Improving Hyper-heuristic Performance through Feature Transformation

Ivan Amaya*, José Carlos Ortiz-Bayliss*, Andrés Eduardo Gutiérrez Rodríguez*,
Hugo Terashima-Marín*, Carlos A. Coello Coello†
*School of Engineering and Sciences
Tecnologico de Monterrey
Email: {iamaya2, jcobayliss, terashima}@itesm.mx
† CINVESTAV-IPN (Evolutionary Computation Group)
Email: ccoello@cs.cinvestav.mx

***Abstract*—Hyper-heuristics are powerful search methodologies that can adapt to different kinds of problems. One element of paramount importance, however, is the selection module that they incorporate. Traditional approaches define a set of features for characterizing a problem and, thus, define how to best solve it. However, some features may vary nonlinearly as the solver progresses, requiring higher resolution in specific areas of the features domain. This work focuses on assessing the advantage of using feature transformations to improve the given resolution and, as a consequence, to improve the overall performance of a hyper-heuristic. We provide evidence that using feature transformations may result in a better discrimination of the problem instance and, as consequence, a better performance of the hyper-heuristics. The feature transformation strategy was applied to an evolutionary-based hyper-heuristic model taken from the literature and tested on constraint satisfaction problems The proposed strategy increased the median success rate of hyper-heuristics by more than 13% and reduced its standard deviation in about 7%, while reducing the median number of adjusted consistency checks by almost 30%.**

I. Introduction

The algorithm selection problem [1] is the task of selecting the most suitable algorithm for a particular problem, among a set of available ones. This idea is based on the highly variable properties of problem instances and on the irregular performance of each solver. Algorithm selection is a generic problem, as it appears in many different domains. The past few years have witnessed a rapid growth in the number of techniques that address this problem by relating problem instances to one or more suitable solving strategies. Examples of algorithm selection strategies include, but are not limited to: algorithm portfolios [2], [3], [4], selection hyper-heuristics [5], [6] and instance specific algorithm configuration (ISAC) [7]. In general, all these methods manage a set of solving strategies and apply one that is suitable to the current problem state of the instance being solved. Striving to unify terms, from this point on, we will use *selection hyper-heuristic* to refer to the methods proposed in this paper.

Selection hyper-heuristics attempt to learn patterns of different human-designed heuristics to robustly tackle a wider range of problems on various domains [8]. A hyper-heuristic tries to find the right heuristic for a particular situation rather than trying to solve a problem directly [9]. The idea behind hyper-heuristics is to automate as much of the algorithm design process as possible [10].

One of the main problems related to hyper-heuristics is how to properly characterize the instances to allow a correct mapping into heuristics. This problem is not only related to feature selection, as it covers a wider spectrum (e.g., when the set of available features does not reflect the properties of the problems). In this work, we propose a method for transforming the features used by the selectors in such a way that the predictive power of the new features increases, obtaining a better mapping from instances to heuristics.

Summarizing, this investigation has to main contributions:

- It suggests that it is possible to increase the descriptive power of the features used to characterize a problem by applying transformations that increase the resolution of some specific regions of the feature space.
- It provides statistical evidence that transforming the features that characterize the problem state may improve the search when using rule-based hyper-heuristics.

The rest of the paper is organized as follows. Section II presents the foundations of this work. The problem itself, as well as the proposed solution and the domain where it was tested, are described in Section III. Section IV describes the three-step methodology adopted in this work. Section V presents the experiments and the results obtained. Finally, our conclusions and some possible paths for future work are laid out in Section VI.

II. Foundations

Hyper-heuristics are strategies for selecting one from among several approaches for dealing with the current state of a given problem [11], [12]. They are mainly comprised of a selection module and of a core with some kind of learning algorithm. The former is usually represented as a set of rules, where each one is a conjunction of feature-value items of the problem, and a final value corresponding to a specific heuristic. The latter, uses a set of training scenarios for finding the optimum feature values (and even the heuristics) that lead to the best selection for all test problems. Some details regarding these two components are provided in the next lines.

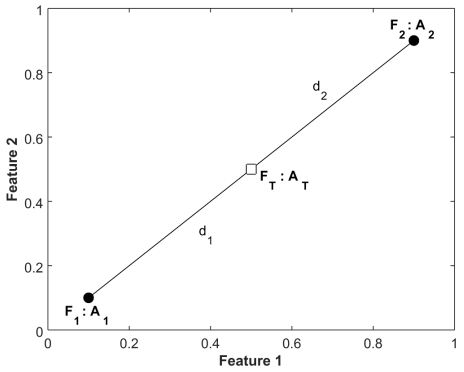

Fig. 1. Action selection for rule-based hyper-heuristics. Black dots: rules. White box: current state of the problem.

