Exploring Reward-based Hyper-heuristics for the Job-shop Scheduling Problem

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

***Abstract*—The Job-Shop Scheduling Problem represents a challenging field of study due to its *NP*-Hard nature. Its many industrial and practical, real-world applications skyrocket its importance. Particularly, hyper-heuristics have attracted the attention of researchers on this topic due to their promising results in this, and other optimization problems. A hyper-heuristic is a method that determines which heuristic to apply at each step while solving a problem. This investigation aims at rendering hyper-heuristics by combining unsupervised and reinforcement learning techniques. The proposed solution applies a clustering approach over the feature space, and then, it generates knowledge about heuristic selection through a reward-based system. Results show that our hyper-heuristics surmount competent heuristics, such as SPT and MRT, in various test instances. Besides, some of these hyper-heuristics outperformed the best result obtained among all the heuristics in more than 33% of the instances. Hence, we believe that the proposed approach is promising and that more knowledge about its benefits and limitations should be derived through its application on different problems.**

***Index Terms*—Job Shop Scheduling, Hyper-heuristic, Heuristics.**

I. Introduction

Scheduling problems are widespread in the industry and concern the allocation of resources to tasks over specific periods. Despite the many variants of the problem [1], [2], [3], [4], we have focused this work on the Job-Shop Scheduling Problem (JSSP) as an initial implementation of our solution approach. Solving a JSSP requires designating a set of jobs in a set of machines, subject to the constraint that each machine can only handle one job at a time. Also, each job consists of a set of ordered activities. It is common to use the maximum completion time of the jobs to evaluate the quality of a schedule (usually referred to as the makespan [5]). Therefore, the main motivation behind solving JSSPs is to minimize the makespan. This is not straightforward since the problem belongs to the non-deterministic polynomial-time (*NP*-Hard) problem class [6]. In fact, a JSSP instance has $n! \times m$ different ways to schedule n jobs in m machines [7].

Literature is prolific with methods for solving JSSPs [8], [9], [10], [11]. Although exact ones produce optimal results, they suffer from being limited to the instance size. Conversely, approximated techniques such as heuristics and metaheuristics are likely to render sub-optimal solutions. However, they scale easily. In the case of JSSPs, heuristics operate by generating schedules with few computational resources and, hence, short times. Nonetheless, they are restricted to this particular domain. Some examples of such an approach include dispatching rules [12] and the shifting bottleneck procedure [13]. In turn, Metaheuristics (MHs) are considered higher-order heuristics that work for different problem domains. Some illustrative examples of MHs and their versatility are Simulated Annealing [14], [15], [16], Tabu Search [17], [18] and Genetic Algorithms [19], [20].

Unfortunately, no single heuristic or metaheuristic performs best on every instance of the problem. A recent trend seeks to combine the strengths of such techniques in a somehow intelligent fashion, to obtain a robust performance for a broad range of problems [21], [22], [23]. The literature usually refers to selecting the most suitable algorithm (or solving strategy) for a particular situation as the algorithm selection problem. Examples of strategies based on this principle include, but are not limited to, algorithm portfolios [24], [25], [26], hyper-heuristics [27], [28], [29], and instance-specific algorithm configuration [30]. In general, all of them manage a set of solving strategies and apply the most suitable one for the current problem state of the instance under exploration. Striving to unify terms, from this point on, we use the term hyper-heuristic to refer to the methods proposed in this paper.

Although some existing hyper-heuristic approaches have incorporated reinforcement learning at different levels [31], [32], we propose to use the k-means algorithm as the initial step of hyper-heuristic creation (to decide which instances are similar to the others). Only after clusters are created, our model trains the hyper-heuristic with the reward function. To the best of the authors' knowledge, this is the first work that explores this combination of techniques for producing hyper-heuristics focused on solving JSSPs.

As mentioned, this research focuses on implementing a reward-based Hyper-Heuristic (HH) model for JSSPs. For this task, we encompass a combination of unsupervised (clustering) and reinforcement learning. The former allows the HH to identify regions where decisions are made based on features of the JSSP instance and the solution progress. Thence, the

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

system labels each cluster with a single heuristic to apply when an instance belongs to that cluster (by using Euclidean distance). The refinement of such a labeling process occurs by rewarding a good decision (the right choice of heuristic) or penalizing a bad one.

II. Background

We can define a Job-Shop Scheduling Problem (JSSP) in terms of two sets: J, containing n jobs, and M, containing m machines. The problem requires that the machines in M process each job $j \in J$ in an order given by the activity permutation $A_j = \{a_j^1, \ldots, a_j^m\}$. For each job, a non-negative integer $p_{i,j}$ represents its processing time on the i-th machine. Besides, the time in which the j-th job exits the system (the makespan of such a job) is denoted by $C^j_{\max}$, and $t_{i,j}$ denotes its completion time. In this work, we aim to minimize the makespan of the whole instance, *i.e.,* the time in which all jobs are completed.

Among the many recent strategies proposed for solving the JSSP, we can highlight hyper-heuristics (HHs). The literature usually describes them as *heuristics to choose heuristics* [33]. In some recent works, they were applied to JSSPs. Hyper-heuristics were based on Simulated Annealing [34], and on Artificial Neural Networks [35]. Both works concluded that, by using HHs, it seems feasible to overcome the individual performance of some heuristics for the JSSP by minimizing the makespan as an objective metric.

When dealing with the learning process in hyper-heuristics, most of the work focuses on providing labeled cases where a particular problem state is linked with the most suitable heuristic. This is known as supervised learning. However, little work has explored other learning scenarios, such as Unsupervised or Reinforcement Learning.

