Applying Automatic Heuristic-Filtering to Improve Hyper-heuristic Performance

Andres E. Gutierrez-Rodríguez*, José C. Ortiz-Bayliss*, Alejandro Rosales-Pérez*,
Ivan M. Amaya-Contreras*, Santiago E. Conant-Pablos*, Hugo Terashima-Marín*, Carlos A. Coello Coello†
* School of Engineering and Sciences, Tecnologico de Monterrey, Monterrey, Nuevo Leon, 64849, Mexico
Email: {aegr82, jcobayliss, arosalesp, iamaya2, sconant, terashima}@itesm.mx
† Evolutionary Computation Group (EVOCINV), CINVESTAV-IPN, Mexico City, 07360, Mexico
Email: ccoello@cs.cinvestav.mx

***Abstract*—Hyper-heuristics have emerged as an important strategy for combining the strengths of different heuristics into a single method. Although hyper-heuristics have been found to be successful in many scenarios, little attention has been paid to the subsets of heuristics that these methods manage and apply. In several cases, heuristics can interfere with each other and can be harmful for the search. Thus, obtaining information about the differences among heuristics, and how they contribute to the search process is very important. The main contribution of this paper is an automatic heuristic-filtering process that allows hyper-heuristics to exclude heuristics that do not contribute to improving the solution. Based on some previous works in feature selection, two methods are proposed that rank heuristics and sequentially select only suitable heuristics in a hyper-heuristic framework. Our experiments over a set of Constraint Satisfaction Problem instances show that a hyper-heuristic with only selected heuristics obtains significantly better results than a hyper-heuristic containing all heuristics, in terms of running times. In addition, the success rate of solving such instances is better for the hyper-heuristic with the suitable heuristics than for the hyper-heuristic without our proposed filtering process.**

I. Introduction

Heuristic-based methods have been successfully adopted to solve complex combinatorial optimization problems. In recent years, hyper-heuristics [1] have emerged as an important strategy for combining the strengths of different heuristics into a single method, with the aim of providing more flexibility for solving a wide variety of problems. Thus, selection hyper-heuristics [1] attempt to learn patterns of different human-designed heuristics to robustly tackle several problems of different domains such as bin packing [2], vehicle routing [3] and the multi-dimensional knapsack problem [4], to mention just a few.

Hyper-heuristics rely on a mechanism that interprets the problem state and decides the most suitable heuristic to apply based on such state. Although hyper-heuristics have been found to be successful in many scenarios, little attention has been paid to the subsets of heuristics that these methods manage and apply. In most hyper-heuristic related works, the set of heuristics is empirically chosen without a formal justification. The most common criterion for including a heuristic in the hyper-heuristic process is based on the isolated performance of such heuristics. The general assumption is that one heuristic that solves 'well' –given a certain metric of performance– a set on instances, is expected to contribute to the hyper-heuristic process. In other words, we assume that two 'good' heuristics in isolation will benefit from working together; unfortunately, this is not always the case. It is reported that, in several cases, heuristics can interfere with each other and be harmful for the search [5]–[8]. Obtaining information about the differences between heuristics, and how they contribute to the search process is an important topic with many applications in the field of optimization and, particularly, to the hyper-heuristics community.

The main contribution of our work is the proposal of an automatic heuristic-filtering process that allows hyper-heuristics to exclude heuristics that do not contribute to improving the solution process, in such a way that such a process is optimized. Based on some works on feature selection [9], we propose two methods for heuristic selection. These methods rank the heuristics and sequentially select only suitable heuristics for a hyper-heuristic framework. In this paper, we use an evolutionary-based hyper-heuristic generation model [10] to produce hyper-heuristics for Constraint Satisfaction Problems (CSP) [10], but our proposal can be naturally extended to other domains.

The remainder of this paper is organized as follows. We present the previous related work in Section II. Section III describes the two heuristic-filtering strategies that we propose. Our experimental results are shown in Section IV. Finally, our conclusions and some possible paths for future research are presented in Section V.

II. Related Work

Since our goal is to efficiently select only useful heuristics for hyper-heuristics, in this section we briefly describe some relevant feature selection methods which are related to our proposal. Additionally, we also describe the hyper-heuristic framework used to solve CSP, because our experiments were conducted over this domain.

