A Simulated Annealing Hyper-heuristic for Job Shop Scheduling Problems

Fernando Garza-Santisteban*, Roberto Sánchez-Pámanes*, Luis Antonio Puente-Rodríguez*,
Ivan Amaya*, José Carlos Ortiz-Bayliss*, Santiago Conant-Pablos* and Hugo Terashima-Marín*
*Tecnologico de Monterrey
National School of Engineering and Science
Email: {a00918788, a01139935, a00809683}@itesm.mx, {iamaya2, jcobayliss, sconant, terashima}@tec.mx

***Abstract*—Job Shop Scheduling problems (JSSPs) have become increasingly popular due to their application in supply chain systems. Several solution approaches have appeared in the literature. One of them is the use of low-level heuristics. These methods approximate a solution but only work well on some kind of problems. Hence, combining them may improve performance. In this paper, we use the classical stochastic local optimization algorithm Simulated Annealing to train a selection hyper-heuristic for solving JSSPs. To do so, we use an instance generator provided in literature to create training sets with a different number of instances: 20, 40, and 60. In addition, we select instances from the literature to create two test scenarios, one similar to the training instances, and another with bigger problems. Our results suggest that training with the highest number of instances lead to better and more stable hyper-heuristics. For example, in the first test scenario, we achieved a reduction in the data range of over 60% and an improvement in the median performance of almost 30%. Moreover, under these conditions about 75% of the generated hyper-heuristics were able to perform equal to or better than the best heuristic. Even so, less than 25% were able to outperform the synthetic Oracle. Because of the aforementioned, we strongly support the idea of using a selection hyper-heuristic model powered by Simulated Annealing for creating a high-level solver for Job Shop Scheduling problems.**

***Index Terms*—Job Shop Scheduling, hyper-heuristic, Simulated Annealing.**

I. Introduction

Scheduling systems have been increasingly popular in supply chain systems [1], [2]. The performance of many manufacturing processes greatly depends on how optimal an schedule is. The Job Shop Scheduling (JSS) problem requires scheduling a set of jobs on a set of machines, subject to the constraint that each machine can handle at most one job at a time, and to the fact that each job has a specified processing order throughout the machines. The objective is to schedule the jobs in a way that minimizes the amount of time required to complete all jobs.

JSS problems become quite difficult to handle in real-world applications. Moreover, JSS problems belong to the non-deterministic polynomial time (*NP-hard*) problem class. This relation places JSS problems into the class of those that are not solvable in polynomial time (called *P* problems) unless $P = NP$. In fact, there are $(n!)^m$ possibilities to schedule n number of jobs and m number of machines on a given shop floor [3]. Thus, while two jobs can only be arranged in one machine in two possible ways, eight jobs can be scheduled in over 40,000 different ways.

Many solution methods to JSS problems have been proposed throughout the history of operations research. Kurdi [4] provides an overview of some methods used for solving the JSS that include an enumerative approach implemented by Lageweg, Lenstra and Rinooy [5], and the branch and bound method used by Carlier and Pinson [6]. These methods worked very well for small instances. However, when increasing the instance size their computational cost increased very fast. Metaheuristics have also been used for solving the JSS problem. For example, SA algorithms are used in the works by Satake, Morikawa, Takahashi and Nakamura [7], and by Laarhoven, Aarts and Lenstra [8]. Traditional tabu search is present in the research made by Nowicki and Smutnicki [9] and Zhang, Li, Guan and Rao [10]. Recent improvements include the method proposed by Peng, Lü and Cheng [11], which use a Tabu Search/Path Relinking (TS/PR) technique that improves the upper bounds for 49 of the 205 instances in which it was tested.

Nevertheless, the inherent complexity of the JSS problem has shifted some research efforts towards heuristic approaches. Examples of this approach were shown early on by the dispatching rules proposed by Blackstone, Philips and Hogg [12]. Another early example includes the *Shifting Bottleneck* procedure, developed by Adams, Balas and Zawack [13]. These methods although computationally efficient, provide solutions for which quality is not guaranteed.

The use of heuristics is a common approach whenever exhaustive search becomes unfeasible. This technique is used to generate an approximate solution in less time and with less computing resources. Since their efficiency can be improved when used in conjunction with other optimization algorithms, it is common to see them applied to *NP* problems.

Heuristics are usually designed to tackle specific problem instances. Consequently, they can be very sensitive to problem features. A heuristic may exhibit good or optimal performance over some problem instances but poor over others. However, if instead of using a single heuristic for all instances, we select

This project was funded by Consejo Nacional de Ciencia y Tecnología (CONACyT) through the Basic Science Projects with grant numbers 241461 and 287479, and by the Research Group with Strategic Focus in Intelligent Systems at Tecnologico de Monterrey.

the best one for each problem, the overall performance of a solver may be improved.

Despite the several variations of this approach, all of them can be covered by stating them as an algorithm selection problem. One of such approaches is to switch heuristics not only at the beginning of the problem but also throughout its solution, which is commonly referred to as a selection hyper-heuristic (HH). This solution is not the only of its kind, there are other hyper-heuristic models that involve combining or generating several simple heuristics during the search. A selection HH can use the most suitable heuristic for each step of the search and have the ability to be applicable to a range of different instances and problems [14]. One recent example of this technique is the work of Vazquez-Rodriguez et al. [15], where the authors used a Genetic Algorithm to generate sequences of dispatching rules for the JSS problem. Several hyper-heuristic models in the literature use a stochastic process to build a good solution strategy instead of finding the solutions directly (please refer to Section II-B).

