Exploring Problem State Transformations to Enhance Hyper-heuristics for the Job-Shop Scheduling Problem

Fernando Garza-Santisteban*, Ivan Amaya*, Jorge Cruz-Duarte*, José Carlos Ortiz-Bayliss*,
Ender Özcan† and Hugo Terashima-Marín*
*Tecnologico de Monterrey, School of Engineering and Science
Email: a00918788@itesm.mx, {iamaya2, jorge.cruz, jcobayliss, terashima}@tec.mx
†University of Nottingham, School of Computer Science, COL Lab
Email: ender.ozcan@nottingham.ac.uk

***Abstract*—This study presents an offline learning Simulated Annealing approach to generate a constructive hyper-heuristic evaluated through training and testing on a set of instances for solving the Job-Shop Scheduling problem. The generated hyper-heuristic uses a range of state features to control a set of low-level constructive heuristics. A hyper-heuristic is represented in terms of a set of rules, where each rule contains a fixed set of values for the features in consideration and the low level heuristic to be invoked. At each constructive step, the 'closest' rule is selected and then the corresponding constructive low level heuristic is applied. Our distance metric is the Euclidean distance between the values within the rule and the state features characterising the partial schedule along with the remaining jobs to be scheduled for the partial solution. In this paper, we study a set of features computed with various well-known metrics and different feature transformation methods for improving the characterization of the problem instances and solutions to Job-Shop Scheduling as a part of our approach. Eight different scenarios are evaluated on a set of randomly generated problem instances. Each scenario represents a distinct approach combining a different feature transformation applied during the training and testing phases. The empirical results show that transformations can improve the spread of feature values and the choice of the transformation methods is influential on the performance of the overall approach. A particular choice generates a slightly better performance when compared to the standard approach, which uses the original features at all times, indicating the potential of the proposed approach for the future studies.**

***Index Terms*—Hyper-heuristics, Job-Shop Scheduling Problem, Feature transformation, Combinatorial optimization**

I. Introduction

A Job-Shop scheduling problem (JSSP) is a challenging combinatorial optimization problem studied by many researchers and practitioners due to its many industrial and practical applications. Solving a JSSP involves searching for an arrangement of all jobs on a set of machines, subject to two constraints: each machine must handle at most one job at a time, and each job must respect a specified processing order. At the same time, the solution must minimize the time needed to complete all jobs, that is, the *makespan*. Although easily described, JSSP is an NP-hard problem.

There is a variety of approaches for solving a JSSP. Some of the previous related works include metaheuristics such as Simulated Annealing [1], Tabu Search [2], [3], and Genetic Algorithms [4], [5]. Heuristics, also known as "dispatching rules", represent another class of solution methods. They are computationally cheap construction rules making decisions throughout the search process to build a solution for a given instance. Hence, through them, a problem instance can be solved quite rapidly, but the solutions produced are likely to be suboptimal [6]. Some examples of heuristics were presented in the studies from Blackstone et al. [7], and Adams et al. [8].

Unfortunately, heuristics are often sensitive to the characteristics of the problem instance. Their performance could change dramatically from one instance to another. This variability in their performance represents one major drawback that has been the focus of attention in various works. One of the most important ideas used to tackle this drawback is that, when solving a given problem instance, the search incorporates a learning module (e.g., a metaheuristic). Then, the learning module applies heuristics in tandem, in such a way that the search is conducted by combining their strengths. This high-level method mixing low-level heuristics is broadly known as a hyper-heuristic (HH) [9]. The term *hyper-heuristic* was first used in 2000 to describe "heuristics to choose heuristics" [10]. A more recent definition proposes that a HH is an "automated methodology for selecting or generating heuristics to solve computational problems" [11]. We usually classify hyper-heuristics into constructive and perturbative [12], where both types of hyper-heuristics have been employed in different fields with promising results. For example, constructive hyper-heuristics have been applied to bin packing [13], [14], timetabling [15], and Job-Shop Scheduling [16] problems. Similarly, perturbative hyper-heuristics have been applied to packaging [17] and logistics [18] problems.

There are online and offline learning hyper-heuristics. This study focuses on the latter, where a hyper-heuristic gets trained on a set of sample instances. Furthermore, the model we

This work was supported in part by Consejo Nacional de Ciencia y Tecnología (CONACyT) Basic Science Project [Grant number 287479], and by ITESM Research Group with Strategic Focus in Intelligent Systems.

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

selected uses a set of features for mapping instances into actions. After the model is trained, it can be applied to unseen (i.e. test) instances. There is a vast number of works on hyper-heuristics, where some of them deal with different models, while others analyze diverse problems and ideas.

One of such ideas relates to transforming features. This approach aims at enhancing the 'readability' of the instance by separating conflicting regions while compressing the others. An example of such conflict is a region in the feature space where several rules interact, leading to small regions that are prone to errors throughout the training phase. So, if the evolution of a rule does not lead to the specific set of required values, then the hyper-heuristic may perform poorly. Hence, the model is sensitive to errors whenever training is 'placing' rules in the feature domain. However, when features are transformed, the conflicting regions are expanded, and it became less sensitive to the 'placement' error.

