Exploring Heuristic Interactions in Constraint Satisfaction Problems: A Closer Look at the Hyper-Heuristic Space

José C. Ortiz-Bayliss*, Hugo Terashima-Marín†, Ender Özcan*,Andrew J. Parkes* and Santiago E. Conant-Pablos†
*Automated Scheduling, Optimisation and Planning (ASAP)
School of Computer Science, University of Nottingham, UK
Email: {Jose.Ortiz_Bayliss, Ender.Ozcan, Andrew.Parkes}@nottingham.ac.uk
†Tecnológico de Monterrey
Monterrey, Mexico
Email: {terashima, sconant}@itesm.mx

***Abstract*—Variable ordering has been a recurrent topic of study in the field of constraint satisfaction because of its impact in the cost of the search. Various variable ordering heuristics have been proposed to help guiding the search under different situations. One important direction of the study about variable ordering is the use of distinct heuristics as the search progresses to reduce the cost of the search. Even though the idea of combining heuristics goes back to the 60's, only a few works that study which heuristics to use and how they interact with each other have been described. In this investigation, we analyse the interactions of four important variable ordering heuristics by combining them through hyper-heuristics that decide the heuristic to apply based on the depth of the nodes in the search tree. The paper does not include any specific model for generating such hyper-heuristics; instead, it presents an analysis of the changes in the cost when different heuristics are applied during the search by using one simple hyper-heuristic representation. The results show that selectively applying distinct heuristics as the search progresses may lead to important reductions in the cost of the search with respect to the performance of the same heuristics used in isolation.**

I. Introduction

A constraint satisfaction problem (CSP) is defined by a set of variables X_1, X_2, $\ldots$, X_n and a set of constraints C_1, C_2, $\ldots$, C_m. Each variable X_i has a nonempty domain D_i of possible values. Each constraint C_i involves some subset of variables and specifies the allowable combinations of values for that subset [1]. CSPs can be solved by local search algorithms that do not guarantee to find a solution [2]; or by complete methods that guarantee to find a solution if at least one exists [3]. We will focus on complete methods that require to expand a search tree to find a solution. Each time a variable has to be instantiated, a heuristic is invoked and as result, the next variable to instantiate is selected (according to the criterion of such heuristic). Because of this, the (variable) ordering heuristic has a tremendous effect in the structure of the search tree, because it decides which variables will be instantiated before the others. For small combinatorial problems, exact methods can be applied and optimal orderings can be found. However, when larger and more complex problems appear, exact solutions are not a reasonable choice since the search space grows exponentially with the number of variables, and so does the time for finding the optimal ordering of variables. Various heuristic and approximation approaches have been proposed that find near optimal solutions. But, it has not been possible to find a reliable heuristic to solve all instances of a given problem. In general, some heuristics work well for particular instances, but not for all of them because each heuristic exploits distinct features (or combinations of features) of the instances under exploration.

The idea of selecting the most suitable solution method for a given problem is usually referred to as the algorithm selection problem [4]. This problem has been addressed in many investigations where distinct solution approaches have been proposed (see for example [5] and [6]. In this investigation we will use the hyper-heuristic terminology to refer to the method that selects among heuristics based on the current problem features [7].

Hyper-heuristics can be divided into two main classes: those which select from existing heuristics and those that generate new heuristics [8]. A more detailed description about the classification of hyper-heuristics can be found in [9], [10]. In this investigation we will focus our attention on selective hyper-heuristics; hyper-heuristics that select from existing heuristics according to the current problem state [11]. The hyper-heuristic should decide when and where to apply each single heuristic, based on some criterion. The choice of heuristics may depend on the features of the problem state, such as expected number of solutions, values on the objective function, elapsed time, etcetera. The hyper-heuristics described in this investigation correspond to selective hyper-heuristics, where the output at each time they are invoked is a variable ordering heuristic.

With respect to CSPs, one of the first attempts to systematically map CSPs to algorithms and heuristics according to the features of the instances was presented by Tsang and Kwan [12]. In that study, Tsang and Kwan presented a survey of algorithms and heuristics for solving CSPs and proposed a relation between the formulation of the CSP and the most adequate solution method for that formulation. Petrovic and Epstein [13], [14] studied the idea of producing mixtures of heuristics that work well on particular classes of instances. Also, algorithm portfolios for constraint programming have been successfully studied before [15] with promising results.

Other studies about the dynamic combination of heuristics applied to CSPs include the work done by Terashima-Marín *et al.* [16], who proposed a genetic algorithm framework to generate hyper-heuristics for variable ordering in CSPs and the research developed by Bittle and Fox [17] who presented a hyper-heuristic approach for variable and value ordering for CSPs based on a symbolic cognitive architecture augmented with case based reasoning. Recent ideas on selecting the most suitable heuristic for a given problem instance include the use of artificial neural networks [18], choice function [19] and autonomous search [20].