Even though hyper-heuristics have been around for some years, and in spite of all the work hovering around JSS problems, evidence of their synergy is scarce. Thus, in this paper, we propose to fill that knowledge gap by presenting a Simulated Annealing based selection hyper-heuristic for solving JSS problems. The proposed method is unique because while there are cases [9], [16], [17] where search or evolutionary methods are used to find approximate schedules, in this research we use low-level heuristics. Such heuristics are either the result of empirical findings or intuitive rules-of-thumb. Also, we include a search method to find the best combination of heuristics for each problem state, thus operating on the solver space instead of the solution space.

This paper is organized as follows: Section II provides relevant background information about the problem and the main concepts used in this work; Section III explains the inner workings of the selection hyper-heuristic model, and includes information about the features, heuristics, and instances considered in this work. Further on, Section IV presents a summary of the testing carried out in this work and Section V contains the experimental data we gathered. Finally, Section VI summarizes the main findings of our work.

II. Preliminaries

A. Problem formulation

A JSS problem can be formally stated as follows: Given a set of jobs $J = \{1, ..., n\}$ and a set of machines $M = \{1, ..., m\}$, for each $j \in J$ there exists a permutation $\sigma_j = (\sigma_j^1, \ldots, \sigma_j^m)$ of machines, which specifies the order in which j must be processed through them. Problem instances are described with the notation $m \times n$. Also, associated to each job j and machine i there is a nonnegative integer $p_{j,i}$ representing the processing time of job j on machine i. The completion time of job j on machine i is denoted by $C_{i,j}$. The time in which job j exits the system, is denoted by C_j. Furthermore, the make-span (C_{max}) is defined as the maximum time required for completing a job j, i.e. $C_{max} = \max(C_1, \ldots, C_j)$ [18]. Our objective in this work is to minimize such a make-span.

B. Background and related work

The state-of-the-art regarding JSS problems and optimization is too vast to summarize here. Nonetheless, we want to highlight some recent works that we deem relevant.

Zhao et al. [1] proposed a modification of Shuffled Complex Evolution (SCE), called Improved SCE (ISCE). The authors showed that ISCE outperforms SCE algorithm in terms of solution quality and convergence rate. However, the performance of ISCE on large-scale JSS problems is not as stable as it is in smaller instances. Peng et al. [11] used a Tabu Search/Path Relinking (TS/PR) approach to improve the upper bounds on 49 out of 205 instances. Other works include the one from Wang et al. [19], where the authors provided an adaptive multi-population genetic algorithm; Kurdi, however, used an Island Model Genetic Algorithm (IMGE) [4]. Gao et al. [20], conversely, proposed using a hybrid Particle Swarm Optimization (PSO) algorithm based on Variable Neighborhood Search (VNS); Cheng et al. [21] integrate a TS into the framework of an Evolutionary Algorithm (EA), yielding what they call a Hybrid Evolutionary Algorithm (HEA).

Kurdi recently presented an overview of the methods historically used for tackling JSS problems [4]. Among them reside an enumerative approach implemented by Lageweg et al. [5], and a *branch and bound* method used by Carlier and Pinson [6]. These methods worked very well for small instances. However, the computational cost increased too fast when increasing the instance size.

These problems caused research efforts to shift towards heuristic approaches. Examples of such transitions are evidenced in the dispatching rules proposed by Blackstone et al. [12], and in the *Shifting Bottleneck* procedure by Adams et al. [13]. These methods are computationally efficient but the quality of their solutions is not guaranteed.

Metaheuristics have also been used for solving the JSS problem. For example, Simulated Annealing (SA) was used in the works by Satake et al. [7], and by Laarhoven et al. [8]. Similarly, Taboo Search (TS) was also used for this endeavor [9], [10]. Particle Swarm Optimization (PSO) was implemented by Lian et al. [22], and by Lin et al. [23]. Another example is the use of Ant Colony Optimization (ACO), as exemplified by Blum and Sampels [24], and by Seo and Kim [25].

III. Our Proposed Approach

A. An overview of the hyper-heuristic solution model

We modeled the hyper-heuristic as an array of n_b blocks. Each block consists of a series of features (see Section III-E), each of them represented by a two-digit decimal, and an action. Figure 1 shows an example of the model. Note that features are ordered, so the first value of each block always point to the Accumulated Processed Time (APT) feature (in this example). The way in which the hyper-heuristic proposes the best action given a state of the problem is as follows:

1) Map the current state of the JSS problem to the set of features $(f_1, f_2, ..., f_f)$
2) Compare the current state against the hyper-heuristic with n_b blocks, each of them with their own feature values and actions. This is done by calculating the Euclidean distance between the current state of the problem and each block of the hyper-heuristic.
3) Select the block that minimizes this distance and identify its action (i.e. a heuristic).