Despite the benefits that transforming features may have, only a few studies have investigated its effect in offline learning hyper-heuristics. A recent study targeted the domains of constraint satisfaction and knapsack problems [19]. However, we are unaware of previous work targeting feature transformations for solving JSSPs using an offline learning constructive hyper-heuristic.

The remainder of this document is as follows. Section. II presents the fundamentals that revolve around our proposal. Afterward, Sect. III summarizes our hyper-heuristic model and the way to carry out feature transformations for the JSSP. The experiments are presented in Sect. IV, whilst Sect. V discusses the results obtained from those experiments. Finally, we wrap up the work with the most relevant conclusions and some ideas for future work (Sect. VI).

II. Background and Related Work

This section presents the fundamental concepts required for understanding our approach.

A. The Job-Shop Scheduling Problem (JSSP)

Consider a set of n jobs $J = \{j_1, j_2, ..., j_n\}$ to be processed individually and sequentially on one or more machines from a set of m machines $M = \{m_1, m_2, ..., m_m\}$. Naturally, each processing time varies case-by-case. It is also assumed that the sequence of the i-th job on the k-th machine is given by a set of activities $A_i = \{a_{i,1}, a_{i,2}, ..., a_{i,m}\} \subseteq A$ with a corresponding set of processing times $T_i = \{t_{a_{i,1}}, t_{a_{i,2}}, ..., t_{a_{i,m}}\}$. Therefore, a JSSP consists in generating a timetable that minimizes the makespan and ensures that all jobs are completed. The makespan corresponds to the time at which the last activity of the whole schedule finishes. Also, a solution of the JSSP is only valid if it complies with the following constraints [20]:

1) *Precedence*—An activity can only be scheduled after the preceding ones have ended.
2) *Capacity*—Each machine can only process an activity at any given time.
3) *Non-preemption*—An activity cannot be interrupted when it has been scheduled into a machine.

B. Selection hyper-heuristics

Selection hyper-heuristics are general approaches that combine heuristics into a single solver. In this work, we use a constructive selection hyper-heuristic model (cf. Sect. III-A). Bear in mind that the role of a HH is to provide a heuristic h to use at each state S_t of a JSSP instance. Every S_t is represented as a set of predefined features. How the HH determines h is a topic of high research interest [12], [21], [22].

The main drawback of hyper-heuristics, when compared against heuristics, is that they are computationally more expensive since they usually require an iterative process for training the model. For example, Tabu Search [23] and Evolutionary Algorithms [24] have been used in the past. In this work, we consider another approach based on the Simulated Annealing (SA) algorithm [25]. Literature evidences its successful application in perturbative selection hyper-heuristics [26]–[28]. Also, this approach includes a stochastic acceptance criterion that has proven useful in the context of hyper-heuristic development and testing [29].

C. Feature Transformations

Transforming features is an idea that has already been explored in other disciplines, such as data mining [30]. The idea is to improve the 'shape' of the problem domain when mapped through the set of features so that a learning algorithm can differentiate states more easily and, thus, can be better trained. So, feature transformations represent a data preprocessing step aimed at reshaping feature values. The goal: aid a learning algorithm throughout the training phase. An illustrative example rests in the work of Yu et al. [31], where the authors studied people recognition from photographs taken with several camera views. They transformed features to account for feature distortions that arise from each camera.

Regarding hyper-heuristics, the incorporation of feature transformations has been scarce, as it began recently. Authors have mainly focused their efforts on the CSP and knapsack problem domains [19], [24]. Here, the idea of using such transformations has been to better differentiate among heuristics. This is achieved by changing features for their transformed versions. In doing so, the feature domain (i.e. what the hyper-heuristic 'sees') shifts and specific regions can be targeted. So, a sigmoid-based transformation can be used for prioritizing a given region by enhancing differences within the region of interest and tuning out values outside it. This way, the effect of altering the hyper-heuristic model (during training) can be more easily perceived. Hence, the process becomes less sensitive to differences between the ideal location of rules and values found at each training iteration. Furthermore, this also allows for higher flexibility as the resolution is increased over the region of interest. This, in turn, provides an opportunity for refining rules in such a way that better decisions are made, which may delay training stagnation.

The effect of feature selection on HHs has also been studied in the literature. However, we do not delve into it due to space restrictions. The interested readers are referred to [32], [33].

D. Simulated Annealing (SA)

Simulated Annealing (SA) was among the first metaheuristics to appear in the world. This approach was proposed by Kirkpatrick, Gelatt and Vecchi back in 1983 [25]. SA stands as one of the simplest and most general techniques available in the literature. This algorithm was inspired by the crystallization process of cooling a material. It begins by repeatedly suggesting random modifications to the current solution but progressively begins keeping only those that improve it. The algorithm uses a probabilistic rule to decide whether a new solution is accepted. Such a rule involves the change in fitness (i.e., the value of the objective function) by measuring how much it improves. It also includes a "temperature" parameter that reflects simulation progress.