A. Feature Selection Methods

In the specialized literature, we can find several techniques to address the problem of reducing irrelevant and redundant features in challenging tasks [9]. In machine learning [11], the focus of feature selection is to select a subset of features from the inputs which can efficiently describe the data while reducing effects from noise or irrelevant features, and still provide good prediction results [12]. We based our proposal on these techniques, with the aim of selecting suitable heuristics.

Some feature selection methods use feature ranking techniques as the principal criteria for feature selection by ordering. One of the simplest criteria adopts Pearson's correlation coefficient [13]. In this method, all the features with a correlation greater than a predefined threshold are removed from the data before the prediction process starts. Similar methods are based on Mutual Information [14], Conditional Mutual Information [15], and statistical tests [14].

Other feature selection methods, called Wrapper methods, depend on classification results for obtaining a useful feature subset. The Wrapper methods are classified in Sequential Selection and Heuristic Search methods. The sequential selection methods start with an empty set and add features while the prediction results are improved. For example, the Sequential Feature Selection method [16] starts with an empty set and adds one feature for the first step which gives the highest value for the objective function. From the second step onwards, the remaining features are added individually to the current subset if the classification accuracy is improved. On the other hand, the heuristic search methods evaluate different feature subsets to select the one with the best prediction results. For example, the Branch and Bound method [17] uses a tree structure to evaluate different feature subsets; thus, the feature subset with the best classification accuracy in the training phase is selected. Several heuristic search techniques such as Genetic Algorithms [18] and Particle Swarm Optimization [19] are also adopted to select useful features.

In spite of the fact that there are several feature selection methods based on classification results, in practical applications the training phase could be computationally complex, or the class labels can be unknown. In these cases, various feature selection methods based on unsupervised learning have been proposed [20]. In addition, semi-supervised learning is another class wherein both labeled and unlabeled data are used for selecting features [21]. Finally, ensemble feature selection [22] is a relatively new technique used to obtain a stable feature subset. A single feature selection algorithm is run on different subsets of data samples obtained from the bootstrapping method. The results are aggregated to obtain a final feature subset.

The use of feature selection methods improves the results of several tasks, mainly related to learning. Feature ranking techniques and sequential selection methods obtain good results, in spite of the fact that they are computationally simple. Since the methods based on classification results repeatedly train and test a classifier to select useful features, most of them are computationally complex, and their results can be difficult to understand. Additionally, they can be impractical in real world applications. Thus, in this paper, we propose two heuristic filtering methods similar to some techniques used for feature selection. The proposed filtering methods are based on ranking the heuristics, and they sequentially select useful heuristics for hyper-heuristics.

B. *Hyper-heuristic for CSP*

Even when the methods proposed in this paper can be applied to different domains, here we apply them to solve CSP as our means to validate them. We selected this problem mainly because of its many practical applications [23], [24]. Before describing our proposal, we briefly describe the framework adopted to solve CSPs.

The CSPs considered for this research are defined by a set of variables and the constraints among them. Each variable can be assigned a value from a finite domain. To solve a CSP, we are requested to assign a value to every variable in the problem such that their values satisfy all the constraints [25]. 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 continues the search from there.

At each node, the variable to assign, as well as the value used for the assignment, are usually selected by heuristics. In this work, we have included five different heuristics: DOM, DEG, KAPPA, WDEG, and DOMDEG. A description of these heuristics appears in [10]. Once a variable has been selected by using any of the previously heuristics, the first available value in the domain of the selected variable is assigned to such variable.

In our experiments, we use the hyper-heuristic described in [10] for solving CSP. This hyper-heuristic decides the best heuristic to apply in each decision point of the solution search, based on the expected performance of such heuristic on the current problem state. In order to estimate the performance of the heuristics on new instances, a learning process is conducted on a set of training instances, using the heuristics' running time as metric of their performance. Thus, the heuristic with the smallest expected running time for the current problem state should be applied (the most suitable to solve the remaining instance starting at this point).

The learning process runs a Genetic Algorithm for generating a set of rules between the characteristics of the training instances and their corresponding heuristic with the lowest running time. The ideal result for the hyper-heuristic is to select, based on these rules, the most suitable heuristic for each problem instance in the test set (the *oracle*). However, some heuristics are irrelevant, or noise, and they hinder the training process. In addition, the training phase of the Genetic Algorithm can be very slow if we consider all the heuristics. Thus, selecting suitable heuristics is necessary for obtaining good solutions in the testing set.