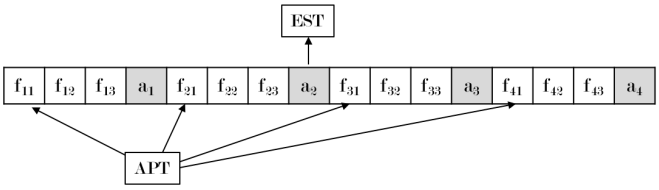

Fig. 1. Representation of a hyper-heuristic (HH) with $n_b = 4$ blocks. In this example, the second block maps to action EST (a_2), and features f_{X1} represent feature APT. Moreover, features f_{1X} represent the features from block 1.

The hyper-heuristic will use the identified action on the problem, thus obtaining an activity to be added to the JSS schedule. Next, the problem state changes and the process must be repeated until all activities are scheduled and a make-span, $C_{s_i}^{HH}$, can be computed.

It is worth mentioning that the model is not limited to a specific number of blocks, nor to having one block per heuristic. The same heuristic could be used in different blocks to allow for more complex behavior. Also, all blocks must always contain the same number of features and in the same order.

B. Training and testing the hyper-heuristic model

Since the objective of the hyper-heuristic is to outperform the low-level heuristics, we implemented an objective function (ff) given by ΔC_O, which relates the performance of the hyper-heuristic to that of the best single heuristic (i.e. the Oracle). This relation is calculated and accumulated over all instances. Equation (1) shows such an expression, where $C_{s_i}^{HH}$ is the make-span achieved by the hyper-heuristic on instance i, and C_{b_i} is the best make-span achieved by the heuristics on instance i. Additionally, n is the number of instances that each candidate hyper-heuristic tries to solve. Therefore, the only difference between assesing the training and testing performance of a hyper-heuristic relies on the set of instances being used.

$$ff(HH) = \Delta C_O = \frac{\sum_{i=1}^{n} \frac{C_{s_i}^{HH} - C_{b_i}}{C_{b_i}}}{n}, \qquad (1)$$

C. The Simulated Annealing (SA) algorithm

Simulated Annealing was originally presented by Kirkpatrick et al. [26] in 1983. SA represents one of the simplest and most general techniques for stochastic search and has turned out to be among the most efficient ones. The algorithm seeks to mimic the crystallization process during cooling or annealing: when a material is hot, particles have high kinetic energy and move more or less randomly. The cooler the material gets, the more particles tend to move towards the direction that minimizes the energy balance. The SA algorithm does the same when searching for optimal values for a set of decision parameters: it repeatedly suggests random modifications to the current solution, but progressively keeps only those that improve the current one. SA applies a probabilistic rule to decide whether the new solution replaces the current one or not. This rule considers the change in the objective function (measuring the improvement/impairment) and a parameter akin to "temperature" (reflecting the progress in the iterations). Dueck and Scheuer [27] produced a variation of the algorithm by suggesting a deterministic acceptance rule instead which makes the algorithm even simpler: accept any random modification unless the resulting impairment exceeds a certain threshold; this threshold is lowered over the iterations. This variation is known as Threshold Accepting (TA).

In this work, we used SA for optimizing Eq. (1), thus training the hyper-heuristic model. To do so, we adjusted the SA implementation to work with hyper-heuristics on the JSS problem. The pseudocode is shown in Algorithm 1. The output of the algorithm is the hyper-heuristic that gets the best fitness function in the search process.

Algorithm 1 Simulated Annealing pseudocode.

```
Set initial temperature t to a defined value
Set cooling rate c to a defined value
Generate an initial hyper-heuristic HH
Set the best hyper-heuristic HH* = HH
while The system is not cool, meaning t > threshold do
    Create a neighbor hyper-heuristic HH_n by randomly modifying HH
    Compute fitness function for HH and HH_n, using Eq. (1)
    if Acceptance probability (HH, HH_n) > random number then
        HH ← HH_n
    end if
    if fitness of HH < fitness of HH* then
        HH* = HH
    end if
    Update temperature t = t(1 − c)
end while
return HH*
```

Training requires the generation of a neighboring hyper-heuristic (HH_n) based on the current hyper-heuristic (HH). We considered the following three possible ways to generate such a neighbor (Fig. 2):

1) Remove block: The size of the HH is decreased by 1 through the elimination of a random block. This option is omitted if the current HH has only one block.
2) Add new block: The size of the HH is increased by 1 block, with random features and actions. Other actions and feature values are preserved.
3) Randomize blocks: The current values for each feature are randomized. Actions remain the same.

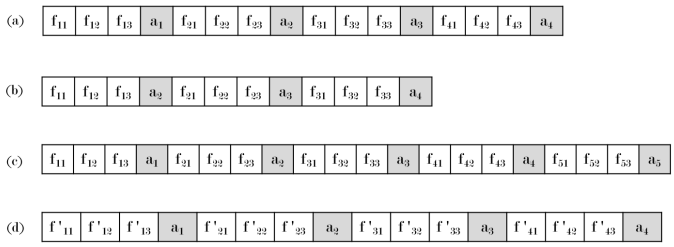

Fig. 2. Example of possible neighbors of a candidate solution (a), generated with each approach: (b) Remove block, (c) Add new block, (d) Randomize blocks.