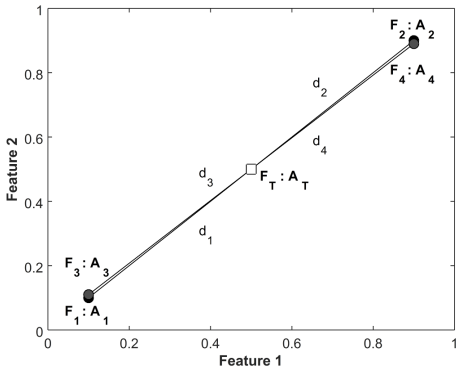

Fig. 2. Action selection for rule-based hyper-heuristics. Black dots: previous rules. Gray dots: new rules. White box: current state of the problem.

Hyper-heuristics need a way of assessing the agreement between the current state of a problem and that of the N problems it has already learned. One way of doing so is by calculating the Euclidean distance from the current state to each of the learned ones.[1] Should the distance be small enough, then both problems are considered similar. Thus, the action for the next step of the solution (A_T) is selected following the closest neighbor rule, or $A_T = A_j$, where $j = arg\min ||F_T - F_i||; i = 1, 2, 3, \ldots, N$. Figure 1 shows an example for $N = 2$ rules (black dots). After calculating the distance from the current test state (white box) to each rule (d_1 and d_2 respectively), an action is selected depending on whether d_1 or d_2 is the smallest value. In the first case, the next action is selected as A_1. Otherwise, $A_T = A_2$.

The core of each hyper-heuristic rests on its learning algorithm, since its task is to improve the selection module so that it performs appropriately. Producing the rules that minimize the cost is seen as an optimization problem, as shown in (1), where $C_i(X, A)$ represents the cost of evaluating problem i (out of N_p) with the rules defined by feature values X, and with its corresponding actions A.

$$\begin{aligned} \min : & F(X, A) \\ & F(X, A) \quad = \sum_{i=1}^{N_p} C_i(X, A) \end{aligned} \tag{1}$$

III. Problem description

This section presents a brief description of the problems that arise when using features, as well as the way in which feature transformation may alleviate them. Some transformations are shown with exemplary purposes. Also, some comments are given about the domain selected for this study, including the considered features and heuristics.

[1]There are other types of hyper-heuristics, such as those based on probabilistic models [13], reinforcement learning [14] and case-based reasoning [15], among others. However, due to space restrictions, and striving to keep this manuscript as simple as possible, their discussion is omitted.

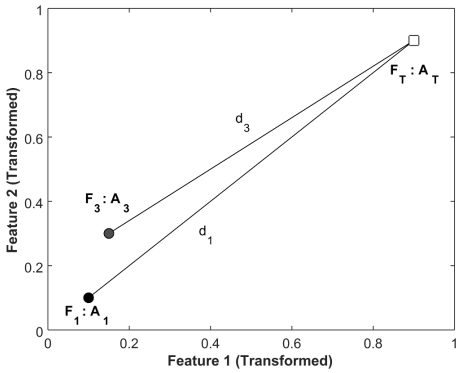

Fig. 3. Action selection for rule-based hyper-heuristics with transformed features. Black dot: initial rule. Gray dot: new rule with similar feature values. White box: current state of the problem. Note: rules two and four are not shown in order to simplify the plot.

A. The problem with features and how transforming them may be of help

Hyper-heuristics require a set of features that properly identify the current state of a problem, and they are usually normalized to balance their importance over the selection process. However, simply normalizing a feature may not be enough. Rule-based hyper-heuristics with features directly extracted from the optimization problem may exhibit two drawbacks:

- **Likeliness**: There may be two or more problem states with similar features that are best solved by different strategies (Figure 2). Since features are very alike, their distance to a test point would also be alike, leading to sometimes selecting an inappropriate approach. Consequently, there could also be a large set of feature values where the best action remains constant, thus 'wasting' part of the range. Feature transformation may be of help here, since those small (or big) changes can be mapped to broader (or narrower) ranges (Figure 3), thus increasing (or reducing) the difference between the distance values and reducing the risk of selecting the wrong one (or

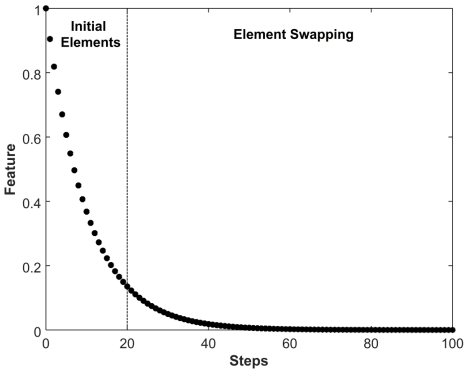

Fig. 4. Example of feature behavior concentrated over a given region (lower values in this case).

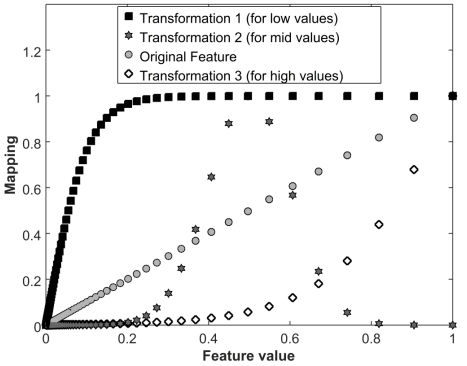

Fig. 5. Example of implementing feature transformations for increasing the resolution of specific regions from the main feature.

saving part of the range for differentiating other values).