Clustering, or Unsupervised Learning (UL), is the task of grouping items by common properties. We consider clustering an unsupervised learning procedure because learning takes place with no explicit examples of what to learn, but the features of the items to cluster. Among the available clustering algorithms in the literature, k-means is probably the most popular one. It works by creating clusters and assigning the items to such groups based on a distance metric—usually the Euclidean distance due to its computation simplicity [36]. The main purpose of this algorithm is to minimize the overall distance from items to their corresponding centroids (the centers of the clusters). The centroids, which are randomly initialized, are updated through a series of iterations according to the items each cluster contains [37].

Conversely, Reinforcement Learning (RL) is a strategy that learns the characteristics of the problem space by *trial-and-error* [38]. In detail, the learning system interacts directly with its environment, and it decides which actions to perform based on a policy. The most common and straightforward strategy to implement RL is called Q-learning [38]. Before learning begins, the system initializes a Q-table to some fixed arbitrary values. Then, each time, the system selects an action, observes a reward, enters a new state (that may depend on both the previous state and the selected action), and updates the Q-table accordingly.

Some works are closely related to this investigation because they use reinforcement learning and hyper-heuristics to solve the JSSP. For example, Falcão *et al.* [39] proposed a Q-learning-based hyper-heuristic model that selects a suitable metaheuristic to apply within an optimization process, as well as their parameters. Wang [40] proposed an adaptive job-shop scheduling strategy based on the weighted Q-learning algorithm combined with clustering and dynamic search to determine the most suitable operation on each step. Shiue *et al.* [41] proposed a RL approach, based on an offline learning module and a Q-learning module, applied on real-time scheduling for a smart factory. Similarly, Shahrabi *et al.* [31] studied an RL approach with a Q-factor algorithm for the dynamic job-shop scheduling, resulting in an improved performance when compared to general variable neighborhood search methods. More recently, Zhao *et al.* [32] proposed a Q-learning algorithm with double-layer actions for flexible job-shops with machine failures, where the agent handles the activity corresponding to a job and a machine at a different machine failure status.

A. Heuristics

Similar previously published works inspired the choice of heuristics for this work [34], [35]. Thus, we have only considered constructive heuristics, which build a solution from scratch by making one decision at each step. These heuristics for the JSSP decide, at each step of the search, which job to process next among all the available options. We describe the heuristics selected for this work as follows:

- **Shortest Processing Time (SPT)** selects the activity with the shortest processing time, from the available activities to schedule.
- **Longest Processing Time (LPT)** chooses the activity with the longest processing time, from the available activities yet to schedule.
- **Maximum Job Remaining Time (MRT)** first picks the job that requires more time to finish (*i.e.,* the one with the largest sum of the processing times of its activities yet to be scheduled). Then, it returns the first possible activity (in precedence order) that corresponds to such a job.
- **Most Loaded Machine (MLM)** takes the machine with the maximum total processing time (the one with the largest sum of the processing times of the activities it has allocated) from the available machines. Then, it casts the activity with the shortest processing time, from the activities that can be allocated on the selected machine.
- **Least Loaded Machine (LLM)** works similar to MLM, but it selects the machine with the minimum total processing time from the set of machines. Then, it chooses the activity with the shortest processing time from the activities that can be allocated on the selected machine.
- **Earliest Start Time (EST)** finds the job with the earliest possible starting time at the current problem state, by

considering the available activities yet to be scheduled. Then, it picks the activity that corresponds to such a job.

In all cases, these heuristics break ties by using the job index that corresponds to its belonging activity (smaller values are preferred).

B. JSSP Characterization

A different number of features can represent the states of the solving process for a JSSP instance. As in the case of heuristics, the feature set was also influenced by previous works [34], [35]. The ones we selected are:

- **DNPT** describes the ratio between the standard deviation and the mean of the processing times of all the unscheduled jobs.
- **NAPT** is the complement of APT. It calculates the ratio between the sum of the processing times of unscheduled jobs and the sum of the processing times of the complete list of jobs (including scheduled ones).
- **NJT** determines the sum of the processing times normalized for each job. Then, it divides such an amount by the number of pending jobs. It only applies to the pending jobs.

All these features are dynamic. Then, they change as the solution process occurs since scheduling a job changes the characteristics of the remaining problem.

C. JSSP Instances

All the instances considered for this research were synthetically produced by using the algorithm proposed by Taillard [42]. This algorithm produces JSSP instances with a given number of jobs and machines, specified by the user. The processing times of the jobs were uniformly generated in the range of 1 to 99. The random generator for the processing times was proposed by Bratley *et al.* [43]. For this investigation, we set the values of the parameters of the instance generator, as recommended by Taillard [42]. By following this process, we generated 30 JSSP instances of ten jobs and ten machines. We used half of these instances for training and the remaining ones for testing. The train and test sets are mutually exclusive, which means that no instances used in the train set are also part of the test set.

III. Solution Approach

The proposed approach consists of a combination of two popular algorithms from the Machine Learning realm, *i.e., k*-means and Q-learning. These two algorithms are implemented in a sequence to produce a hyper-heuristic (HH) for the JSSP. As in other offline HHs, the model must be trained before release. Therefore, hyper-heuristic creation comprises two main phases: training and testing.

The training phase (depicted in Fig. 1) begins by collecting points from the feature space (see Sect. II-B). Each point represents a particular instance at a given step of the solution process. Hence, these points may be different depending on the heuristic used to solve the instance. Since it will be unfeasible to generate all heuristic combinations, we recorded the points generated when solving the training instances with a randomly selected heuristic for each instance in the training set. This way, we should have a higher diversity than by solving with a single heuristic. Since each instance is of size 10×10, and considering that the training set contains 15 instances, a total of 1500 points were generated (100 points per instance $\times$ 15 instances).