D. Problem instances used in this work

In this work we consider reported and generated instances. The former are used for testing and correspond to those generated and published in the work of Taillard [28]. The latter are constructed using the same generator, aiming to create instances with a similar nature that can be used for training. To achieve so, we followed the same method described by Taillard [28] for generating the random scheduling problems of each training set. The main characteristic of this generator is that it produces a random distribution of numbers for the machines and times of each job. Algorithms 2 and 3 describe the pseudocode of the instance generator and the random number generator, respectively. In each experiment, half of the instances in the training set are of size 15×15, and the other half of size 20×15. The training and both test sets are mutually exclusive, i.e. all instances are different.

As mentioned, we use two test sets taken from [28] for assessing the performance of hyper-heuristics. The author proposed 260 randomly generated scheduling problems whose size was greater than that of the examples published up to that date. Such sizes correspond to real dimensions of industrial problems. The instances we considered have the following characteristics, chosen for comparison purposes: fixed processing times, no set-up times, and no due dates nor release dates.

E. Problem characterization (Features)

Our hyper-heuristic model requires to map the current state of the problem to a set of features. But, we have to keep in mind that two parts can be distinguished within a JSS problem:

1) Activities to be scheduled: Set of remaining activities.
2) Current schedule: Set of already scheduled activities.

Hence, the current state of the problem can be intuitively seen as the union of both parts. We used the following set of features, which consider both parts:

1) Accumulated Processed Time (APT): Ratio between the remaining required processing time and the processing times of the sub-jobs that have already been scheduled. This feature gives a rough idea of how advanced is the state of the process.
2) Slack Percentage (SLACKPERC): Ratio between the amount of unused machine time (slack) in the whole schedule and the current make-span of the schedule. The more slack, the less compact activities are. However, there is more space that could be used for smaller activities.
3) Completed Jobs (PC_JOBS): Ratio between the total number of jobs and the number of already completed jobs at a given state of the schedule.
4) Accumulated Processing Time of pending activities (ANPT): Ratio between the time that has been already processed and time to be processed. This feature is the complement of APT.
5) Activity Distribution over Machines (MAD): Mean of the number of activities that are distributed among the machines. Gives insight on how loaded is each machine in comparison to the others.

Algorithm 2 Taillard's instance generator.

Let m be the number of machines and n the number of jobs.
$T \leftarrow n \times m$ array which represents the processing times of the j_{th} operation of job i.
$M \leftarrow n \times m$ array which represents the machine in which an activity i, j it to be processed.
$t_{seed} \leftarrow$ seed for times.
$m_{seed} \leftarrow$ seed for machines.
for $i = 1$ to n **do**
  for $j = 1$ to m **do**
    $t_{ij} \leftarrow floor(99 * \text{UNIF}((t_{seed}))$
  end for
end for
for $i = 1$ to n **do**
  for $j = 1$ to m **do**
    $ma_{ij} \leftarrow j$
  end for
end for
for $i = 1$ to n **do**
  for $j = 1$ to m **do**
    Swap $ma[i, j]$ and $ma[i, j + floor((m - j + 1) * \text{UNIF}(m_{seed}))]$
  end for
end for

Algorithm 3 Uniform random number generator.

function UNIF(x)
  $a \leftarrow 16807, b \leftarrow 127773, c \leftarrow 2836, m \leftarrow 2^{31} - 1$
  $kl \leftarrow x/b$
  $x \leftarrow a * (x \bmod b) - kl * c$
  if $x < 0$ **then**
    $x \leftarrow x + m$
  end if
  return x/m
end function

F. Low-level heuristics

For the purpose of this research, we focused our attention on implementing only constructive heuristics that determine the next activity to be scheduled and the best time for doing so. Also, we limited our heuristics so that they only produce feasible solutions to the JSS problem. Hence, no heuristic proposes an (activity, time) pair whose scheduling conflicts with others. To facilitate heuristic description, it is convenient to define U_a as the list of activities to be scheduled, and $S_i = (a_{j,i}, t_{a_j})$ as a list of tuples, where i stands for the machine number, $a_{j,i}$ is an activity of job j that goes to

machine i, and t_{a_j} is the time at which activity a_j is being scheduled. The heuristics can be summarized as:

1) Default (DEFAULT): The list of activities to be scheduled is generated job by job, and machine per machine in a natural order. The heuristic just selects the next available activity on this queue and returns said activity into its first available time.
2) Most Loaded Machine (MLMACH): Finds the machine i in U_a with the maximum total processing time. The heuristic returns the activity a_j that has the lowest possible t_{a_j} if scheduled in machine i. If no activity is possible, discard machine i and repeat the process until a suitable activity is found. An example is shown in Figure 3.
3) Maximum Remaining Time (MRTIME): Select the job that needs the most time for completion. The heuristic returns the first possible activity (in precedence order) that corresponds to said job in the first available time. An example of MRTIME is shown in Figure 4.
4) Earliest Start Time (EST): Scan U_a and get the activities that can be scheduled at the given state (possible activities). Select the activity from the job with the earliest possible starting time. An example of this heuristic is shown in Figure 5.