- **Stagnation**: As the solution to an optimization problem begins to stagnate, features become almost invariant, as we describe next. Imagine there exists a feature that measures wasted space in an allocation problem. Thus, a value of one would indicate that all space is being wasted while a value of zero would imply perfect allocation. Most problems would likely start at one (since nothing has been allocated), but quickly migrate to lower values as elements are placed (Figure 4). However, a time will come when improvements become small (e.g. going from 0.05 to 0.04), even if relative big changes are being made (e.g., swapping two elements). It could also be the case that these two problem states are best solved by different approaches. Identifying this phenomenon and defining appropriate rules for dealing with them may prove difficult. However, by incorporating feature transformations, a set of values can be created for improving the resolution over different regions of interest (Figure 5). This way, the aforementioned difference of 0.01 could be expanded to a higher value, e.g. 0.07. Furthermore, functions can be defined as to allow for a higher span in these mappings. Thus, rule definition may be made into an easier task, likely improving the performance of its corresponding hyper-heuristic.

Different kinds of functions can be used for transforming features, which makes difficult for a researcher to select one of them. However, a distinction can be made depending on which region of the feature is of interest. For example, (2) and (3) were used in Figure 5 for mapping the original feature into low and high regions of interest, by setting $K = 20$ and $K = 7$, respectively. Nonetheless, should focus be given to the middle region, a bell-shaped transformation following (4) can be implemented, where μ relates to the center and σ regulates the span of the bell. An example of this transformation is shown in Figure 5 for $\mu = 0.5$ and $\sigma = 0.1$ It is worth remarking that the selection of these functions (or other the user may desire) is completely arbitrary and may or may not be an increasing function, but it should focus on expanding the range where most instances are located. In this sense, simpler functions such as $Y = e^{(-K \cdot X)}$ or $Y = 1 - e^{(-K \cdot X)}$ could be used for the transformation. However, preliminary testing carried out with them showed that they did not behave as good as (2).

$$Y = 1 - 2 \cdot \left(\frac{e^{-K \cdot X} - e^{-K}}{1 + e^{-K \cdot X}} \right) \tag{2}$$

$$Y = \frac{2}{1 + e^{-K \cdot (X-1)}} \tag{3}$$

$$Y = e^{-\frac{(X-\mu)^2}{2 \cdot \sigma^2}} \tag{4}$$

B. Domain under study

The feature-transformation strategy described in this work can be applied to any rule-based hyper-heuristic and to many different problem domains. However, due to space limitations, we focused our study on only one domain. We selected a set of constraint satisfaction problems (CSPs) to validate the proposed approach, mainly because of its many practical applications [16], [17]. CSPs are usually solved by traversing a depth-first search tree. At each node in the search tree, the algorithm must select an unassigned variable and one suitable value from its corresponding domain. If the current assignment of the variables breaks at least one constraint, the search backtracks and changes the value of a previously assigned variable, and then, resumes the search from the backtracking point.

The 322 CSP instances considered for this investigation are publicly available at a repository.[2] The specific sets of instances used in this investigation can be referred by the names given in the repository: `geom`, `ehi-85` and `bqwh-15-106`. From these instances, 5% were used for training and the remaining 95%, for testing purposes.

[2]http://www.cril.univ-artois.fr/~lecoutre/benchmarks.html

1) Heuristics and Problem Features: All the hyper-heuristics in this investigation work in the following way. They use a set of features to characterize the current problem state and, by using a set of rules (discovered automatically by using a messy genetic algorithm [5]), decide one suitable heuristic to apply. Once a heuristic has been selected, it decides, based on its own criterion, the next variable to assign. The process is repeated every time a variable is to be assigned. Thus, solving a CSP instance by using a hyper-heuristic derives in a sequence of different heuristics.

As the search progresses, hyper-heuristics use the characterization of the problem state to decide which heuristic to apply. Throughout this work, we focused on two features for characterizing the problem state:

- **Constraint density** (p_1): The global constraint density is calculated as the number of constraints in the instance divided by $n(n-1)/2$, which represents the maximum number of possible bidirectional constraints in the instance (where n stands for the number of variables in the instance).
- **Constraint tightness** (p_2): The local constraint tightness is calculated as the fraction of forbidden pairs of values in a given constraint. The global constraint tightness of a CSP instance is the average of the local constraint tightness among all the constraints.