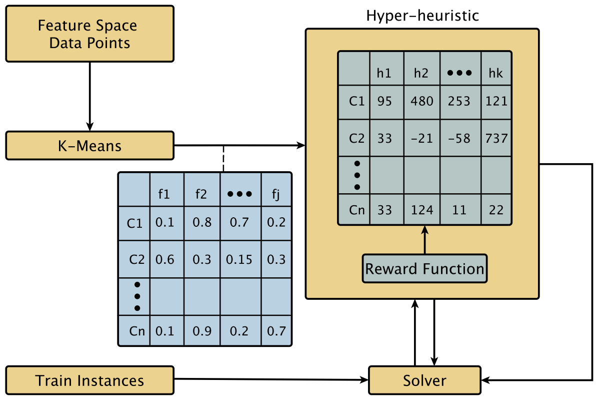

Fig. 1. Graphical representation of the training process of a hyper-heuristic with the proposed model.

Afterward, the model uses k-means to cluster the points. The motivation is to generate clusters that divide the problem space into a given number of regions. We considered 15 clusters for this work, but the model can handle different numbers with no changes. We based this choice on empirical evidence from preliminary research. A refinement step then follows, where the model learns which heuristic to use for each region. For this task, the system keeps a performance matrix. Such a matrix contains a row per cluster and a column per heuristic. Moreover, each cell contains a value representing the expected performance of a particular action. Whenever an instance arrives, the model assigns it to the closest cluster, based on the Euclidean distance. Then, it uses the performance matrix to select the heuristic with the largest expected performance. Bear in mind that the corresponding cell is updated based on the result of the decision using a reward function that only considers local performance. The reward is given by the difference between the real ($C_{\max}^{i+1}$) and synthetic ($C_{\max}'^{i+1}$) makespans of the current activity allocation, as shown in Eq. (1). The reward function forces the hyper-heuristic to act like a greedy algorithm since its behavior is only based on local decisions. It is important to recall that the real makespan corresponds to the makespan achieved after using the selected heuristic, while the synthetic one is the summation of the previous makespan and the processing time of the selected activity. Should the former be lower than the latter, the slack is reduced, and the reward is applied. Otherwise, a penalty takes place. It is essential to remark that the only scenario for the penalty occurs when the heuristic places the activity at the end of the schedule.

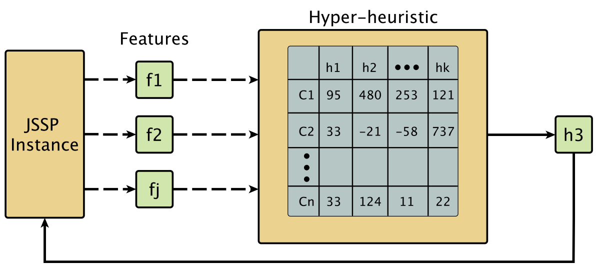

Fig. 2. Graphical representation of the testing process of a hyper-heuristic with the proposed model.

$$R = C_{\max}^{i+1} - C_{\max}'^{i+1} \tag{1}$$

When the training process is over, the hyper-heuristic no longer needs to update the performance matrix. Then, the HH solves an instance by going through an iterative process that:

1) Calculates the features of the current problem state,
2) Assigns the instance to one cluster (based on the distance between the state and its centroid), and
3) Applies the most rewarded heuristic for such a cluster to schedule the next job.

The process is repeated until no more jobs are left to be scheduled. This process is depicted in Fig. 2.

IV. Experiments and Results

As a first experimental step, we solved the test set using the six available heuristics and registered the makespan of the schedules produced. Among the six heuristics, SPT proved to be the most reliable one for the test set, generating a total makespan of 12600 hours. MRT is the second most competent heuristic, with a total makespan that is only 1.97% above the one of SPT. All heuristics but EST contribute to H*, formed by the best individual results per instance. These results are shown in Table I.

Once we calculated the heuristics and H* performance, we generated and tested three sets of hyper-heuristics. The experimental setup is two-fold: it deals with the generation process of hyper-heuristics and with performance measurement on different scenarios while focusing on obtaining insights on hyper-heuristic behavior. Table II summarizes the three experiments we carried out. For each scenario, we generated 15 hyper-heuristics, which are sequentially numbered, as to avoid confusion. In all the tests, we compared each hyper-heuristic performance against H*, which represents the best solution achieved by heuristics for each available instance (Table I). Hence, if the hyper-heuristic matches H* for one particular instance, it means that the hyper-heuristic is, at least, as useful as the best heuristic for such an instance.

We considered three different values for training the hyper-heuristics for the number of iterations over the training set: 20, 50, and 100 iterations. Although this number may seem small, remember that the performance matrix is updated every time a job is scheduled. Since we are dealing with instances of size 10×10, an iteration is equivalent to 100 updates of the performance matrix per instance. Hence, after one iteration, the performance matrix has been updated 1500 times (100 updates per instance times 15 instances). So, when considering 20, 50, and 100 iterations, we are considering 30000, 75000, and 150000 updates of the performance matrix, as Table II shows.

Tables III, IV, and V depict the results obtained by hyper-heuristics HH01 to HH15, HH16 to HH30, and HH31 to HH45, respectively, on the test set. For each hyper-heuristic, we present the makespan of the schedules produced (in hours). The table also presents the sum of the makespans over the test set per hyper-heuristic and their success rate. The success rate (ρ) indicates the percentage of instances in which a hyper-heuristic behaves, at least, as well as the best available heuristic for a particular instance.