III. Heuristic Filtering Methods

Based on the core ideas from feature selection, we propose two ranking methods for selecting suitable heuristics. The first one, measures the correlation and the gain ratio between heuristics. The second is based on a statistical test.

Heuristic filtering is performed in a sequential manner. First, the heuristic with the lowest average of running times in the training set is selected. Then, step by step, the rest of suitable heuristics are selected according to the restrictions of the proposed methods. The details of these methods are given below.

A. Heuristic filtering based on Gain and Correlation

The first proposed method for heuristic filtering is based on two frequently used criteria in feature selection: (1) gain and (2) correlation. Once the best heuristic is selected, the next heuristics are selected if their gain and correlation fulfill the thresholds determined by the user (in this paper, these parameters are empirically determined).

We measure the **gain** (see Eq. (1)) as the difference between the sum of the minimum running time values for the previously selected heuristics, and the sum of these values adding the new heuristic.

$$Gain(x,y) = \bar{x} - \bar{y} \tag{1}$$

In Eq. (1), $\bar{x}$ and $\bar{y}$ are the mean of the execution times for the selected heuristics and the added heuristic, respectively. All the heuristics (y) having this difference greater than a certain threshold (5% of the average of running times for the best heuristic) is considered as a candidate heuristic. Then, we analyze their **correlations** with the selected heuristics.

All the new heuristics with a Pearson Correlation (see Eq. (2)) in the range from -0.75 to 0.75 with all the selected heuristics, are also considered as candidates, because they are not "strongly correlated" with any other of the selected heuristics. The candidate heuristic with the highest value of gain is selected, and the process is repeated, step by step, for the rest of the non-selected heuristics until no heuristic can be added.

$$Pearson(x,y) = \frac{n\sum xy - \sum x \sum y}{\sqrt{\left[n\sum x^2 - \left(\sum x\right)^2\right]\left[n\sum y^2 - \left(\sum y\right)^2\right]}} \tag{2}$$

In Eq. (2), n is the number of instances; and x and y are the minimum running times for the previous selected heuristics and the addition of the new heuristic, respectively.

B. Heuristic filtering based on Z-score

A drawback of the previous method is the selection of correct thresholds. Hence, we propose another method for heuristic filtering based on a statistical test. The second method also follows the idea that each selected heuristic brings new minimum running times for the hyper-heuristic, and the new average of the running times must be smaller than the previous average. This new method measures if the average differences are statistically significant based on the **Z-score** [26] test (see Eq. (3)). We used the formula for Z-score Two Sample One Tailed test because the new vector of times will always be smaller than the vector with the times of the previously selected heuristics.

$$Z\text{-}score(x,y) = \frac{\bar{x} - \bar{y}}{\sqrt{\frac{\sigma_x^2}{n} + \frac{\sigma_y^2}{n}}} \tag{3}$$

In Eq. (3), $\bar{x}$ and $\bar{y}$ are the mean of the execution times for the selected heuristics and the added heuristic, respectively; σ_x^2 and σ_x^2 are their standard deviations; and n is the number of instances.

Since we perform the Z-score test for two samples with one tailed, all heuristics which the value of Eq. (3) greater than 1.65 (95 percent of confidence) are considered as candidates. Then, the heuristic with the highest value of Z-score is selected. This process is repeated, step by step, for all the non-selected heuristics until a heuristic can no longer be added.

C. Automatic heuristic filtering

For automatic heuristic filtering, we propose its general steps in Algorithm 1.

Algorithm 1 AHF: Automatic Heuristic Filtering.

```
Require: T: dataset, H: heuristics
Ensure: S: selected heuristics
 1: Compute the average of running times for all heuristics
 2: Select the heuristic with the highest average of running times
 3: Create a reference set with the obtained running times
 4: repeat
 5:     for all non-selected heuristics do
 6:         Create a new set of running times by adding to the reference set the smallest values provided by the non-selected heuristic
 7:         Compare both sets of running times, using the proposed methods, in order to determine if the non-selected heuristic is a feasible candidate
 8:     end for
 9:     If there are candidate heuristics, select the best one, and update the reference set of running times
10: until a heuristic can no longer be selected
11: return S
```

By using these automatic filtering methods, we can obtain a subset of suitable heuristics that allow us improving the performance of our hyper-heuristics. With the next experiments over CSP data, we validate our proposal.