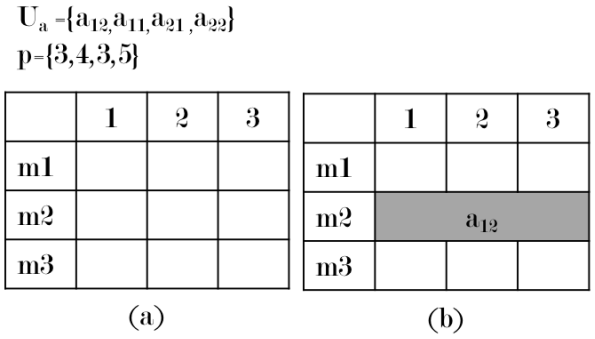

Fig. 3. MLMACH heuristic example. U_a is the list of activities to be scheduled. The current state is shown in (a). We assume that job 1 has a machine precedence order of $2, 1, 3$ and p corresponds to the specific processing times for each activity in U_a. In this case, machine 2 will be the most loaded one. The heuristic will return activity a_{12} at time 1. The state of the schedule after applying this action is shown in (b).

G. Definition of the Oracle

For this work, we considered a synthetic Oracle as a reference point for comparing the performance of heuristics and hyper-heuristics. The performance of such Oracle for a given instance corresponds to the best performance yielded by the heuristics considered in this work for the same instance. Table I shows a simple example with four heuristics (H1, H2, H3, H4) and two instances. In the first instance, H4 is the best heuristic so the Oracle yields a solution of 80. In the second one, the best heuristic is H3, so the Oracle now yields 100. When analyzed over all instances, the Oracle performs better than any single heuristic and represents the lower bound

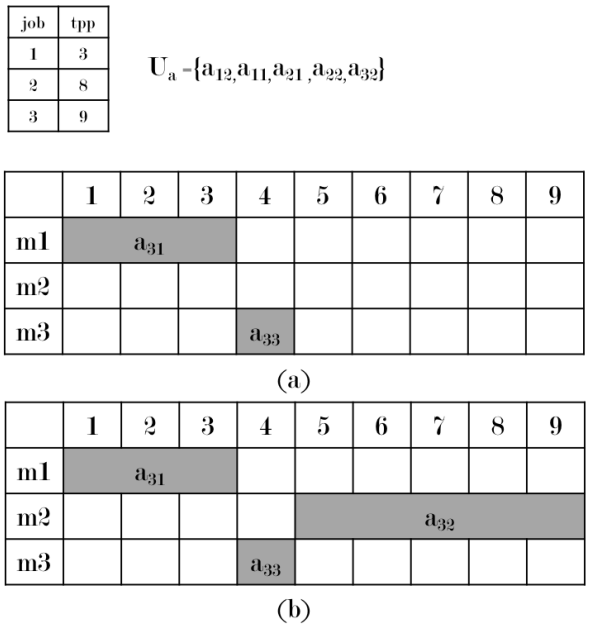

Fig. 4. MRTIME heuristic example. U_a is the list of activities to be scheduled. The current state is shown in (a). We assume that job 3 has a machine precedence order of $1, 3, 2$ and tpp corresponds to the total processing times for each job. In this case, job 3 is the one with maximum processing time. The heuristic will return activity a_{32} at time 5 because of precedence restrictions. The state of the schedule after applying this action is shown in (b).

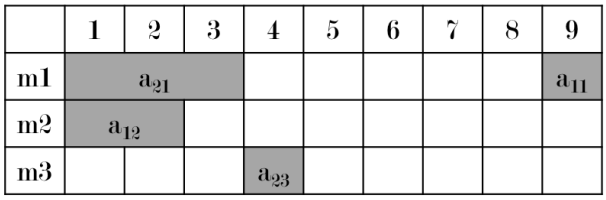

Fig. 5. EST heuristic example. Assume that the precedence restriction for job 1 is $2, 1, 3$ and for job 2 is $1, 3, 2$. Also consider that in U_a we have activities a_{13} and a_{22}, each of them with equal processing times. Then, EST will choose to schedule activity a_{22} because it can start at time 5, while the other one has to start after time 9.

achievable by heuristics. Thus, the reason for selecting it as a reference point.

IV. Methodology

We designed three experiments to test whether our hyper-heuristic model solves JSS problems better than standalone

TABLE I
EXAMPLE OF ORACLE CALCULATION, ASSUMING THAT LOWER VALUES ARE BETTER, AND CONSIDERING FOUR HEURISTICS (H1, H2, H3, H4).

Instance	H1	H2	H3	H4	Oracle
1	100	150	200	80	80 (H4)
2	120	140	100	200	100 (H3)
Total	220	290	300	280	180

TABLE II
EXPERIMENTS FOR BENCHMARKING THE PERFORMANCE OF HYPER-HEURISTICS. EACH SET OF INSTANCES IS DEFINED AS: *# of instances in the set - jobs×machines*

Experiment ID	Training set
T20	$10-15\times15$ $10-20\times15$
T40	$20-15\times15$ $20-20\times15$
T60	$30-15\times15$ $30-20\times15$

low-level heuristics. Each experiment uses a different number of training instances: 20, 40, and 60. Table II details the number and size of the instances used for training. Moreover, we defined two test sets for assessing the quality of hyper-heuristics with 10 instances each. The first test set is composed of problems with the same instance size as the training set whilst the second corresponds to bigger problems than those that the hyper-heuristic has seen. In this way, the first set contains 5 problems of size 15×15 and 5 problems of size 20×15, and strives to evaluate the performance of hyper-heuristics on problems with similar complexity. The second set contains 5 instances of size 20×15 and 5 instances of size 30×20, seeking to assess the generalization capabilities of the hyper-heuristic model. Changing the size of the problem suggests that the solution space should have a different complexity than that of the problems seen during training. Since problems are bigger, one should expect them to be more difficult to solve. Because of the stochastic nature of SA, all experiments were repeated 30 times under the same conditions.