Dueck and Scheuer presented a deterministic acceptance rule for SA [34]. They suggested accepting any random modification that does not worsen the solution by more than a given threshold. But, the threshold decreases as iterations progress. This variation of SA is known as Threshold Accepting (TA). Nonetheless, both versions of SA have some parameters that may be tuned. Among the most relevant ones is the cooling schedule, which determines how temperature is updated. More details on the different applications of SA can be consulted in [35], [36].

Simulated Annealing has been applied to many optimization problems, including combinatorial ones. Recent examples include the work of Karagul et al., where the authors tackled the Green Vehicle Routing Problem with Fuel Consumption [37]. Also, Wei et al. worked on the Capacitated Vehicle Routing Problem with two-dimensional loading constraints [38]. Other works have focused on studying different SA variants [39].

III. Our Proposed Approach

This section describes the tools that we used for studying feature transformation on JSSPs.

A. Selected hyper-heuristic model

In this work, we employed the hyper-heuristic (HH) model presented in [40], which is based on [13]. This model can be represented as an array of *blocks* (Fig. 1). Each *block* is defined by a set of features $\{f_{i,1}, f_{i,2}, ..., f_{i,q}\}$ and a solver (i.e., an action) to be used. Such a couple portrays a rule.

At each step of the solution, the HH calculates the current state of the problem S_t and compares it to each block. The comparison finds the closest block, in terms of Euclidean distance, and it returns the corresponding action. Note that the number of features on each rule may differ and that more than one rule may target the same action. Nonetheless, for this research, we used a fixed number of features on every rule.

B. Features considered in this work

Features can be defined from two perspectives. One relates to the state of the solution (i.e., schedule) and the other one to the state of the problem (i.e., what remains of the problem instance). We selected some features of both kinds, as described next.

- Solution features:
  1) *Average processing time (APT)*—Ratio between the summation of processing times of scheduled activities, and total processing time. This feature provides a rough progress estimate.
  2) *Dispersion of processing time index for scheduled activities (DPT)*—Ratio between the standard deviation of processing times and the mean of processing times, considering only the scheduled activities.
  3) *Makespan slack ratio (SLACK)*—Ratio between the amount of unused machine time (slack) in the whole schedule and the current makespan of the schedule. A higher slack indicates that activities are more spread out, though it also represents space that could be used to schedule smaller activities.
- Problem features:
  4) *Average pending time (NAPT)*—Complement of APT. It represents the ratio between the summation of processing times of pending activities, and total processing time.
  5) *Dispersion of processing time index for pending activities (DNPT)*—Equivalent to DPT, but considering only the pending activities.
  6) *Average pending processing time per job (NJT)*—Ratio of the average pending processing times per job and the total processing time.

C. Heuristics considered in this work

A heuristic is a simple rule that specifies which activity to schedule next. We selected them based on their performance, striving for variety, and both good and bad results. So, let $A_p \subset A$ be a list of activities to be scheduled. Recall the definitions of $a_{i,k}$ and $t_{a_{i,k}}$ from Sect. II-A. Besides, let $p_{i,k}$ be the time it takes to complete activity $a_{i,k}$. Hence, the following heuristics can be defined:

1) *Shortest Processing Time (SPT)*—Select the activity from A_p with the shortest $p_{i,k}$.
2) *Largest Processing Time (LPT)*—Select the activity from A_p with the largest $p_{i,k}$.
3) *Maximum Job Remaining Time (MRT)*—Sum the processing time of all pending activities for each job. Then, select the job with the highest value and return its upcoming activity (considering precedence).
4) *Most Loaded Machine (MLM)*—Find the machine with the maximum total processing time m_{k_M}. Return the feasible activity a_{i,k_M} with the lowest $t_{a_{i,k}}$. If no activity is feasible, ignore this machine and repeat the process.
5) *Least Loaded Machine (LLM)*—Similar to MLM, but considers the machine with the minimum total processing time m_{k_m}.
6) *Earliest Start Time (EST)*—Select the activity from A_p that has the earliest possible starting time.

D. Training of the hyper-heuristic model

Training a suitable hyper-heuristic for solving the JSSP implies finding appropriate features and action values for each

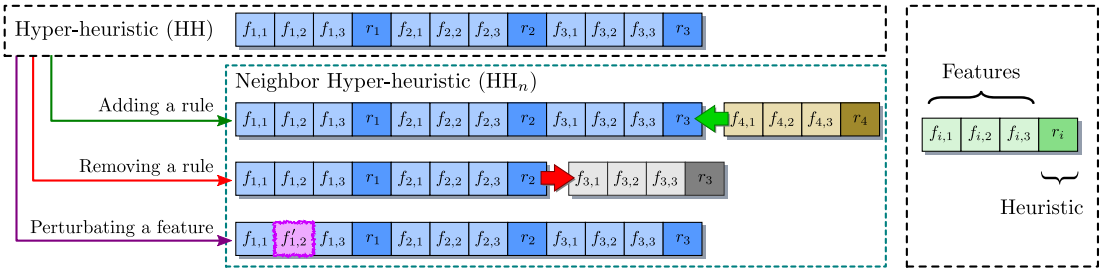


Fig. 1: Example of a hyper-heuristic with three rules (HH) and possible ways to generate a neighboring one. Each rule is formed by a set of features $f_{i,q}$, and a corresponding action r_i, which represents a call to a specific heuristic.

rule. There are many ways to carry out this process. However, we chose the Simulated Annealing (SA) algorithm, as mentioned above. Hence, SA minimizes the objective function (cf. Sect. III-E) following Algorithm 1 over a set of training instances. Note that a neighbor HH (line 5) is generated by mutating the current HH with one of the following approaches (Fig. 1):

1) *Add new rule*—The hyper-heuristic grows by one block with random values.
2) *Remove rule*—The hyper-heuristic shrinks by losing a randomly selected block. This process is omitted if the hyper-heuristic only has one block.
3) *Perturbate feature*—A random block and feature are selected and replaced by random values.