IV. Experimental Results

In this section, we first describe the setup of the experiments. Second, we show the results obtained for the proposed methods in the selection of suitable heuristics according to the training set. Third, we compare the results obtained for the hyper-heuristic using all the heuristics, and using only the suitable heuristics, in the test set. Fourth, we compare our proposal with each one of the simple heuristics. Finally, we discuss the obtained results.

A. Experimental Setup

We generated the hyper-heuristics following the methodology described in [10] and using the set of instances for this work. The latter was splitted into two halves, one for training and one for testing. In both cases, we stopped each heuristic runs in an instance if the running time was greater than 20000 milliseconds. In such cases, we considered that the instance could not be properly solved by such heuristic.

B. Results of the filtering method based on Gain and Correlation in the training set

Prior to running a hyper-heuristic, we first ran all heuristics with the training dataset. Table I shows the average of running times for each heuristic in the training dataset. The KAPPA heuristic outperformed all the other heuristics, exhibiting an average time of 5516.64 milliseconds. Thus, KAPPA was the first heuristic selected for our methods.

TABLE I: Average of running times for each heuristic in the training dataset.

Heuristic	Runtime (ms)
DOM	6030.80
DEG	14397.37
KAPPA	5516.64
WDEG	8199.51
DOMDEG	7970.12

Once the first heuristic has been selected, we analyzed the remaining heuristics. Table II shows the correlation across all heuristics, according to Eq. (2), in order to determine which is a proper candidate. Since no combination of heuristics yields a correlation out of the range of -0.75 to 0.75, all of them are feasible candidates. Thus, we measured the effect of adding them to KAPPA.

For all the instances in the training set, we measured the minimum of running times between KAPPA and the rest of heuristics, one at a time. The newly generated values are considered as a new reference set. The differences between them and the average of running times for KAPPA are considered as the gain of selecting each heuristic (Table III).

TABLE II: Pearson Correlation across all heuristics.

-	DOM	DEG	KAPPA	WDEG	DOMDEG
DOM	1.00	0.23	0.05	0.55	0.73
DEG	0.23	1.00	0.38	0.29	0.13
KAPPA	0.05	0.38	1.00	-0.02	-0.06
WDEG	0.55	0.29	-0.02	1.00	0.60
DOMDEG	0.73	0.13	-0.06	0.60	1.00

TABLE III: Gain from adding each heuristic to KAPPA.

heuristics	Gain
KAPPA&DOM	3520.00
KAPPA&DEG	112.34
KAPPA&WDEG	3181.10
KAPPA&DOMDEG	3419.29

Joining DOM and KAPPA yields the highest gain. 5% of the average of running times for KAPPA is 275.83 milliseconds, and the gain of joining DOM (3520.00 millisecond) is greater than this threshold; thus, we selected DOM as another suitable heuristic.

The process of selecting suitable heuristics continues until no heuristic can be added. Table IV shows the new gain values of KAPPA&DOM with the rest of the non-selected heuristics. Non of these gains fulfill the gain threshold, so no more heuristics are selected[1].

TABLE IV: Gain values of adding the rest of the heuristics to KAPPA&DOM.

heuristics	Gain
KAPPA&DOM&DEG	14.25
KAPPA&DOM&WDEG	117.74
KAPPA&DOM&DOMDEG	22.68

According to gain and correlation, we select KAPPA and DOM as the suitable heuristics for our hyper-heuristic.

C. Results of the filtering method based on Z-score in the training set

We perform the same analysis, but using the Z-score test. The heuristic filtering method based on Z-score is also initialized by selecting the heuristic with the smallest average of running times, i.e. KAPPA. Then, we evaluate the Z-score test, according to Eq. (3), between KAPPA and the remaining heuristics (Table V). The highest Z-score is 5.79, resulting from the union of KAPPA and DOM. Since this value is greater than 1.65 (95% of confidence), we also selected DOM as a suitable heuristic.

