marin, "Exploring Problem State Transformations to Enhance Hyper-heuristics for the Job-Shop Scheduling Problem," in *2020 IEEE Congress on Evolutionary Computation (CEC)*, pp. 1–8, IEEE, 7 2020.

[20] S. Mirshekarian and D. N. Šormaz, "Correlation of job-shop scheduling problem features with scheduling efficiency," *Expert Systems with Applications*, vol. 62, pp. 131–147, 2016.

[21] E. Taillard, "Benchmarks for basic scheduling problems," *European Journal of Operational Research*, vol. 64, no. 2, pp. 278–285, 1993.

[22] E. Demirkol, S. Mehta, and R. Uzsoy, "Benchmarks for shop scheduling problems," *European Journal of Operational Research*, vol. 109, no. 1, pp. 137–141, 1998.

[23] A. Vela, J. M. Cruz-Duarte, J. C. Ortiz-Bayliss, and I. Amaya, "Tailoring Job Shop Scheduling Problem Instances through Unified Particle Swarm Optimization," *IEEE Access*, vol. 4, pp. 1–1, 2021.

[24] P. I. Cowling, G. Kendall, and E. Soubeiga, "A hyperheuristic approach to scheduling a sales summit," in *PATAT*, 2000.

[25] S. Nguyen, M. Zhang, and M. Johnston, "A genetic programming based hyper-heuristic approach for combinatorial optimisation," in *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*, GECCO '11, (New York, NY, USA), pp. 1299–1306, ACM, 2011.

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

[27] N. R. Sabar and G. Kendall, "Population based monte carlo tree search hyper-heuristic for combinatorial optimization problems," *Information Sciences*, vol. 314, pp. 225–239, 2015.

[28] Y. Su, Y. Dai, and Y. Liu, "A hybrid hyper-heuristic whale optimization algorithm for reusable launch vehicle reentry trajectory optimization," *Aerospace Science and Technology*, vol. 119, 2021.

[29] X. Sánchez-Díaz, J. C. Ortiz-Bayliss, I. Amaya, J. M. Cruz-Duarte, S. E. Conant-Pablos, and H. Terashima-Marín, "A feature-independent hyper-heuristic approach for solving the knapsack problem," *Applied Sciences (Switzerland)*, vol. 11, no. 21, 2021.

[30] F. Din and K. Z. Zamli, *Hyper-Heuristic Strategy for Input-Output-Based Interaction Testing*, vol. 730. Springer Singapore, 2022.

[31] J. M. Cruz-Duarte, I. Amaya, J. C. Ortiz-Bayliss, S. E. Conant-Pablos, H. Terashima-Marín, and Y. Shi, "Hyper-Heuristics to Customise Metaheuristics for Continuous Optimisation," *Swarm and Evolutionary Computation*, vol. 66, no. July 2020, p. 100935, 2021.

[32] A. Vela, J. M. Cruz-Duarte, J. C. Ortiz-Bayliss, and I. Amaya, "Beyond hyper-heuristics: A squared hyper-heuristic model for solving job shop scheduling problems," *IEEE Access*, vol. 10, pp. 43981–44007, 2022.

[33] J. M. Cruz-Duarte, J. C. Ortiz-Bayliss, and I. Amaya, "MatHH: A Matlab-based Hyper-Heuristic framework," *SoftwareX*, vol. 18, p. 101047, 2022.