V. EXPERIMENTS AND RESULTS

Training revealed an interesting behavior that is illustrated in Figure 6. Increasing the training set size from 20 to 40 actually seemed to worsen the stability of hyper-heuristics, since some of them yielded higher gaps between their performance and that of the Oracle. Nonetheless, the median performance in both cases was virtually the same. Furthermore, increasing the training set to 60 proved fruitful, since hyper-heuristics yielded a more stable performance. There were no outliers, and the worst-case scenario was quite close to the Oracle (see Section III-G), exhibiting a performance gap of about 0.03. The hyper-heuristic with the median performance (0.008) reduced the performance gap of the best heuristic in about 64%. It is worth remarking that for all experiments there were some hyper-heuristics that outperformed the Oracle, proving that changing heuristics midway can lead to better results.

Testing over problems with the same nature seems feasible (Figure 7). When using few instances (T20), the median performance of hyper-heuristics proved better than the best heuristic, with a performance delta below 0.015. Moreover, the best-case scenario outperformed the Oracle by a similar value (0.014). However, the worst-case scenario (an outlier)

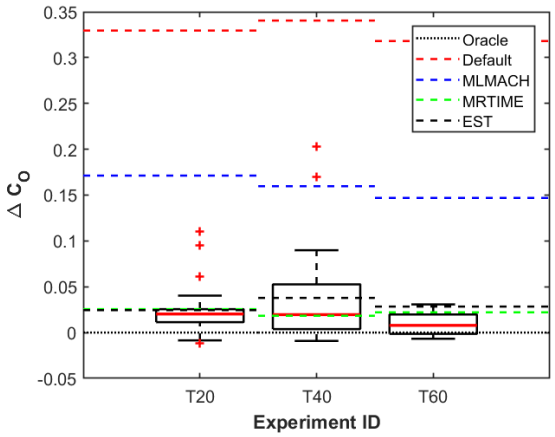

Fig. 6. Performance of hyper-heuristics trained with different number of instances, over the training set. Each box shows data of 30 repetitions. Horizontal lines are used to show the performance of each heuristic over the training sets. ΔC_O is defined in Eq. (1).

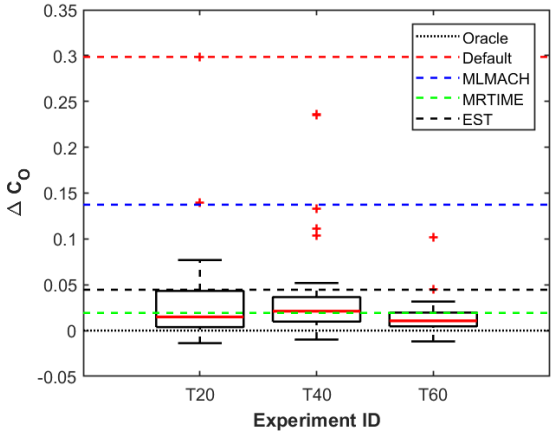

Fig. 7. Performance of hyper-heuristics trained with different number of instances, over the first test set. Each box shows data of 30 repetitions. Horizontal lines are used to show the performance of each heuristic over the test set. ΔC_O is defined in Eq. (1).

replicated the behavior of the worst heuristic, yielding a performance delta of about 0.300.

As in the case of the training data, using 40 instances for training seemed to worsen hyper-heuristics as the median performance increased in about 42%. Nonetheless, stability improved. The worst-case scenario and the range of data diminished in more than 20%. Moreover, using all 60 instances further improved the results, as the range decreased an additional 54%. The median performance also improved, as it now yielded a gap of 0.011, representing a reduction of almost 30% with respect to the first scenario.

Moving on to the second test set we found a curious response. This set was supposed to be more difficult as it contained bigger problems which the hyper-heuristic model did not see when training. Nonetheless, our data show that it performed better (Figure 8). This could be due to two

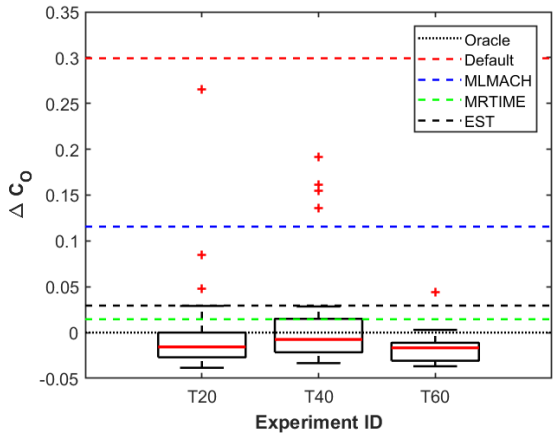

Fig. 8. Performance of hyper-heuristics trained with different number of instances, over the second test set. Each box shows data of 30 repetitions. Horizontal lines are used to show the performance of each heuristic over the test set. ΔC_O is defined in Eq. (1).