By using the information from the features, one heuristic among the set of available options is chosen at each node. This work considers four different heuristics:

- **Domain (DOM)**: selects the variable with the fewest remaining values in its domain [18].
- **Degree (DEG)**: selects the variable with the largest degree [19], where the degree of a variable is the number of constraints where such a variable participates.
- **Kappa (K)**: selects first the variable that minimizes the κ value of the resulting instance [20]:

$$\kappa = \frac{-\sum_{c \in C} \log_2(1 - p_c)}{\sum_{x \in X} \log_2(d_x)} \tag{5}$$

  where p_c represents the tightness of constraint c (the proportion of forbidden pairs of values in the constraint) and d_x stands for the domain size of variable x.
- **WDEG**: attaches a weight to every constraint in the problem and increases it when its respective constraint fails during the search [21]. The weighted degree of a variable is calculated as the sum of the weights of the constraints in which the variable is currently involved. WDEG selects the variable with the largest weighted degree.

Once a variable has been selected by using one of the heuristics described above, the first available value in its domain is used for the assignment.

In order to compare the performance of different solvers for CSPs, in this work we have adopted the following performance metrics:

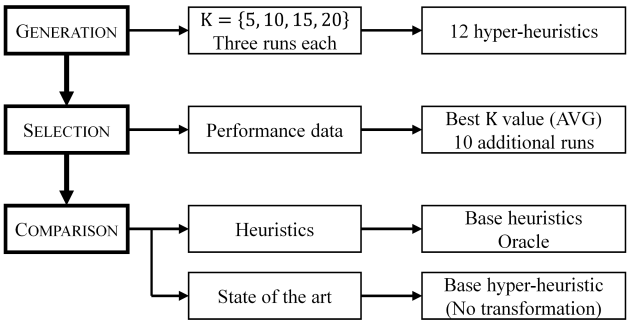

Fig. 6. Methodology followed throughout this work

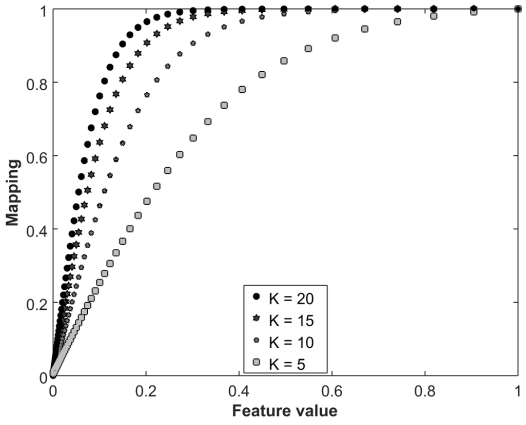

Fig. 7. Transformations applied in this work

- **Consistency checks (CC)**: The revisions of the constraints required to solve a given instance. The larger the number of consistency checks, the more expensive the search.
- **Adjusted consistency checks (ACC)**: This measure is similar to the previous one, but the adjustment discards the tests where the solver times out.
- **Success rate (SR)**: The success rate is a relation between the number of instances that a solver is able to complete, divided by the total number of problem instances tested. The higher the rate, the better the solver.

IV. Methodology

Throughout this work we followed a three-stage methodology (Figure 6), which is briefly discussed in the next lines.

- **Generation**: The first stage consists on generating a large enough number of hyper-heuristics so that conclusions can be derived from the resulting data. In this sense, we chose to generate 12 different hyper-heuristics, where each one used a transformation that highlighted one of four possible ranges, and with three hyper-heuristics per range. The hyper-heuristic generation process used in this investigation is an extension of the work described in [5]. This generation process is based on a messy genetic