Algorithm 1 Hyper-heuristic training with SA

1: Set initial parameters: temperature θ and cooling rate c
2: Generate an initial random hyper-heuristic HH
3: Set the best hyper-heuristic as $\text{HH}^* = \text{HH}$
4: **while** System is not cool ($\theta >$ threshold) **do**
5: Create a neighbor hyper-heuristic HH_n
6: Calculate fitness function for HH and for HH_n
7: **If** ACCEPTANCE$(\text{HH}, \text{HH}_n) <$ rand **then** $\text{HH} = \text{HH}_n$
8: **if** fitness of HH $<$ fitness of HH^* **then** $\text{HH}^* = \text{HH}$
9: Update temperature $\theta = \theta(1 - c)$
10: **return** HH^*.

E. Objective function used in this work

We measure the hyper-heuristic performance by comparing its makespan ($C^{\text{HH}}_{s_i}$) with the one given by the best heuristic (C_{b_i}), i.e., the Oracle. Bear in mind that negative values are allowed and desired. This implies that the hyper-heuristic outperformed all heuristics. Since several instances are analyzed at every training iteration, the objective function must consider an accumulated value. Hence, $d(\text{HH}) : H \rightarrow \mathbb{R}$ represents a feasible objective function such as:

$$d(\text{HH}) = \frac{1}{N_I} \sum_{i=1}^{N_I} \frac{C^{\text{HH}}_{s_i} - C_{b_i}}{C_{b_i}}, \tag{1}$$

where HH is the hyper-heuristic, N_I is the number of instances that it solves.

F. Instances used in this work

We used two kinds of instances. They both relate to the work of Taillard [41]. One set of instances is used for training purposes and was created by us following the proposal laid out by Taillard. The other one is used for testing purposes and corresponds directly with instances published by the author. According to him, the latter represents the most difficult ones. All the instances have the following nature: fixed processing times, no set-up times, no due dates nor release dates, and processing times for each activity between 1 and 99.

G. Feature transformations used in this work

Many transformations can be applied to the features of a JSSP. Some of them have already been applied to other problem domains [19]. However, the main goal remains the same: to separate regions of influence for each heuristic. In this work, we analyzed two of such ideas, but for JSSPs: the linear (Φ_L) and the s-shaped (Φ_S) transformations. They are defined in (2) and (3). M_i and W_i are the mid-point and half-width of the transformation, respectively. These parameters are defined in terms of a transformation range, bounded by lower (a) and upper (b) values, as shown in (4). Such a definition allows analyzing the effects of different ranges on each feature. Moreover, it enables exploring how hyper-heuristic performance changes under such ranges.

$$\Phi_L(x_i, M_i, W_i) = \max\left\{0, \min\left\{1, \frac{x_i - M_i + W_i}{2W_i}\right\}\right\}, \tag{2}$$

$$\Phi_S(x_i, M_i, W_i) = 1 - \left(1 + e^{\frac{6M_i}{W_i}\left(\frac{x_i}{M_i} - 1\right)}\right)^{-1}, \tag{3}$$

$$M_i = (a + b)/2, \quad \text{and} \quad W_i = (b - a)/2. \tag{4}$$

To apply the transformations, the original feature value must be calculated and then mapped through the corresponding equation. Afterward, the hyper-heuristic model operates, as usual, measuring Euclidean distances and selecting the closest rule, as mentioned in Sect. III-A.

IV. Methodology

Throughout this work, we followed a dual-stage methodology. The idea was to determine how feature transformations may affect hyper-heuristic performance when tackling Job-Shop Scheduling Problems. So, we began by analyzing how

feature values change and then focused on hyper-heuristic performance.

A. Effect of transformations in feature values

Recall that the transformations used in this work (Sect. III-G) can be tailored. Through this stage, we focused on the dispersion across heuristics, seeking to differentiate the feature path followed by each solver.

The features considered in this work (cf. Sect. III-B) are associated with the problem or to the solution, respectively. Considering that only the initial values of the former change from one instance to the other (i.e., the initial solution is always empty), we began by analyzing said effect through the NJT and DNPT features. To do so, we applied the s-shaped transformation (ST) in such a way that it covered the whole [0, 1] range. The linear transformation was disregarded as it represents a direct mapping and thus had no effect over the data. We then analyzed the effect of the ST across the solution process. So, we gathered data about the original and transformed values at every step and for all features and heuristics. This was done to detect whether paths converge after some point, even if initially spread out by the ST.