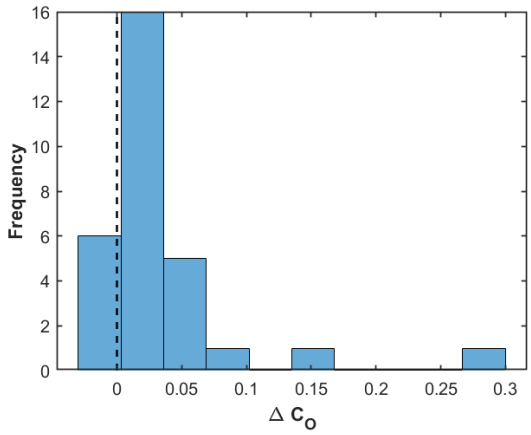

Fig. 9. Performance distribution of hyper-heuristics over the first test set, and trained with 20 instances (T-20). The vertical dashed line represents the performance of the Oracle. ΔC_O is defined in Eq. (1).

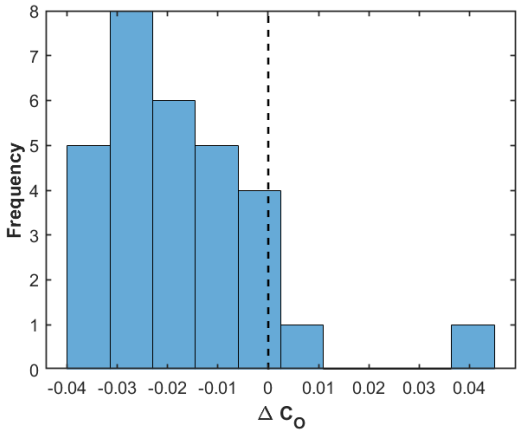

Fig. 10. Performance distribution of hyper-heuristics over the second test set, and trained with 60 instances (T-60). The vertical dashed line represents the performance of the Oracle. ΔC_O is defined in Eq. (1).

reasons: the problems are actually easier, or this problem domain has a particularity which makes hyper-heuristics able to generalize well when training with simpler problems. In any case, all median hyper-heuristics were able to outperform the Oracle. Moreover, for the T60 tests, almost all hyper-heuristics achieved such a feat. In the best-case scenario, the performance delta was of 0.037.

Trying to better analyze the way in which the performance of hyper-heuristic is distributed, we plotted histograms for each testing scenario. However, due to space restrictions, we are unable to show them all. Nonetheless, it is worth mentioning that for the first test case, the behavior for all three kinds of hyper-heuristics (i.e. T20, T40, and T60) was similar. As an example, consider Figure 9, which represents the T20 hyper-heuristics. As can be seen, the majority of hyper-heuristics exhibit similar performance, but worse, than the Oracle. However, in the second test case, their performance improves. Again, all kinds of hyper-heuristics behave similarly. Figure 10 shows the T60 case, as an example. Most hyper-heuristics now outperform the Oracle.

VI. Conclusions

Throughout this work, we trained hyper-heuristics with a different number of instances from the Job Shop domain. As far as we are aware, this is the first time that a selection hyper-heuristic powered by a Simulated Annealing approach is used for such an endeavor. We split our testing into three categories with a different number of instances: 20, 40, and 60. These instances were generated following the procedure laid out by Taillard [28]. Also, we defined two test scenarios, based on hard instances published by the same author: one similar to the training instances, and one with bigger problems. Our data suggest that using the highest number of instances improve the stability and performance of the generated hyper-heuristics. For example, for the first test scenario, using 60 instances instead of 20 led to a reduction in the data range of over 60% and to an improvement in the median performance of almost 30%.

Even though they were close, median hyper-heuristics were unable to outperform the Oracle in the first test scenario. However, by moving to the second scenario things changed and all median hyper-heuristics were able to outperform the Oracle. This makes the benefit of changing heuristics midway evident. Nonetheless, it also raises an interesting question about the difficulty of bigger problems and the way they relate to smaller ones. The problems analyzed in this work are supposed to represent the most difficult problems that Taillard found in previous work. So, it is possible that the features of problems with different sizes are somehow related, thus making it feasible to properly generalize bigger problems by training over smaller ones. This is a future path of research that we plan on exploring. Another interesting path is to analyze how performance is affected the other way around: training in bigger instances and testing on smaller ones.

We strongly believe that adding more features could prove fruitful for the hyper-heuristic model. An idea we aim to explore in future work is to add features representing the dispersion of machines and activities within each job. Another path worth pursuing is to add more low-level heuristics, striving to provide the hyper-heuristic with more tools that help it generalize better. Because of the exploratory nature of this work, we only considered one set of instances. Hence, future work should also analyze the performance of the proposed model on other benchmarks, such as the one from [29].

ACKNOWLEDGEMENT

This research was partially supported by CONACyT Basic Science Projects under grants 241461 and 287479, and ITESM Research Group with Strategic Focus in Intelligent Systems.

REFERENCES

[1] F. Zhao, J. Zhang, C. Zhang, and J. Wang, "An improved shuffled complex evolution algorithm with sequence mapping mechanism for job shop scheduling problems," *Expert Systems with Applications*, vol. 42, no. 8, pp. 3953 – 3966, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0957417415000226

[2] E. Ferreyra, H. Hagras, M. Kern, and G. Owusu, "Improving goal-driven simulation performance using fuzzy membership correlation analysis," in *2018 10th Computer Science and Electronic Engineering (CEEC)*. IEEE, 2018, pp. 254–259.