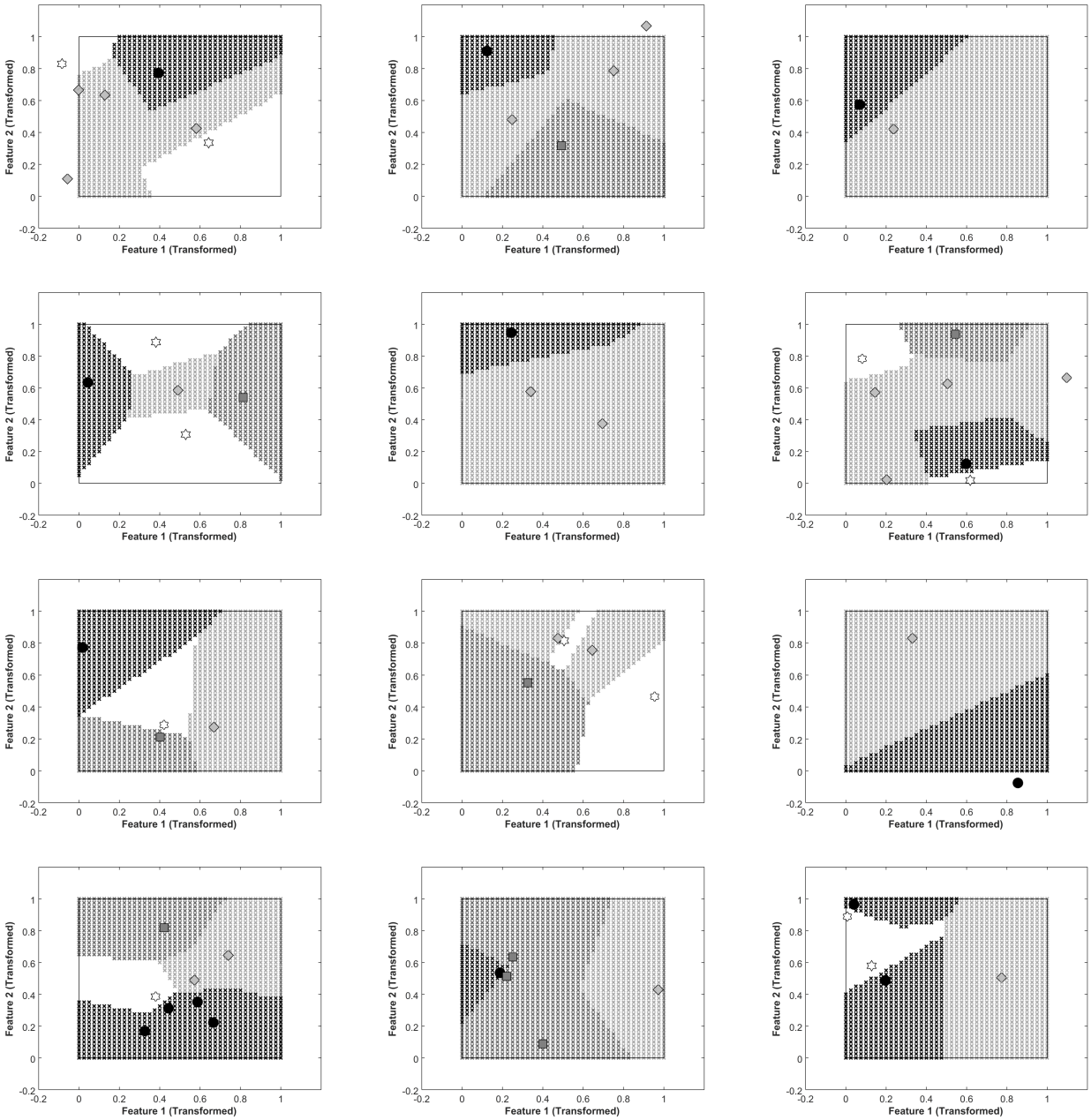

Fig. 8. Selectors generated and their corresponding actions (with influence zones). Black circle: DOM; Dark gray square: DEG; Light gray diamond: KAPPA; White star: WDEG. Each column represents a different run and each row (from top to bottom) represents a value of K, where $K = 5, 10, 15, 20$.

algorithm that produces a set of rules that minimize the number of revisions of the constraints when solving all the training instances (consistency checks). In all cases, the generation model ran with the following parameters: 20 individuals in the population, up to 100 cycles per run,[3] crossover rate of 1.0, and mutation rate of 0.1.

[3]Considering that using a fixed number of generations could lead to terminating the algorithm prematurely, or to simply wasting computing time, we decided to use a criterion based on the eventual performance stagnation. To do so, the algorithm stops after 5% of the total number of cycles has come and gone with no improvement, and where the current fitness sits below the average of the historical fitness, minus its standard deviation [22].

- **Selection**: After training and testing all hyper-heuristics, performance data will be analyzed, focusing on the number of consistency checks required to solve the instances, as well as on their success rate. Using this information, K will be selected as the one leading to the best average performance, and 10 new hyper-heuristics will be generated.
- **Comparison**: The final stage will compare the overall performance of the selected hyper-heuristics against two reference sets. The first one relates to the performance of each independent heuristic, as well as to the performance

of an Oracle: a synthetic strategy that always selects the best heuristic for every instance. This is done to verify that hyper-heuristics indeed outperform heuristics. The second batch of comparisons will focus on the performance of 'traditional' hyper-heuristics. In this sense, and considering that a total of 13 hyper-heuristics (for the best value of K) will be available at this point, the analysis will be done against 13 hyper-heuristics generated with no feature transformation [5], striving to detect any possible gain from the proposed approach.

V. Experiments and Results

The main results of our work are now presented, following the same structure presented in the methodology.

A. Hyper-heuristics Generation

As indicated before, we created three hyper-heuristics by using four different values of K: 5, 10, 15 and 20. These values were empirically defined from a preliminary experimentation phase. In order to test the feasibility of using feature transformations, we limited our tests to transforming each feature via (2), highlighting ranges up to $0.80, 0.45, 0.30$, and 0.23, respectively, as the result of the chosen values of K (Figure 7). Nonetheless, this does not limit the generality of our approach, since other transformations can also be used, either the ones shown in (3) and (4), or others that the user may define.

We produced three hyper-heuristics per value of K. To produce one hyper-heuristic, the genetic algorithm runs until the termination criteria is met. Each hyper-heuristic contains a set of rules in the form `features` $\rightarrow$ `heuristic`.