The results obtained by the hyper-heuristics are encouraging. Among the 45 hyper-heuristics produced for this work, six outperform the total makespan required by SPT, the best heuristic on the test set. Also, 13 of these hyper-heuristics achieve a success rate above 25%. These results indicate that, in at least one-quarter of the test instances, using one of these hyper-heuristics guarantees a solution which makespan will be at least as small as the one produced by H*. In some cases, as it is for HH36, this value increases to 40%. These results support the idea that it is possible to improve the search process by combining heuristics as it progresses.

To deepen the previous results and the hyper-heuristics behavior, we compared the difference between the makespan of each hyper-heuristic against the one yielded by H* in the test set. For example, consider the 12th instance, where the oracle yielded a schedule whose makespan is 848 hours. In contrast, the best hyper-heuristic from scenario C, *i.e.,* HH36, generated a schedule that only requires 811 hours—37 hours less than H*. Conversely, for the 3rd instance, the schedule obtained by H* requires 720 hours, while the one of HH36 requires 825 hours. Then, using HH36 represents 105 more hours than H*, which is an undesired behavior. However, in the same third instance, HH36 overcomes SPT, whose schedule requires 850 hours to finish. Then, we observe that these methods are choosing different heuristics as the search progresses. Hence, their robustness.

Figure 3 depicts the distribution of the savings (in hours) derived from using the three best hyper-heuristics produced in this work (the heuristics are also included for visualization purposes). Figure 3 shows the distribution of the differences in makespans. Negative values indicate that the hyper-heuristic is better than H* for that particular instance. Conversely, a positive value indicates that H* was better than the hyper-heuristic for that instance. Regardless of the remarkable results achieved by these hyper-heuristics in some particular cases, the performance of H* (11858 hours) is still far from being replicated. Even so, these hyper-heuristics represent significant savings concerning single heuristics. For example, the best two heuristics, SPT and MRT, generate schedules that require a total of 12600 and 12848 hours to complete, respectively.

TABLE I

RESULTS OF THE HEURISTICS ON THE TEST SET. THE CELLS REPRESENT THE MAKESPAN OF THE SCHEDULES PRODUCED BY EACH METHOD (IN HOURS). THE BEST RESULT PER INSTANCE AMONG THE SIX HEURISTICS IS HIGHLIGHTED IN BOLD AND CONSIDERED FOR CALCULATING H*.

INSTANCE	SPT	LPT	MRT	MLM	LLM	EST	H*
1	893	1001	827	965	**811**	892	**811**
2	958	1219	890	889	**813**	1095	**813**
3	850	824	**720**	961	813	1027	**720**
4	865	**814**	951	891	969	1009	**814**
5	832	819	910	**812**	942	1002	**812**
6	852	**812**	901	889	1013	1086	**812**
7	853	816	770	**752**	808	878	**752**
8	875	1242	896	**768**	867	978	**768**
9	**687**	775	880	754	960	1028	**687**
10	**771**	881	848	869	1031	978	**771**
11	895	985	**850**	983	1038	1091	**850**
12	871	1012	**848**	968	855	990	**848**
13	**817**	855	866	840	850	985	**817**
14	**809**	972	859	812	1051	833	**809**
15	**772**	826	832	923	906	1009	**772**
Total makespan	12600	13853	12848	13076	13727	14881	**11856**
Success rate, ρ	33.33%	13.33%	20.00%	20.00%	13.33%	0.00%	100%

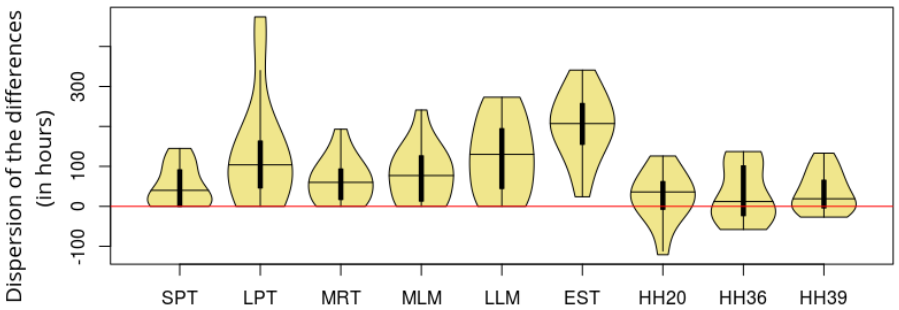

Fig. 3. Dispersion of the differences (in hours) for the six heuristics and the best hyper-heuristics from each configuration when compared against H*. Negative numbers indicate that the H* was outperformed.

TABLE II

CONDITIONS OF THE EXPERIMENTS CARRIED OUT IN THIS INVESTIGATION.

Scenario	Hyper-heuristics	Updates of performance matrix
A	HH01 to HH15	30000
B	HH16 to HH30	75000
C	HH31 to HH45	150000

Meanwhile, hyper-heuristics HH20, HH36, and HH39 require only 12219, 12313, and 12392 hours, respectively, saving 381, 287, and 208 hours in each case, concerning the best heuristic, SPT.

A. Discussion

What makes a heuristic a good solver for the instances considered in this work? Why does it seem that including features that characterize the solution space harms the learning process? We believe the answer may be found in the way features change as the instance is solved.

Fig. 4 shows how the solution changes for the 15 instances, from the lens of features. For simplicity, we only show the evolution of the best two heuristics (SPT and MRT) and the best HH of the second testing scenario (HH20), which we consider the best among the 45 HHs produced. The figure shows the evolution of the features that characterize the problem state (NJT, NAPT, and DNPT). Besides, each feature triplet contains 1500 colored points ranging from dark purple to pink, which correspond to the extrema (0% and 100%) of the solution progress.