TABLE V: Z-score values of adding each heuristic to KAPPA.

heuristics	Z-score
KAPPA&DOM	5.79
KAPPA&DEG	0.15
KAPPA&WDEG	5.24
KAPPA&DOMDEG	5.58

[1]Note that, at this point, adding the DOMDEG heuristic has a gain value smaller than the one produced by adding WDEG; this is because DOMDEG has a high correlation with DOM.

In the next step, new vectors are created for the remaining heuristics (Table VI). None of the new Z-score values is greater than 1.65, hence adding a new heuristic to KAPPA&DOM cannot add a statistically significant difference to the final result. According to the heuristic filtering method based on Z-score, we also select KAPPA and DOM as the only suitable heuristic for our hyper-heuristic.

TABLE VI: Z-score values of adding each heuristic to KAPPA&DOM.

heuristics	Z-score
KAPPA&DOM&DEG	0.03
KAPPA&DOM&WDEG	0.28
KAPPA&DOM&DOMDEG	0.05

D. Comparison between the hyper-heuristics with KAPPA and DOM, and with all heuristics

With the selected heuristics KAPPA and DOM, we trained the hyper-heuristic in the training dataset, and a set of rules was generated in order to use it in the test dataset. To validate our approach, we compared the results of the hyper-heuristic in the test dataset trained with the five initial heuristics with respect to the results of the hyper-heuristic trained with the two selected heuristics. Each hyper-heuristic predicts, for each instance, one heuristic to solve the instance, and we capture the running time of such heuristic solving the instance. Then, the vector of running times for the different hyper-heuristics were also compared.

We ran three times the Genetic Algorithm to reduce random effects. In Table (VII), we show the average of running times for these three runs of both hyper-heuristics. In the three cases, the average of running times for the hyper-heuristic with KAPPA and DOM are smaller than those of the other hyper-heuristic.

TABLE VII: Average of running times of the three runs for the hyper-heuristics with five and two heuristics, respectively.

Runs	HHw5	HHw2
Run1	7972.51	3105.39
Run2	7220.88	3000.59
Run3	5964.96	2799.60

Two new running time vectors are created from calculating, for each hyper-heuristic, the average of running times of each instance in the three generated runs. Fig. 1 shows, in logarithmic scale, these new vectors of running times, sorted in ascending order according to the hyper-heuristic with five heuristics. In Fig. 1, we can see that, in a few cases, the hyper-heurisitc with KAPPA and DOM has greater running times than the hyper-heuristic with all heuristics. In most cases, the hyper-heuristic with KAPPA and DOM obtains smaller running times. When the hyper-heuristic with all heuristics obtains running times greater than 20000 milliseconds, the running times for the hyper-heuristic with KAPPA and DOM are significantly smaller. Moreover, according to the Success Rate (percent of instances solved), the hyper-heuristic with KAPPA and DOM solved 95% of the testing instances, while the hyper-heuristic with all heuristics only solved 80% of such instances.

The BoxPlots for these vectors of running times appear in Fig. 2. In this figure, we can also note that, except for some outliers, the maximum running time for the hyper-heuristic with KAPPA and DOM is considerably smaller than the maximum running time of the hyper-heuristic with all heuristics. Additionally, we can see than the median of running times for the hyper-heuristic with KAPPA and DOM is smaller than the median of the other hyper-heuristic.

In order to know if the differences between the two hyper-heuristics are statistically significant, we performed a Wilcoxon Signed-Rank Test with $\alpha = 0.05$, according to [27]. The p-value for the test is 0.001, thus, the differences between both hyper-heurisitcs are statistically significant.

E. Comparison between each heuristic and the hyper-heuristic with KAPPA and DOM

A comparison between the hyper-heuristic with KAPPA and DOM, and each heuristic, is also performed. The BoxPlots for the vectors with the average of running times for the hyper-heuristic, and running times for each heuristic, appear in Fig. 3. In this figure, the high running time values for DEG heuristic are relevant. On the other hand, the median of the running times for the hyper-heuristic, and the 50% of their running times, are significantly smaller than the running times of the heuristics.

Additionally, we perform a Friedman Aligned Test with $\alpha = 0.05$, suggested in [27] for comparing the algorithms. The ranking values for the test appear in Table VIII. The p-value of the test is equal to 9.5e-11; hence, the differences between the hyper-heuristic with KAPPA and DOM, and the simple heuristics, are statistically significant.

TABLE VIII: Average rankings for our proposal and each heuristic (Freedman Aligned Test).