Figure 8 summarizes all hyper-heuristics generated by applying feature transformation, where each column represents a different run and where different values of K are shown in each row. It is evident that a high variability in the number of generated rules exists, ranging from two to eight. Moreover, their location is not always the same, and it partly depends on the number of rules generated. In some cases, overlapping two runs of the same transformation would place two rules with the same action on similar locations (e.g., runs two and three for $K = 5$). But, in others, rules with different actions will end up close by (e.g., runs one and two for $K = 10$), or in equivalent regions (e.g., runs one and two for $K = 5$). This, however, only implies that it is likely that both actions are good at solving the set of instances whose features rest at its corresponding region. This is more clearly seen when analyzing the overlapping of the influence zones across all runs for each K. For example, with $K = 5$ it is clear that most of the diagonal going from (0,0) to (1,1) can be properly solved with KAPPA (light gray diamonds), while the same happens for $K = 10$ but limited to the central region.

B. Hyper-heuristic Selection

After testing all the generated hyper-heuristics in the training set, it was clear that selectors created with $K = 5$ outperformed all others in virtually all scenarios (see highlighted values in Table I). Thus, we selected this value as the best K and created 10 additional hyper-heuristics. However, and due to space restrictions, they are not plotted in the paper. Nonetheless, the average performance of each value of K is compared below.

TABLE I
Consistency checks (CC), adjusted consistency checks (ACC), and success rate (SR) for all the generated hyper-heuristics, including the average (AVG) and standard deviation (SD) of each configuration. Values in **bold** represent the best result for each metric at every run

Performance	Run 1	Run 2	Run 3	AVG	SD
CC (K=5)	**410533**	**427049**	**391642**	**409741**	**17717**
CC (K=10)	795318	461162	793408	683296	192376
CC (K=15)	440387	719455	698086	619310	155319
CC (K=20)	749041	839898	756196	781712	50518
ACC (K=5)	**169815**	185069	**171410**	**175431**	**8384**
ACC (K=10)	462871	**182868**	307611	317783	140279
ACC (K=15)	188632	381979	222910	264507	103167
ACC (K=20)	228457	268794	225464	240905	24199
SR (K=5)	**90%**	**90%**	**90%**	**90%**	**0%**
SR (K=10)	74%	89%	60%	75%	15%
SR (K=15)	89%	77%	64%	77%	13%
SR (K=20)	63%	59%	64%	62%	2%

C. Hyper-heuristics Comparison

The final batch of results is presented below. As was stated in the methodology, data are split into two groups: one regarding base heuristics and the Oracle, and one comparing against traditional hyper-heuristics.

1) Performance of heuristics and Oracle: Comparing the performance of each individual heuristic against that of the Oracle reveals that the latter is significantly better than each heuristic on their own (Figure 9). Nonetheless, it is worth mentioning that the Oracle is a synthetic approach used in this work to denote the best possible performance that could be achieved for each instance. Thus, the existence of the Oracle does not imply the existence of a strategy that works best for every scenario. Instead, this Oracle operates on the principle that it is able to accurately predict which one of the available heuristics will perform best in every scenario. Or, in other words, it chooses a heuristic using the performance data of each available option for every instance. However, since running all solvers for every particular problem is not practical, the use of this technique for a purpose different to benchmarking is out of the question.

In spite of what has been said before, throwing hyper-heuristics into the mix yields interesting results. Even if it is evident that some values of K hinder performance (i.e., lower the success rate to around 60% and increase the number of adjusted consistency checks to about 300000), there are other values that drive the performance quite close to the ideal scenario. For the tests carried out in this stage, this implied reaching a success rate of over 90% (about 10% higher than the best heuristic, which required almost thrice the number of consistency checks) while requiring slightly above 20% more checks than the Oracle, but with a reduction of more than 20% when compared to the heuristic with lowest checks (which had a success rate lower than 65%). It is also important

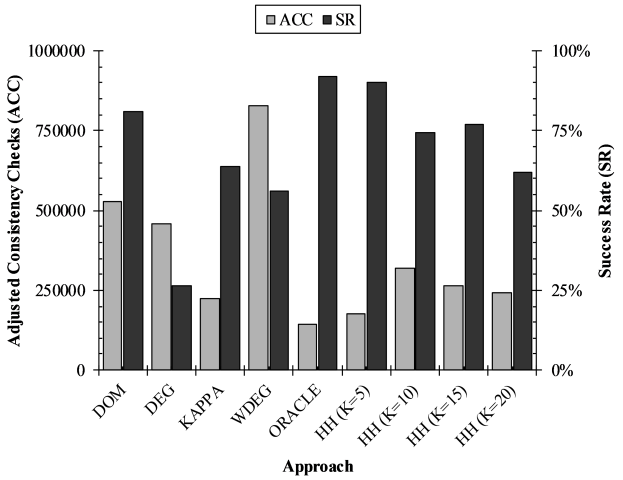

Fig. 9. Adjusted consistency checks (ACC) and success rates (SR) for all heuristics, as well as for the Oracle and for all hyper-heuristics (averaged).