Recall that DNPT represents the coefficient of variation, and NJT is a processing time average. They begin with values of about the unit and a quarter of the unit, respectively. For DNPT, this represents a large dispersion, as would be expected in an unsolved problem. So, throughout the solution procedure, these features tend towards zero (naturally, the solution). However, for about 60% of the process, these features remain almost unchanged. After that point, their change depends on the solver. For example, in the case of SPT, it is relatively slow, while the feature changes more rapidly when solving with HH20. It is noteworthy that not all the instances reached a DNPT of zero. Also, similar behavior is noticed for NAPT values. Nevertheless, the evidence shows no substantial dissimilarities due to the solution method.

In summary, we observed that the nature of SPT, MRT,

TABLE III
RESULTS OF THE HYPER-HEURISTICS PRODUCED FOR SCENARIO A (HH01 TO HH15) ON THE TEST SET. THE CELLS REPRESENT THE MAKESPAN OF THE SCHEDULES PRODUCED BY EACH METHOD (IN HOURS). CASES WHERE THE MAKESPAN OF THE RESULTING SCHEDULE FROM USING A HH IS SMALLER OR EQUAL THAN THE ONE PRODUCED BY H* ARE HIGHLIGHTED IN BOLD.

INSTANCE	**HH01**	**HH02**	**HH03**	**HH04**	**HH05**	**HH06**	**HH07**	**HH08**	**HH09**	**HH10**	**HH11**	**HH12**	**HH13**	**HH14**	**HH15**
1	840	1049	883	933	**802**	825	**784**	**757**	905	933	830	859	846	863	967
2	984	1226	909	838	919	945	930	914	826	982	916	896	922	912	880
3	792	889	899	1005	858	838	841	829	809	920	932	854	790	999	911
4	1031	860	966	985	1045	970	935	889	981	934	885	865	920	928	830
5	851	880	917	926	845	**786**	**800**	**803**	815	841	**812**	**752**	**796**	872	908
6	939	1010	918	1083	923	**796**	1000	1103	875	1074	**781**	930	820	1053	878
7	761	774	796	831	780	826	850	776	852	823	**752**	**746**	847	816	773
8	935	1142	969	955	1123	874	918	904	961	944	784	780	818	941	951
9	936	849	821	1042	822	797	894	901	916	896	781	724	890	923	914
10	808	860	875	**770**	818	820	920	1001	818	854	878	898	886	1036	984
11	919	1133	875	1040	965	**815**	1023	912	939	888	983	**792**	853	1125	1035
12	849	924	**841**	934	883	890	971	**795**	**803**	1025	870	1037	**782**	892	**847**
13	896	824	915	911	870	**811**	883	850	917	840	**746**	830	879	891	**812**
14	871	942	**773**	882	844	816	892	853	882	839	841	**804**	822	826	875
15	875	973	798	871	824	**738**	971	844	804	908	843	878	795	795	927
Total makespan	13287	14335	13155	14006	13321	12547	13612	13131	13103	13701	12634	12645	12666	13872	13492
Success rate, ρ	0.00%	0.00%	13.33%	6.67%	6.67%	33.33%	13.33%	20.00%	6.67%	0.00%	26.67%	26.67%	13.33%	0.00%	13.33%

TABLE IV
RESULTS OF THE HYPER-HEURISTICS PRODUCED FOR SCENARIO B (HH16 TO HH30) ON THE TEST SET. THE CELLS REPRESENT THE MAKESPAN OF THE SCHEDULES PRODUCED BY EACH METHOD (IN HOURS). CASES WHERE THE MAKESPAN OF THE RESULTING SCHEDULE FROM USING A HH IS SMALLER OR EQUAL THAN THE ONE PRODUCED BY H* ARE HIGHLIGHTED IN BOLD.

INSTANCE	**HH16**	**HH17**	**HH18**	**HH19**	**HH20**	**HH21**	**HH22**	**HH23**	**HH24**	**HH25**	**HH26**	**HH27**	**HH28**	**HH29**	**HH30**
1	824	908	933	1033	884	823	827	933	1087	881	**778**	968	904	870	967
2	886	1019	970	1087	939	1025	934	857	964	970	897	970	875	988	968
3	770	806	823	922	831	889	870	937	997	754	829	989	730	936	866
4	951	1056	941	959	865	1016	899	998	976	907	848	963	1064	941	884
5	**761**	877	926	1011	**807**	964	**812**	841	983	**778**	841	926	826	953	908
6	867	1023	1064	952	**794**	968	**784**	950	938	880	880	952	849	**810**	987
7	770	791	832	791	761	812	784	798	845	868	**746**	**727**	768	836	**723**
8	958	956	788	961	**707**	918	858	951	987	833	893	996	819	1022	860
9	917	846	867	932	723	834	808	890	965	840	794	846	743	911	783
10	887	913	821	1063	850	869	854	872	913	875	842	832	834	949	866
11	893	1022	871	1116	**729**	976	940	981	1006	956	943	**797**	**797**	1069	954
12	**842**	861	979	911	886	861	**831**	**822**	**813**	924	**836**	**759**	948	**837**	929
13	956	827	933	968	824	861	**805**	**778**	968	853	**747**	922	**795**	983	837
14	**786**	901	882	936	857	**780**	875	883	826	**809**	899	883	**799**	812	835
15	**758**	918	787	900	**762**	882	867	973	815	836	895	**722**	819	903	816
Total makespan	12826	13724	13417	14542	12219	13478	12748	13464	14083	12964	12668	13252	12570	13820	13183
Success rate, ρ	26.67%	0.00%	0.00%	0.00%	33.33%	6.67%	26.67%	13.33%	6.67%	13.33%	26.67%	26.67%	20.00%	13.33%	6.67%

and HH20 led to behaviors that resembled a beam that moves from large values of NAPT and DNPT to smaller values of NAPT involving almost no changes to NJT and DNPT. Only for NAPT values close to 0, we observe significant changes to the values of NJT and DPNT. Therefore, smooth changes in the problem features characterize a robust method for solving JSSP instances while the solution progresses. Of course, more work is needed to confirm these preliminary ideas.