B. Effect of combining transformations

As a second stage, we analyzed the effect of using transformed features when training hyper-heuristics. Although the set of features comprises six elements, it is convenient to select a pair of them for studying interactions more efficiently. Since NJT and DNPT can be studied without solving the problem, they were selected for testing. In particular, we aimed to assess the effect of assigning a different transformation to each feature. Thus, we analyzed all combinations of transformations for these two features, as shown in TABLE I. It is important to highlight that, for all tests, the transformation range was defined as [0.4, 0.7], based on the data that will be shown in Sect. V-A. Moreover, the four remaining features were considered in their original domain (i.e., without transformations). Each one of these experiments was repeated 30 times. Also, 30 instances of size 5×5 were used during training, whilst five instances of size 15×15 were used for testing.

TABLE I: Experiments for analyzing the effect of combining transformations in the [0.4, 0.7] range. O: Original feature value. S: S-shaped transformation. L: Linear transformation. In all cases, the remaining features were used in their original domain (i.e., without transformations).

Feature	E01	E02	E03	E04	E05	E06	E07	E08	E09
NJT	O	O	L	L	L	S	S	O	S
DNPT	O	L	O	L	S	L	S	S	O

V. Results

A. Effect of transformations in feature values

Fig. 2 shows the effect of using the s-shaped transformation (ST) over the NJT and DNPT features. As can be seen, the transformed values cover a wider range than the original ones. In fact, both ranges are almost doubled due to the transformation. Nonetheless, distribution shape is preserved.

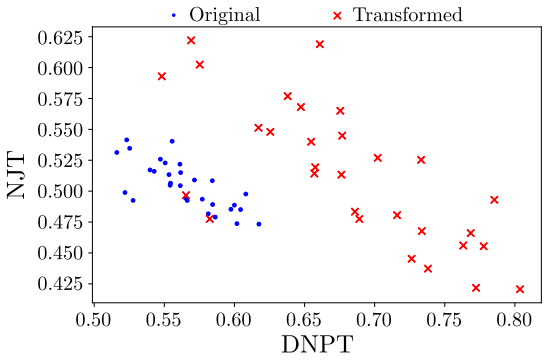


Fig. 2: Original (blue dots) and transformed (red crosses) feature values for 30 instances of size 15×15, when considering an s-shaped transformation covering the [0, 1] range.

It is also important to diversify paths that the features follow when solving an instance. Fig. 3 compares the original and transformed paths for each feature and heuristic, for a 15×15 instance. Because of the size of the instance, there are 225 actions to schedule, so 225 steps are required (each step schedules an action). Although separation among paths is not quite evident in all cases, the spread for NJT, DPT, and SLACK is clear. Such a separation is measured at each step of the solution. A higher separation means that heuristics yield partial solutions that are more different among them. Thus, the effect of using one or the other should be easier to identify. This, in turn, may be fruitful when training hyper-heuristics.

Consider the NJT feature as an example. The variation range of original values was below 0.1 units for most of the solution process. However, by implementing the s-shaped transformation, such a range grew to over 0.2 units. A similar effect occurred with the DNPT feature, though this time, the whole feature values also shifted upwards. In the case of the SLACK feature, the transformed feature can achieve a spread of about half the range for most of the solution process. The original feature, instead, was never able to achieve such a feature spread.

Despite feature diversity, it is interesting to analyze whether the distribution of features is affected by the transformation. Even though values change, in most cases, the general behavior of the feature holds. The only exceptions are for features with an approximately linear behavior. That is, features APT and NAPT. The reason: The mathematical model of the transformation maps a straight line between [0, 1] into a line that follows an s-shape in the same range. Hence, linear relationships for small ranges should be mapped into similar relationships with higher slopes. Please note that such an effect becomes stronger as the range grows smaller and as long as it does not belong to extreme values of the transformation. So, this kind of transformation provides the right way of enhancing feature diversity without compromising feature behavior.

As it was mentioned in the Methodology, using the linear

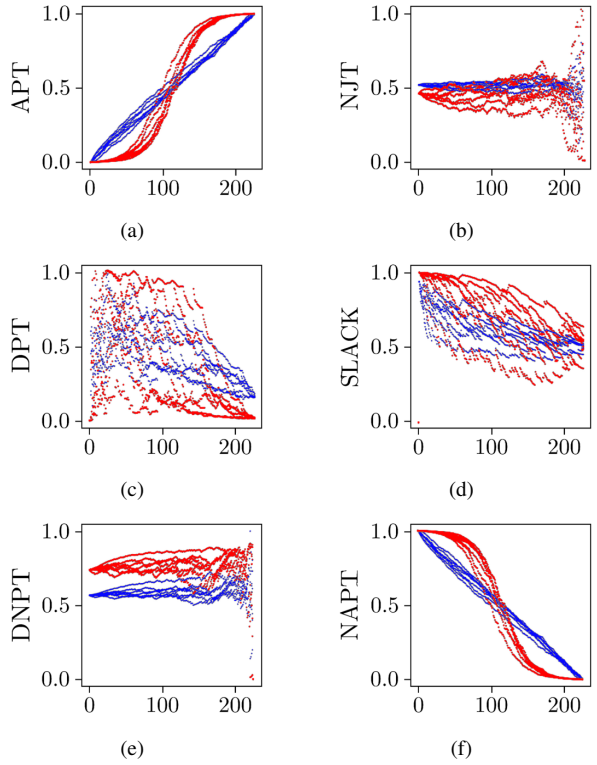


Fig. 3: Original (blue) and transformed (red) feature values for all heuristics when solving an instance of size 15×15, and considering an s-shaped transformation in the [0, 1] range. Horizontal axis: Step number. Vertical axis: Feature values. Lines: Path of feature values generated by each heuristic. Each subplot represents a different feature: (a) APT, (b) NJT, (c) DPT, (d) SLACK, (e) DNPT, and (f) NAPT.