Algorithm	Ranking
HHw2	433.72
DOM	628.79
DEG	1107.18
KAPPA	585.12
WDEG	808.49
DOMDEG	777.70

Since the differences among all the algorithms compared are statistically significant, we performed a Post Hoc comparison, also suggested in [27]. The goal is to determine which of the simple heuristics, separately, have statistically significant differences with the hyper-heuristic. Table IX shows the results of applying the Holm procedure with $\alpha = 0.05$. In all cases, p is less or equal than the adjusted α according to Holm. Thus, the hyper-heuristic

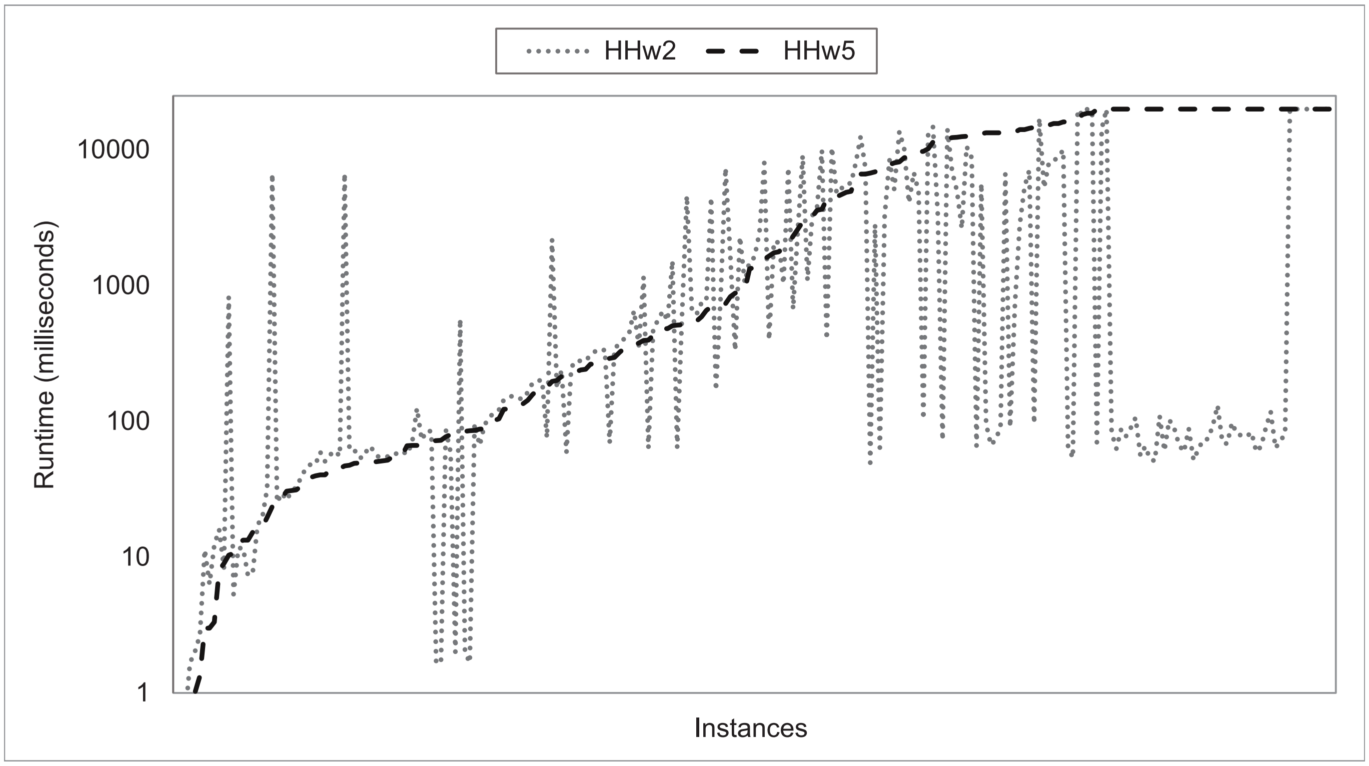

Fig. 1: Average of running times for the hyper-heuristics with two and five heuristics.