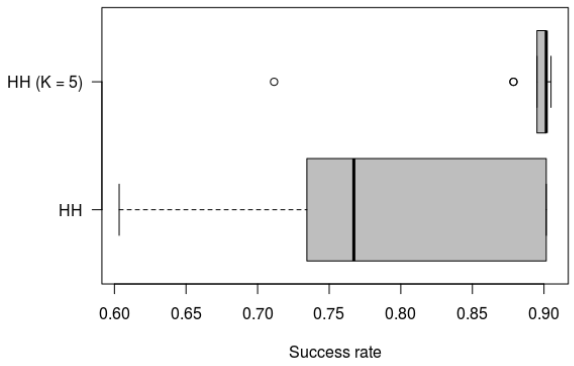

Fig. 10. Box plot of the success rates using feature transformation with $K = 5$ (top) and without it (bottom).

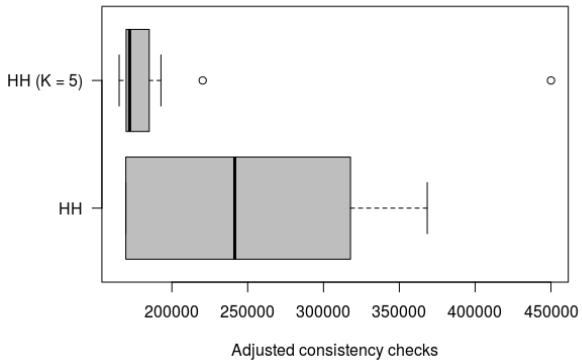

Fig. 11. Box plot of the adjusted consistency checks using feature transformation with $K = 5$ (top) and without it (bottom).

to highlight that, on average, all tested values of K required less checks than most heuristics (including the one with the highest success rate), while maintaining a success rate higher than most heuristics.

2) Performance of hyper-heuristics without feature transformation: Since there were 13 available results for the best value of K, we generated the same number of hyper-heuristics but disregarding feature transformations. There is an evident benefit in doing a proper transformation, since it helps in improving and stabilizing the success rate (Figure 10) and the number of consistency checks required for solving the dataset (Figure 11). For hyper-heuristics with no feature transformations, it is normal to expect success rates between 60% and 90%, with consistency checks ranging from below 200000 and all the way up to over 350000. However, when using $K = 5$, most of the data are concentrated on success rates of around 90% and consistency checks below 200000. Nonetheless, there are a few outliers with success rates of about 70% and with consistency checks of about 450000. But, they are precisely that (i.e., outliers) and not the core of our data.

A Welch's two-sample t-test carried out over all 13 samples of each set showed that there is a significant difference in the performance of both approaches, with 5% of significance. The statistical evidence suggests that the averaged results of the hyper-heuristics produced with feature transformation ($K = 5$) are better than the ones produced without such transformation in terms of adjusted consistency checks and success rate (p-values of 0.0310 and 0.0082, respectively).

VI. Conclusions and Future Work

This work described a feature transformation approach that improves the heuristic selection process conducted by hyper-heuristics. Incorporating feature transformations allowed hyper-heuristics to outperform the heuristics considered for this study when applied in isolation, while showing to be statistically as competent as a synthetic Oracle. Specifically, and compared to the heuristic with highest success rate (SR), we managed to reduce the number of adjusted consistency checks (ACC) by 2/3 while increasing SR by about 10%. When compared to the heuristic with lowest ACC, we managed to increase the SR by about 25% while reducing the number of ACC by more than 20%.

The hyper-heuristics generated in this work by incorporating feature transformation (particularly those with $K = 5$) were more stable than the ones produced with no transformation. It seems that the transformation not only represents an improvement to the heuristic selection process, but a reduction in the deviation of the hyper-heuristics produced through the messy genetic algorithm model [5]. In terms of the success rate (SR), using $K = 5$ increased the median by more than 13% while reducing its standard deviation in about 7%. At the same time, the median of the number of adjusted consistency checks (ACC) were reduced by almost 30%.

Some really interesting results that need to be analyzed in

more detail are depicted in Fig. 8. At the moment, all we can conclude from these figures is that there are similar regions assigned to one particular heuristic, but a further study needs to be conducted to get more insights into what these results really mean.

We are aware that the model requires to be refined. For example, the selection of the transformation is, at this point, completely arbitrary. We expect to extend this work and provide more ideas on how to properly define the most suitable transformation by using a semi-automatic methodology for this purpose. Other important trends for future work lies in fuzzy logic, since we have found similarities between our feature-transformation approach and some concepts in fuzzy logic. For example, the latter incorporates a set of membership functions to estimate the degree in which a given input state belongs to each rule. We think that exploring how these membership functions can be used as an equivalent to feature-transformation may represent an interesting idea for further study.

Finally, another path worth exploring is the use of different strategies for estimating the distance between the features of the problem state and the conditions in the rules. We think that a modification to the distance function can also improve the performance of rule-based hyper-heuristics.