[3] 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, Dec 2003. [Online]. Available: https://doi.org/10.1007/s11518-006-0149-z

[4] M. Kurdi, "An effective new island model genetic algorithm for job shop scheduling problem," *Computers & Operations Research*, vol. 67, pp. 132 – 142, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0305054815002361

[5] B. J. Lageweg, J. K. Lenstra, and A. H. G. R. Kan, "Job-shop scheduling by implicit enumeration," *Management Science*, vol. 24, no. 4, pp. 441–450, 1977. [Online]. Available: https://doi.org/10.1287/mnsc.24.4.441

[6] J. Carlier and E. Pinson, "An algorithm for solving the job-shop problem," *Management Science*, vol. 35, no. 2, pp. 164–176, 1989. [Online]. Available: http://www.jstor.org/stable/2631909

[7] 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. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0925527398001716

[8] 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. [Online]. Available: http://www.jstor.org/stable/171189

[9] E. Nowicki and C. Smutnicki, "A fast taboo search algorithm for the job shop problem," *Management Science*, vol. 42, no. 6, pp. 797–813, 1996. [Online]. Available: https://doi.org/10.1287/mnsc.42.6.797

[10] 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.

[11] B. Peng, Z. L, and T. Cheng, "A tabu search/path relinking algorithm to solve the job shop scheduling problem," *Computers & Operations Research*, vol. 53, pp. 154 – 164, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0305054814002160

[12] J. H. BLACKSTONE, D. T. PHILLIPS, and G. Hogg, "A state-of-the-art survey of dispatching rules for manufacturing job shop operations," *International Journal of Production Research*, vol. 20, pp. 27–45, 01 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. [Online]. Available: http://www.jstor.org/stable/2632051

[14] 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, no. 1, pp. 43–66, Mar 2012. [Online]. Available: https://doi.org/10.1007/s10288-011-0182-8

[15] J. A. Vázquez-Rodríguez and S. Petrovic, "A new dispatching rule based genetic algorithm for the multi-objective job shop problem," *Journal of Heuristics*, vol. 16, no. 6, pp. 771–793, 2010.

[16] L. Davis, "Job shop scheduling with genetic algorithms," in *Proceedings of an international conference on genetic algorithms and their applications*, vol. 140. Carnegie-Mellon University Pittsburgh, Pennsylvania, 1985.

[17] P. J. Van Laarhoven, E. H. Aarts, and J. K. Lenstra, "Job shop scheduling by simulated annealing," *Operations research*, vol. 40, no. 1, pp. 113–125, 1992.

[18] M. L. Pinedo, *Scheduling: Theory, Algorithms, and Systems*. Springer, 2016.

[19] L. Wang, J.-C. Cai, and M. Li, "An adaptive multi-population genetic algorithm for job-shop scheduling problem," *Advances in Manufacturing*, vol. 4, no. 2, pp. 142–149, 2016.

[20] 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 & Industrial Engineering*, vol. 88, pp. 417 – 429, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0360835215003381

[21] T. C. Cheng, E., B. Peng, and Z. L, "A hybrid evolutionary algorithm to solve the job shop scheduling problem," *Annals of Operations Research*, vol. 242, no. 2, pp. 223–237, 07 2016.

[22] Z. Lian, B. Jiao, and X. Gu, "A similar particle swarm optimization algorithm for job-shop scheduling to minimize makespan," *Applied Mathematics and Computation*, vol. 183, no. 2, pp. 1008 – 1017, 2006. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0096300306006369

[23] T.-L. Lin, S.-J. Horng, T.-W. Kao, Y.-H. Chen, R.-S. Run, R.-J. Chen, J.-L. Lai, and I.-H. Kuo, "An efficient job-shop scheduling algorithm based on particle swarm optimization," *Expert Systems with Applications*, vol. 37, no. 3, pp. 2629 – 2636, 2010. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0957417409007696

[24] C. Blum and M. Sampels, "An ant colony optimization algorithm for shop scheduling problems," *Journal of Mathematical Modelling and Algorithms*, vol. 3, no. 3, pp. 285–308, Sep 2004. [Online]. Available: https://doi.org/10.1023/B:JMMA.0000038614.39977.6f

[25] 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. [Online]. Available: https://doi.org/10.1080/00207540802538021

[26] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983. [Online]. Available: http://science.sciencemag.org/content/220/4598/671

[27] G. Dueck and T. Scheuer, "Threshold accepting: A general purpose optimization algorithm appearing superior to simulated annealing," *Journal of Computational Physics*, vol. 90, no. 1, pp. 161 – 175, 1990. [Online]. Available: http://www.sciencedirect.com/science/article/pii/002199919090201B

[28] E. Taillard, "Benchmarks for basic scheduling problems," *European Journal of Operational Research*, vol. 64, no. 2, pp. 278 – 285, 1993, project Management anf Scheduling. [Online]. Available: http://www.sciencedirect.com/science/article/pii/037722179390182M

[29] 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. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0377221797000192