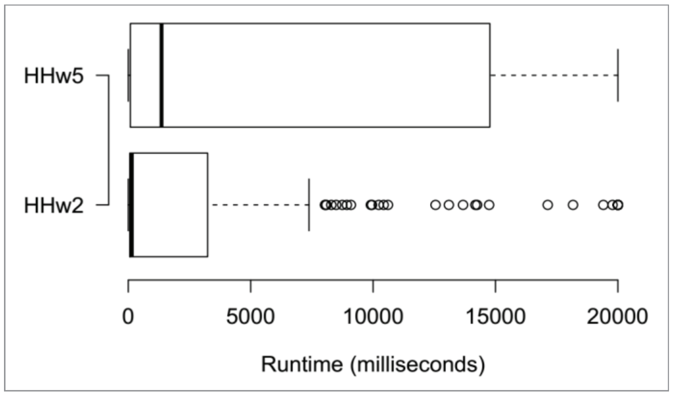

Fig. 2: Boxplots of the average of running times for the hyper-heuristics with two and five heuristics.

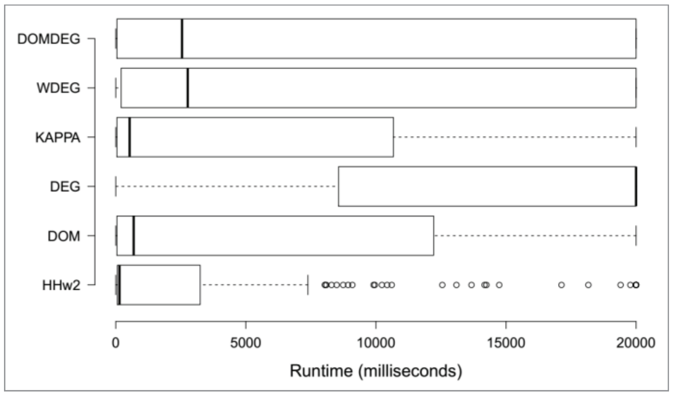

Fig. 3: Boxplots of the running times for each heuristic and the hyper-heuristics with KAPPA and DOM.

has differences statistically significant with respect to each of the simple heuristics.

TABLE IX: Pos Hoc comparison for $\alpha = 0.05$.

Algorithm	p	Holm
DEG	0	0.01
WDEG	0	0.0125
DOMDEG	0	0.016667
DOM	0	0.025
KAPPA	6.9e-5	0.05

F. Discussion

Building a hyper-heuristic with only suitable heuristics contributes to yield better results. In our experiments, the hyper-heuristic with KAPPA and DOM obtained better performance results than the hyper-heuristic with all heuristics, and also than each heuristic considered independently. Moreover, our hyper-heuristic obtained a better Success Rate. One of the advantages of our proposal is that the training phase has a smaller computational complexity than training the hyper-heuristic with all heuristics. As another advantage, the set of generated rules with only two heuristics are more understandable for users.

V. Conclusions and Future Work

Automatic heuristic-filtering allows hyper-heuristics to ease its learning stage by discarding irrelevant (or noise) heuristics that do not contribute significantly to improve the results. Based on some related feature selection methods, this paper proposes two methods for selecting suitable heuristics. Both of them are ranking methods, and they sequentially add one heuristic at each step to the final solution. The first method selects heuristics with a high gain of running times, and small correlations across them. The second method uses the Z-score test to measure if the gain in running times is statistically significant. Although the experiments were conducted over CSP data, the proposed methods can be naturally extended to other optimization domains.

In the experiments, only KAPPA and DOM were selected as suitable heuristics for the hyper-heuristic framework to solve CSP. Trained with these two heuristics, the hyper-heuristic outperformed its results in terms of average of running times for solving the test instances, obtaining differences which are statistically significant. Additionally, the learning stage was faster, while the generated rules were smaller and easier to understand for users. In comparison with each heuristic considered separately, the hyper-heuristic with two selected heuristics also obtained differences which were statistically significant, in terms of running times. As future work, we will focus on extending the obtained results to different optimization domains, and we will test them not only with constructive heuristics, but also with perturbative heuristics.

Acknowledgments

This research was supported in part by CONACyT Basic Science Projects under grants 241461 and 221551, and ITESM Research Group with Strategic Focus in Intelligent Systems.

References

[1] 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.

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

[3] R. J. Marshall, M. Johnston, and M. Zhang, "Developing a hyper-heuristic using grammatical evolution and the capacitated vehicle routing problem," in *Asia-Pacific Conference on Simulated Evolution and Learning.* Springer, 2014, pp. 668–679.