transformation (LT) has no effect whatsoever. Nevertheless, this does not imply that such a transformation is useless. Consider, for example, Fig. 4, where the original and transformed values of the APT feature are shown. Here, using a LT with a range between [0,0.1] seems to improve the spread for the lower region. Although, it worsens the spread after the feature reaches the value of one. This indicates that additional tuning should be done for each transformation to detect the best parameters for each feature and transformation.

Since we will now move on to analyze the HH performance, it is necessary to define a range for the transformations. For this work, the only features that will be transformed are NJT and DNPT (cf. TABLE I). It is essential to consider not only the initial feature values (Fig. 2), but also the values that could be selected throughout the search. Even though Fig. 3 provides such a variation, it is important to remark that these values are only valid when using the same heuristic to solve the whole instance. Nevertheless, Fig. 3 allows us to glimpse at the feature range that could be expected. Both features begin slightly over the 0.5 mark. Afterward, the NJT remain virtually stable for some heuristics, while it diminishes for others, nearing the 0.4 mark. After half the problem has

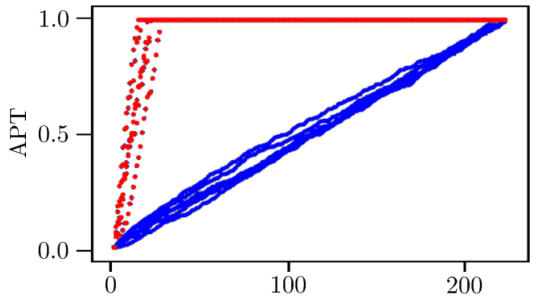


Fig. 4: Original (blue) and transformed (red) values of the APT feature for all heuristics when solving an instance of size 15×15, and considering a linear transformation in the [0, 0.01] range.

been solved, all heuristics begin providing increasingly higher feature values. When the problem is nearing completion, features oscillate and reach both higher and lower values. In the case of the DNPT feature, the effect is slightly different. Only a couple heuristics remain stable while the others begin increasing the feature value, reaching values around the 0.7 mark. The only time when the feature falls below 0.5 is at the last steps of the solution and for only one heuristic. Because of this, the transformation range [0.4, 0.7] is defined for both features.

B. Effect of combining transformations

Fig. 5(a) shows the training data for each combination of feature transformation. Here, experiment E02 (O, L) seems to be the best result. Even though the numeric difference across scenarios seems small, the gain in stability is evident. In fact, the range of hyper-heuristic performance diminished by almost 50%. Moreover, the mean performance improved by almost 0.01 units. Nonetheless, the median performance did not improve as much, implying that data is skewed. However, another set of transformations performed well. Our data showed that linearly transforming NJT and using an s-shaped transformation for DNPT also improves performance. Actually, under these conditions, both median and mean are quite close. They also represent a similar improvement to the already mentioned scenario (almost 0.01 units). Also, this combination of transformations did not improve stability. Besides, there was not a single hyper-heuristic (for any scenario) that outperformed the Oracle.

Fig. 5(b) shows performance data over the testing set. This time, however, another scenario reigned. The (L, O) combination (E03) yielded the best hyper-heuristic performance. Nonetheless, once again, there was no significant improvement concerning the other cases. No combination of transformation allowed shrinking the variance of hyper-heuristic performance. Even so, the best performing hyper-heuristic when linearly transforming NJT was about 0.25 units better than the best traditional hyper-heuristic. A similar effect also happened to the mean and median performing hyper-heuristics. Although, to a lower degree. In the first case, a difference of only about

0.03 units was achieved. In the second one, however, the gain raised to about 0.12 units.

It is important to highlight that all testing scenarios contained hyper-heuristics that outperform the Oracle. This is rather interesting since this feat was impossible at the training phase when HHs were being optimized to solve the instances. A possible explanation for such a behavior relates to the difficulty of instances. Bear in mind that, although all instances were generated with the same approach, the ones used for testing are those Taillard found to be most challenging.

VI. Conclusion and future work

In this study, we explored the influence of feature transformations on the performance of an offline learning constructive hyper-heuristic (HH) for Job Shop Scheduling Problems (JSSPs). We began by analyzing how feature spread changes when solving the problem with different heuristics and transformation models. Then, we observed the hyper-heuristic performance under different transformation conditions.