V. CONCLUSION AND FUTURE WORK

Our results prove that it is indeed possible to combine heuristics into a more robust solver. Some of our generated hyper-heuristics (HHs) outperformed all the available heuristics at some test instances. It is important to stress that the best solution obtained by heuristics for a particular instance may not necessarily be optimal. Hence, a HH can improve upon it. Also, bear in mind that HHs operate by using available heuristics. It is fascinating to see how performance improves by changing heuristics throughout the solution process.

An issue that caught our attention is the effect of the characterization in hyper-heuristic performance. We consider it possible that the proposed model is sensitive to the expressive power of the features used to characterize the problem. So far, we have only considered problem-related features, but it would be interesting to observe the behavior of the approach when adding solution-related ones. We believe that analyzing other types of features may deepen our understanding of the contributions of the model and also generate some ideas for parameter selection that could improve its performance. Hence, future work should focus on testing our model under

TABLE V

RESULTS OF THE HYPER-HEURISTICS PRODUCED FOR SCENARIO C (HH31 TO HH45) ON THE TEST SET. THE CELLS REPRESENT THE MAKESPAN OF THE SCHEDULES PRODUCED BY EACH METHOD (IN HOURS). CASES WHERE THE MAKESPAN OF THE RESULTING SCHEDULE FROM USING A HH IS SMALLER OR EQUAL THAN THE ONE PRODUCED BY H* ARE HIGHLIGHTED IN BOLD.

INSTANCE	HH31	HH32	HH33	HH34	HH35	HH36	HH37	HH38	HH39	HH40	HH41	HH42	HH43	HH44	HH45
1	844	983	**778**	**809**	**809**	823	814	962	**804**	832	**808**	835	968	818	926
2	943	938	897	845	845	945	1020	988	862	959	971	952	970	938	920
3	877	999	829	793	793	825	839	916	853	874	840	960	989	1008	957
4	870	966	848	846	846	951	911	929	819	924	950	905	963	954	963
5	966	895	841	870	870	**793**	874	**798**	**809**	911	850	841	926	901	898
6	919	1034	880	877	877	818	1017	1118	870	990	870	1002	952	1057	875
7	**729**	769	**746**	**744**	**744**	**704**	**724**	865	825	823	**751**	**708**	**727**	804	**738**
8	949	977	893	834	834	787	962	918	**748**	930	**730**	912	996	890	**731**
9	980	851	794	863	863	785	934	999	791	936	725	874	846	934	747
10	923	962	842	837	837	895	934	838	818	863	792	844	832	943	911
11	985	934	943	1005	1005	**831**	1045	898	869	1002	**738**	898	**797**	969	896
12	903	922	**836**	858	858	**811**	860	871	951	**806**	934	**847**	**759**	**832**	953
13	883	912	**747**	835	835	**789**	835	1063	824	919	830	910	922	883	854
14	887	912	899	820	820	**751**	842	857	**782**	860	852	868	883	863	880
15	869	846	895	781	781	805	834	773	**767**	1023	**769**	846	**722**	942	880
Total makespan	13527	13900	12668	12617	12617	12313	13445	13793	12392	13652	12410	13202	13252	13736	13129
Success rate, ρ	6.67%	0.00%	26.67%	13.33%	13.33%	40.00%	6.67%	6.67%	33.33%	6.67%	33.33%	13.33%	26.67%	6.67%	13.33%

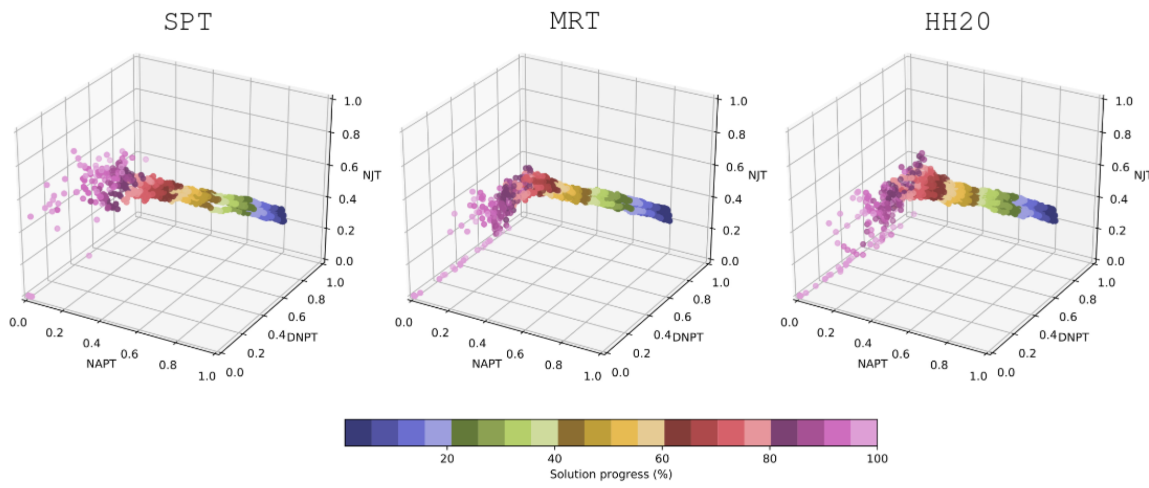

Fig. 4. Feature variation while solving 15 JSSP instances of size 10 $\times$ 10 with the SPT and MRT heuristics, as well as with the best hyper-heuristic found in the second testing scenario (HH20). The color maps the solution progress, ranging from dark purple (0%) to pink (100%).

different feature scenarios.