ACKNOWLEDGMENTS

This research was supported in part by CONACyT Basic Science Projects under grant 241461 and ITESM Research Group with Strategic Focus in intelligent Systems. The last author gratefully acknowledges support from CONACyT project no. 221551.

REFERENCES

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

[2] 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*, ser. CP '02. London, UK, UK: Springer-Verlag, 2002, pp. 525–542.

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

[4] S. Petrovic and R. Qu, "Case-based reasoning as a heuristic selector in a hyper-heuristic for course timetabling problems," in *Proceedings of the 6th International Conference on Knowledge-Based Intelligent Information Engineering Systems and Applied Technologies (KES'02)*, vol. 82, 2002, pp. 336–340.

[5] J. C. Ortiz-Bayliss, H. Terashima-Marín, and S. E. Conant-Pablos, "A neuro-evolutionary hyper-heuristic approach for constraint satisfaction problems," *Cognitive Computation*, vol. 8, no. 3, pp. 429–441, 2016.

[6] K. Sim, E. Hart, and B. Paechter, "A lifelong learning hyper-heuristic method for bin packing," *Evol. Comput.*, vol. 23, no. 1, pp. 37–67, Mar. 2015.

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

[8] 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.

[9] E. K. Burke, G. Kendall, and E. Soubeiga, "A tabu-search hyperheuristic for timetabling and rostering," *Journal of Heuristics*, vol. 6, no. 9, pp. 451–470, 2003.

[10] G. L. Pappa, G. Ochoa, M. R. Hyde, A. x. Freitas, J. Woodward, and J. Swan, "Contrasting meta-learning and hyper-heuristic research: the role of evolutionary algorithms," *Genetic Programming and Evolvable Machines*, vol. 15, no. 1, pp. 3–35, 2014.

[11] M. Misir, K. Verbeeck, P. Causmaecker, and G. Berghe, "A new hyper-heuristic as a general problem solver: an implementation in HyFlex," *Journal of Scheduling*, vol. 16, no. 3, pp. 291–311, 2013.

[12] G. Ochoa, M. Hyde, T. Curtois, J. Vazquez-Rodriguez, J. Walker, M. Gendreau, G. Kendall, B. McCollum, A. Parkes, S. Petrovic, and E. Burke, *HyFlex: A Benchmark Framework for Cross-domain Heuristic Search*, ser. LNCS. Heidelberg: Springer, 2012, vol. 7245, pp. 136–147.

[13] W. Van Onsem, B. Demoen, and P. De Causmaecker, *Learning a Hidden Markov Model-Based Hyper-heuristic*. Cham: Springer International Publishing, 2015, pp. 74–88.

[14] E. Özcan, M. Misir, G. Ochoa, and E. K. Burke, "A reinforcement learning-great-deluge hyper-heuristic for examination timetabling," *Int. J. Appl. Metaheuristic Comput.*, vol. 1, no. 1, pp. 39–59, Jan. 2010.

[15] E. K. Burke, S. Petrovic, and R. Qu, "Case-based heuristic selection for timetabling problems," *Journal of Scheduling*, vol. 9, no. 2, pp. 115–132, 2006. [Online]. Available: http://dx.doi.org/10.1007/s10951-006-6775-y

[16] J. Berlier and J. McCollum, "A constraint satisfaction algorithm for microcontroller selection and pin assignment," in *Proceedings of the IEEE SoutheastCon 2010 (SoutheastCon)*, march 2010, pp. 348–351.

[17] S. V. Bochkarev, M. V. Ovsyannikov, A. B. Petrochenkov, and S. A. Bukhanov, "Structural synthesis of complex electrotechnical equipment on the basis of the constraint satisfaction method," *Russian Electrical Engineering*, vol. 86, no. 6, pp. 362–366, 2015.

[18] P. W. Purdom, "Search rearrangement backtracking and polynomial average time," *Artificial Intelligence*, vol. 21, pp. 117–133, 1983.

[19] R. Dechter and I. Meiri, "Experimental evaluation of preprocessing algorithms for constraint satisfaction problems," *Artificial Intelligence*, vol. 38, no. 2, pp. 211–242, 1994.

[20] I. P. Gent, P. Prosser, and T. Walsh, "The constrainedness of search," in *Proceedings of AAAI'96*, 1999, pp. 246–252.

[21] F. Boussemart, F. Hemery, C. Lecoutre, and L. Sais, "Boosting systematic search by weighting constraints," in *European Conference on Artificial Intelligence (ECAI'04)*, 2004, pp. 146–150.

[22] J. M. Cruz-Duarte, A. Garcia-Perez, I. M. Amaya-Contreras, and C. R. Correa-Cely, "Designing a microchannel heat sink with colloidal coolants through the entropy generation minimisation criterion and global optimisation algorithms," *Applied Thermal Engineering*, vol. 100, pp. 1052–1062, 2016. [Online]. Available: http://dx.doi.org/10.1016/j.applthermaleng.2016.02.109