Our tests revealed that transformations can, indeed, significantly change the feature paths traversed by each heuristic (cf. Fig. 3). Even so, this does not seem to alter the shape of the path beyond the own shape of the transformation. For example, in the case of the s-shaped transformation (Fig. 3) only features APT and NAPT changed shape. The reason for this was the linear behavior of the original feature, which entailed a change into the s-shaped behavior of the transformation. However, all other features preserved their behavior. Nonetheless, they changed by expanding the differences at a single solution step. For example, in the case of the NJT feature, spread became about twice as big.

Even though this should have improved hyper-heuristic performance, something different happened. We detected that transformations mainly seem to increase hyper-heuristic stability. It means that the performance of two HHs trained under the same conditions will be more similar when using a proper transformation than when using the original feature values. Hence, a lower number of repetitions may be required for representing adequate behavior when using hyper-heuristics. In the case of the training set, linearly transforming DNPT with the [0.4, 0.7] range yielded the most stable results. It yielded a performance range of about 50% smaller than the base case. Despite this, our data revealed that transformations tend to worsen the hyper-heuristic stability in the test set. Such behavior may rest on the noise provided by features that remained in their original state. These may have a stronger effect than the transformed features, thus biasing conclusions. Even so, an interesting behavior arose. No hyper-heuristic (with or without transformation) was able to outperform the Oracle in the training set. However, in all testing scenarios, at least 25% of the HHs outperformed the Oracle. This may be due to instance difficulty or, to instance likeliness. We plan on further exploring such elements in upcoming studies.

In summary, even though feature transformations increased feature dispersion across heuristics in a clear way, the hyper-heuristic performance was not as strongly influenced. We believe that this behavior is, in part, due to the effect of the features that remained in their original state. However, it may also be due to the need to use more specific ranges for each feature. Thus, coming up with a methodology that automatically tunes a transformation range may benefit hyper-heuristic performance. Such an approach will be explored in future work. We plan on using a pre-processing step for analyzing instances and determine the most convenient transformation range for each feature. Another avenue for future work lies in analyzing the effect of other transformations. This may be as simple as using other mathematical models or as complex as using approaches such as Fuzzy logic. Finally, this work only considered a rule-based selection hyper-heuristic due to its exploratory nature. So, we plan on extending this work to other HH models as well as to other problem domains.

References

[1] 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.

[2] 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.

[3] 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.

[4] 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)*, pp. 7–14, 2015.

[5] 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*, pp. 549–554, 2011.

[6] P. Todd, "Heuristics for decision and choice," in *International Encyclopedia of the Social & Behavioral Sciences* (N. J. Smelser and P. B. Baltes, eds.), pp. 6676 – 6679, Oxford: Pergamon, 2001.

[7] 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.

[8] 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.

[9] J. H. Drake, A. Kheiri, E. Özcan, and E. K. Burke, "Recent advances in selection hyper-heuristics," *European Journal of Operational Research*, 2019.

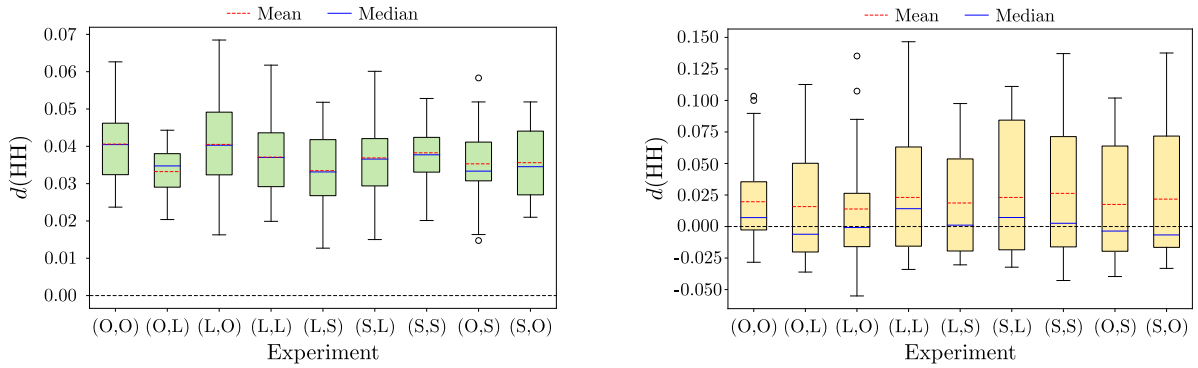
[10] P. Cowling, G. Kendall, and E. Soubeiga, "A hyperheuristic approach to scheduling a sales summit," in *International Conference on the Practice and Theory of Automated Timetabling*, pp. 176–190, Springer, 2000.

[11] E. K. Burke, M. R. Hyde, G. Kendall, G. Ochoa, E. Özcan, and J. R. Woodward, "A classification of hyper-heuristic approaches: Revisited," in *Handbook of Metaheuristics*, pp. 453–477, Springer, 2019.

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

[13] P. Ross, S. Schulenburg, J. G. Marín-Bläzquez, and E. Hart, "Hyper-heuristics: learning to combine simple heuristics in bin-packing problems," in *Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation*, pp. 942–948, Morgan Kaufmann Publishers Inc., 2002.