Finally, we are aware that we have not tested the scalability of the approach appropriately. In this work, we have used a minimal set of small synthetic JSSP instances. However, we need to consider a larger number of more challenging and realistic instances to evaluate the capability of the model for adapting to other types of instances. Mainly, we are aware that in the real world context, static scheduling is usually unrealistic. Unforeseen events, like new job arrivals, machine breakdowns, or changes to due dates, are commonly faced in industrial settings. Such situations require updating an existing schedule based on changes. These new testing scenarios could yield a clearer idea about the scalability of our approach and its real usefulness.

REFERENCES

[1] H. Öztop, M. F. Tasgetiren, D. T. Eliiyi, and Q.-K. Pan, "Metaheuristic algorithms for the hybrid flowshop scheduling problem," *Computers & Operations Research*, vol. 111, pp. 177–196, 2019.

[2] J. V. Moccellin, M. S. Nagano, A. R. P. Neto, and B. de Athayde Prata, "Heuristic algorithms for scheduling hybrid flow shops with machine blocking and setup times," *Journal of the Brazilian Society of Mechanical Sciences and Engineering*, vol. 40, no. 2, p. 40, 2018.

[3] M. Nouiri, A. Bekrar, A. Jemai, S. Niar, and A. C. Ammari, "An effective and distributed particle swarm optimization algorithm for flexible job-shop scheduling problem," *Journal of Intelligent Manufacturing*, vol. 29, no. 3, pp. 603–615, 2018.

[4] J. Xie, L. Gao, K. Peng, X. Li, and H. Li, "Review on flexible job shop scheduling," *IET Collaborative Intelligent Manufacturing*, vol. 1, no. 3, pp. 67–77, 2019.

[5] D. Applegate and W. Cook, "A computational study of the job-shop scheduling problem," *ORSA Journal on computing*, vol. 3, no. 2, pp. 149–156, 1991.

[6] M. R. Garey, D. S. Johnson, and R. Sethi, "The complexity of flowshop

and jobshop scheduling," *Mathematics of operations research*, vol. 1, no. 2, pp. 117–129, 1976.

[7] A. Muluk, H. Akpolat, and J. Xu, "Scheduling problems—an overview," *Journal of Systems Science and Systems Engineering*, vol. 12, no. 4, pp. 481–492, 2003.

[8] M. A. Cruz-Chávez, M. G. Martínez-Rangel, and M. H. Cruz-Rosales, "Accelerated simulated annealing algorithm applied to the flexible job shop scheduling problem," *International Transactions in Operational Research*, vol. 24, no. 5, pp. 1119–1137, 2017.

[9] M. Kurdi, "An effective new island model genetic algorithm for job shop scheduling problem," *Computers & Operations Research*, vol. 67, pp. 132–142, 2016.

[10] L. Shen, S. Dauzère-Pérès, and J. S. Neufeld, "Solving the flexible job shop scheduling problem with sequence-dependent setup times," *European Journal of Operational Research*, vol. 265, no. 2, pp. 503–516, 2018.

[11] J. Zhang, G. Ding, Y. Zou, S. Qin, and J. Fu, "Review of job shop scheduling research and its new perspectives under industry 4.0," *Journal of Intelligent Manufacturing*, vol. 30, no. 4, pp. 1809–1830, 2019.

[12] J. H. Blackstone, D. T. Phillips, and G. L. Hogg, "A state-of-the-art survey of dispatching rules for manufacturing job shop operations," *International Journal of Production Research*, vol. 20, no. 1, pp. 27–45, 1982.

[13] J. Adams, E. Balas, and D. Zawack, "The shifting bottleneck procedure for job shop scheduling," *Management Science*, vol. 34, no. 3, pp. 391–401, 1988.

[14] L. Hernández-Ramírez, J. Frausto Solís, G. Castilla-Valdez, J. J. González-Barbosa, D. Terán-Villanueva, and M. L. Morales-Rodríguez, "A hybrid simulated annealing for job shop scheduling problem," *International Journal of Combinatorial Optimization Problems and Informatics*, vol. 10, no. 1, pp. 6–15, 2018.

[15] P. J. M. van Laarhoven, E. H. L. Aarts, and J. K. Lenstra, "Job shop scheduling by simulated annealing," *Operations Research*, vol. 40, no. 1, pp. 113–125, 1992.

[16] T. Satake, K. Morikawa, K. Takahashi, and N. Nakamura, "Simulated annealing approach for minimizing the makespan of the general job-shop," *International Journal of Production Economics*, vol. 60-61, pp. 515–522, 1999.

[17] W. Bozejko, A. Gnatowski, J. Pempera, and M. Wodecki, "Parallel tabu search for the cyclic job shop scheduling problem," *Computers & Industrial Engineering*, vol. 113, pp. 512–524, 2017.