[4] J. H. Drake, E. Özcan, and E. K. Burke, "Modified choice function heuristic selection for the multidimensional knapsack problem," in *Genetic and Evolutionary Computing.* Springer, 2015, pp. 225–234.

[5] G. Ochoa, J. A. Vazquez-Rodriguez, S. Petrovic, and E. Burke, "Dispatching rules for production scheduling: A hyper-heuristic landscape analysis," in *2009 IEEE Congress on Evolutionary Computation*, May 2009, pp. 1873–1880.

[6] J. C. Ortiz-Bayliss, H. Terashima-MarÃŋn, E. ÃŰzcan, A. J. Parkes, and S. E. Conant-Pablos, "Exploring heuristic interactions in constraint satisfaction problems: A closer look at the hyper-heuristic space," in *2013 IEEE Congress on Evolutionary Computation*, June 2013, pp. 3307–3314.

[7] S. Petrovic and S. L. Epstein, "Random subsets support learning a mixture of heuristics," *International Journal on Artificial Intelligence Tools*, pp. 501–520, 2008.

[8] R. J. Wallace, *Analysis of Heuristic Synergies.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 73–87.

[9] G. Chandrashekar and F. Sahin, "A survey on feature selection methods," *Computers & Electrical Engineering*, vol. 40, no. 1, pp. 16–28, 2014.

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

[11] E. Alpaydin, *Introduction to machine learning.* MIT press, 2014.

[12] I. Guyon and A. Elisseeff, "An introduction to variable and feature selection," *Journal of machine learning research*, vol. 3, no. Mar, pp. 1157–1182, 2003.

[13] R. Battiti, "Using mutual information for selecting features in supervised neural net learning," *IEEE Transactions on neural networks*, vol. 5, no. 4, pp. 537–550, 1994.

[14] G. Forman, "An extensive empirical study of feature selection metrics for text classification," *Journal of machine learning research*, vol. 3, no. Mar, pp. 1289–1305, 2003.

[15] F. Fleuret, "Fast binary feature selection with conditional mutual information," *Journal of Machine Learning Research*, vol. 5, no. Nov, pp. 1531–1555, 2004.

[16] P. Pudil, J. Novovičová, and J. Kittler, "Floating search methods in feature selection," *Pattern recognition letters*, vol. 15, no. 11, pp. 1119–1125, 1994.

[17] P. M. Narendra and K. Fukunaga, "A branch and bound algorithm for feature subset selection," *IEEE Transactions on Computers*, vol. 26, no. 9, pp. 917–922, 1977.

[18] D. E. Goldberg and J. H. Holland, "Genetic algorithms and machine learning," *Machine learning*, vol. 3, no. 2, pp. 95–99, 1988.

[19] J. Kennedy, "Particle swarm optimization," in *Encyclopedia of machine learning.* Springer, 2011, pp. 760–766.

[20] M. H. Law, M. A. Figueiredo, and A. K. Jain, "Simultaneous feature selection and clustering using mixture models," *IEEE transactions on pattern analysis and machine intelligence*, vol. 26, no. 9, pp. 1154–1166, 2004.

[21] Z. Zhao and H. Liu, "Semi-supervised feature selection via spectral analysis," in *Proceedings of the 2007 SIAM International Conference on Data Mining.* SIAM, 2007, pp. 641–646.

[22] T. Abeel, T. Helleputte, Y. Van de Peer, P. Dupont, and Y. Saeys, "Robust biomarker identification for cancer diagnosis with ensemble feature selection methods," *Bioinformatics*, vol. 26, no. 3, pp. 392–398, 2010.

[23] 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.

[24] 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.

[25] R. Dechter, "Constraint networks," in *Encyclopedia of Artificial Intelligence.* Wiley, 1992, pp. 276–286.

[26] V. Agarwal and R. J. Taffler, "Twenty-five years of the taffler z-score model: Does it really have predictive ability?" *Accounting and Business Research*, vol. 37, no. 4, pp. 285–300, 2007.

[27] J. Derrac, S. García, D. Molina, and F. Herrera, "A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms," *Swarm and Evolutionary Computation*, vol. 1, no. 1, pp. 3–18, 2011.