[14] K. Sim and E. Hart, "Generating single and multiple cooperative heuristics for the one dimensional bin packing problem using a single node genetic programming island model," in *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation*, GECCO '13, (New York, NY, USA), pp. 1549–1556, ACM, 2013.



(a) Training

(b) Testing

Fig. 5: Performance of hyper-heuristics over the training (a) and testing (b) sets, under all conditions described in Table I (30 runs). $d(HH)$ represents the delta w.r.t. the best available solution, as defined in Equation (1).

[15] E. K. Burke, B. McCollum, A. Meisels, S. Petrovic, and R. Qu, "A graph-based hyper-heuristic for educational timetabling problems," *European Journal of Operational Research*, vol. 176, no. 1, pp. 177–192, 2007.

[16] F. Garza-Santisteban, *Feature Transformations for Improving the Performance of SelectionHyper-heuristics on Job Shop Scheduling Problems*. M.sc. thesis, Tecnologico de Monterrey, 2019.

[17] K. A. Dowsland, E. Soubeiga, and E. Burke, "A simulated annealing based hyperheuristic for determining shipper sizes for storage and transportation," *European Journal of Operational Research*, vol. 179, no. 3, pp. 759–774, 2007.

[18] S. Ropke and D. Pisinger, "An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows," *Transportation science*, vol. 40, no. 4, pp. 455–472, 2006.

[19] I. Amaya, J. C. Ortiz-Bayliss, A. Rosales-Perez, A. E. Gutierrez-Rodriguez, S. E. Conant-Pablos, H. Terashima-Marín, and C. A. C. Coello, "Enhancing selection hyper-heuristics via feature transformations," *IEEE Computational Intelligence Magazine*, vol. 13, pp. 30–41, May 2018.

[20] C. Mencía, M. R. Sierra, and R. Varela, "Depth-first heuristic search for the job shop scheduling problem," *Annals of Operations Research*, vol. 206, pp. 265–296, Jul 2013.

[21] X. F. C. Sanchez-Diaz, *Analysis of a Feature Independent Hyper-heuristic model for Constraint Satisfaction and Binary Knapsack Problems*. M.sc. thesis, Tecnologico de Monterrey, 2017.

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

[23] J.H. Drake, A. Kheiri, E. Özcan, and E.K. Burke, "Recent advances in selection hyper-heuristics," *European Journal of Operational Research*, vol. 285, no. 2, pp. 405–428, 2020.

[24] I. Amaya, J. C. Ortiz-Bayliss, A. E. Gutiérrez-Rodríguez, H. Terashima-Marín, and C. A. C. Coello, "Improving hyper-heuristic performance through feature transformation," in *2017 IEEE Congress on Evolutionary Computation (CEC)*, pp. 2614–2621, June 2017.

[25] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.

[26] B. Bilgin, E. Özcan, and E. E. Korkmaz, "An experimental study on hyper-heuristics and exam timetabling," in *International Conference on the Practice and Theory of Automated Timetabling*, pp. 394–412, Springer, 2006.

[27] 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, Mar 2012.

[28] M. Kalender, A. Kheiri, E. Özcan, and E. K. Burke, "A greedy gradient-simulated annealing selection hyper-heuristic," *Soft Computing*, vol. 17, no. 12, pp. 2279–2292, 2013.

[29] K. Danach, *Hyperheuristics in Logistics*. PhD thesis, Ecole Centrale de Lille, 2016.

[30] S. García, J. Luengo, and F. Herrera, *Data preprocessing in data mining*. Springer, 2015.

[31] H.-X. Yu, A. Wu, and W.-S. Zheng, "Unsupervised person re-identification by deep asymmetric metric," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2018.

[32] M. Montazeri, "Hhfs: Hyper-heuristic feature selection," *Intelligent Data Analysis*, vol. 20, no. 4, pp. 953–974, 2016.

[33] E. Hart, K. Sim, B. Gardiner, and K. Kamimura, "A hybrid method for feature construction and selection to improve wind-damage prediction in the forestry sector," in *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 1121–1128, ACM, 2017.

[34] 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.

[35] A. Franzin and T. Stützle, "Revisiting simulated annealing: A component-based analysis," *Computers and Operations Research*, vol. 104, pp. 191–206, 2019.

[36] D. Delahaye, S. Chaimatanan, and M. Mongeau, "Simulated Annealing: From Basics to Applications," in *Handbook of Metaheuristics* (M. Gendreau and J.-Y. Potvin, eds.), ch. 1, pp. 1–35, third ed., 2019.

[37] K. Karagul, Y. Sahin, E. Aydemir, and A. Oral, "A simulated annealing algorithm based solution method for a green vehicle routing problem with fuel consumption," in *Lean and Green Supply Chain Management*, pp. 161–187, Springer, 2019.

[38] L. Wei, Z. Zhang, D. Zhang, and S. C. Leung, "A simulated annealing algorithm for the capacitated vehicle routing problem with two-dimensional loading constraints," *European Journal of Operational Research*, vol. 265, no. 3, pp. 843–859, 2018.

[39] A. Franzin and T. Stützle, "Revisiting simulated annealing: A component-based analysis," *Computers & Operations Research*, vol. 104, pp. 191–206, 2019.

[40] 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, jun 2019.

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