[18] C. Zhang, P. Li, Z. Guan, and Y. Rao, "A tabu search algorithm with a new neighborhood structure for the job shop scheduling problem," *Computers & Operations Research*, vol. 34, no. 11, pp. 3229–3242, 2007.

[19] N. Bhatt and N. R. Chauhan, "Genetic algorithm applications on job shop scheduling problem: A review," in *International Conference on Soft Computing Techniques and Implementations (ICSCTI)*, (Faridabad, India), pp. 7–14, IEEE, 2015.

[20] S. Hou, Y. Liu, H. Wen, and Y. Chen, "A self-crossover genetic algorithm for job shop scheduling problem," in *IEEE International Conference on Industrial Engineering and Engineering Management*, (Singapore, Singapore), pp. 549–554, IEEE, 2011.

[21] M. A. Elaziz and S. Mirjalili, "A hyper-heuristic for improving the initial population of whale optimization algorithm," *Knowledge-Based Systems*, vol. 172, pp. 42–63, 2019.

[22] F. Caraffini, F. Neri, and M. Epitropakis, "Hyperspam: A study on hyper-heuristic coordination strategies in the continuous domain," *Information Sciences*, vol. 477, pp. 186–202, 2019.

[23] I. Amaya, J. C. Ortiz-Bayliss, S. Conant-Pablos, and H. Terashima-Marín, "Hyper-heuristics reversed: Learning to combine solvers by evolving instances," in *2019 IEEE Congress on Evolutionary Computation (CEC)*, (Wellington, New Zealand, New Zealand), pp. 1790–1797, IEEE, IEEE, 2019.

[24] S. L. Epstein, E. C. Freuder, R. Wallace, A. Morozov, and B. Samuels, "The adaptive constraint engine," in *Proceedings of the 8th International Conference on Principles and Practice of Constraint Programming*, CP '02, (London, UK, UK), pp. 525–542, Springer-Verlag, 2002.

[25] S. Petrovic and R. Qu, "Case-based reasoning as a heuristic selector in a hyper-heuristic for course timetabling problems," in *Knowledge-Based Intelligent Information Engineering Systems and Allied Technologies*, pp. 336–340, IOS Press, 2002.

[26] E. O'Mahony, E. Hebrard, A. Holland, C. Nugent, and B. O'Sullivan, "Using case-based reasoning in an algorithm portfolio for constraint solving," in *Irish conference on artificial intelligence and cognitive science*, pp. 210–216, 2008.

[27] J. C. Ortiz-Bayliss, H. Terashima-Marín, and S. E. Conant-Pablos, "Combine and conquer: an evolutionary hyper-heuristic approach for solving constraint satisfaction problems," *Artificial Intelligence Review*, vol. 46, no. 3, pp. 327–349, 2016.

[28] K. Sim, E. Hart, and B. Paechter, "A lifelong learning hyper-heuristic method for bin packing," *Evol. Comput.*, vol. 23, pp. 37–67, Mar. 2015.

[29] N. Pillay and R. Qu, *Hyper-Heuristics: Theory and Applications*. Natural Computing Series, Cham: Springer International Publishing, 2018.

[30] Y. Malitsky, "Evolving instance-specific algorithm configuration," in *Instance-Specific Algorithm Configuration*, pp. 93–105, Cham: Springer International Publishing, 2014.

[31] J. Shahrabi, M. A. Adibi, and M. Mahootchi, "A reinforcement learning approach to parameter estimation in dynamic job shop scheduling," *Computers & Industrial Engineering*, vol. 110, pp. 75–82, 2017.

[32] M. Zhao, X. Li, L. Gao, L. Wang, and M. Xiao, "An improved q-learning based rescheduling method for flexible job-shops with machine failures," in *2019 IEEE 15th International Conference on Automation Science and Engineering (CASE)*, (Vancouver, BC, Canada, Canada), pp. 331–337, IEEE, Aug 2019.

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

[34] F. Garza-Santisteban, R. Sánchez-Pámanes, L. A. Puente-Rodríguez, I. Amaya, J. C. Ortiz-Bayliss, S. Conant-Pablos, and H. Terashima-Marín, "A simulated annealing hyper-heuristic for job shop scheduling problems," in *2019 IEEE Congress on Evolutionary Computation (CEC)*, (Wellington, New Zealand, New Zealand), pp. 57–64, IEEE, IEEE, 2019.

[35] E. Lara-Cárdenas, X. Sánchez-Díaz, I. Amaya, and J. C. Ortiz-Bayliss, "Improving hyper-heuristic performance for job shop scheduling problems using neural networks," in *Advances in Soft Computing* (L. Martínez-Villaseñor, I. Batyrshin, and A. Marín-Hernández, eds.), (Cham), pp. 150–161, Springer International Publishing, 2019.

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

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

[38] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of artificial intelligence research*, vol. 4, pp. 237–285, 1996.

[39] D. Falcão, A. Madureira, and I. Pereira, "Q-learning based hyper-heuristic for scheduling system self-parameterization," in *2015 10th Iberian Conference on Information Systems and Technologies (CISTI)*, (Aveiro, Portugal), pp. 1–7, IEEE, IEEE, 2015.

[40] Y.-F. Wang, "Adaptive job shop scheduling strategy based on weighted q-learning algorithm," *Journal of Intelligent Manufacturing*, pp. 1–16, 2018.

[41] Y.-R. Shiue, K.-C. Lee, and C.-T. Su, "Real-time scheduling for a smart factory using a reinforcement learning approach," *Computers & Industrial Engineering*, vol. 125, pp. 604–614, 2018.

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

[43] P. Bratley, B. L. Fox, and L. E. Schrage, *A guide to simulation*. Springer Science & Business Media, 2011.