The investigation in this article analyses the hyper-heuristic approach for CSPs in a different way with respect to previous works on hyper-heuristics for the same domain. We have focused our interest in analysing the heuristics that compose the hyper-heuristics and their interactions as an attempt to understand how to make them work in collaboration. No specific generation method is discussed in this paper and our representation is as simple as it can be. We do not claim that our hyper-heuristic representation could be used in a real application as it is now, but it certainly allowed us to analyse how heuristics behave when applied at distinct stages of the search. Our analysis is relevant because it explores important aspects of the representation of sequential hyper-heuristics along with some insights about which heuristics work well together.

The paper is organized as follows. Section II describes the problem and presents other works related to this investigation. The variable ordering heuristics used in this work are described in Sec. III. Section IV presents the experiments and main results. Finally, Sec. VI presents the conclusion and the future directions of this investigation.

II. Problem Statement and Related Work

The task of the selective hyper-heuristic is to decide, at each decision point, which one of the available heuristics to apply. In the case of CSPs, the search is performed by using depth first search (DFS) [21]. In the search tree, every time a variable is instantiated, the constraints in which that variable is involved must be checked to verify that none of them is violated; this is known as a consistency check. When an assignment breaks one or more constraints, the instantiation must be undone, and another value must be considered for such variable. If there are no more values available, the value of the previous instantiated variable must be changed; this technique is known as backtracking [22]. There are some improvements to this basic search method which try to reduce the number of revisions of the constraints (consistency checks) like constraint propagation [23] and backjumping [24]. The CSP solver used for this investigation incorporates constraint propagation by using the AC3 algorithm [25] and also implements backjumping.

Each node in the search tree represents the instantiation of one variable. We can consider that each time a node in the tree is to be expanded, a decision point occurs and a heuristic has to be invoked. It is at these decision points where the hyper-heuristic works and decides which heuristic to apply. The hyper-heuristic indirectly decides the order in which the variables are instantiated, because it has no information about how the heuristics work. In the simplest form, it also has no access to the problem state, however, it does have access to the position of the node within the search tree and so can be allowed to rely on that information. We will take a scheme in which the decision is allowed to depend on the depth d of search node in the tree, that is, it depends on the number of decision points made to reach that node. We will also take that it is done in a static static fashion; that is, a fixed heuristic is picked at each value of the depth. For an instance with n variables, then it cannot use a list of decision points larger than n. Hence, the hyper-heuristic is given by a sequence of n low-level heuristics $(h_0, h_1, \ldots, h_{n-1})$ with depth d always corresponding to the use of heuristic h_d.

Given a set of k heuristics and a CSP instance with n variables, the maximum number of sequences of heuristics that can be formed is k^n, assuming the instance is satisfiable (one of the k heuristics per decision point in the instance). Of course, because of propagation or pruning due to reaching a contradiction, then the depth achieved will be usually smaller than n. It is important to stress that these sequences are not affected by the backtrack movements during the search. If the search fails at certain point and it has to go back to a previously visited node to change the value of one variable, the heuristic to use will depend again on the depth of the node in the search tree and the same heuristic that was used before will be used this time for the same node. Of course, the heuristic will be the same but the selected variable may be different because the properties of the instance have already changed.

The analysis described in this investigation is related to some studies about the hyper-heuristic landscape and heuristic synergies. To the authors' best knowledge, the first work about the analysis of the hyper-heuristic landscape was presented only a few years ago by Vázquez *et al.* for the hybrid flow shop scheduling problem [26]. Their work introduced important concepts in the topic, like the associated space for hyper-heuristics (the heuristic space) and how they are related to the solution space. Later, Ochoa *et al.* [27], [28] described a landscape analysis technique which provides means for understanding the influence of operators and algorithmic behaviour for a given problem. Also, Maden and Özcan analysed a set of perturbative hyper-heuristics through landscape analysis based on an auto-correlation function on various benchmark functions [29].

With respect to the interactions of heuristics, Wallace [30] conducted an analysis about the search on CSPs when the decision of the next variable to instantiate is guided by a ranking given by various heuristics applied at the same time. Wallace used the term 'heuristic synergies' to refer to the phenomenon that the combination of heuristics produces results which are better than the heuristics applied in isolation. Among the most relevant findings in his investigation is the evidence that it is not enough to combine heuristics to improve the search, but the combination must have some properties because not all the heuristics work well together.

III. Variable Ordering Heuristics

Four variable ordering Heuristics were used in this investigation: Min-domain [31], [32], Max-conflicts, dom/deg [31], [33] and kappa [34]. Each one of these heuristics works under the principle of 'fail first', that suggests to select first the variables which are more likely to fail. If the instantiation of those variables is successful, then it is expected that the remaining subproblem will be easier to solve. Each heuristic works as follows:

Min-domain. This heuristic prefers the variable with the smaller domain size. Then, it assumes that the variable with the fewer values in its domain is more likely to fail [35].

Max-Conflicts. Max-conflicts uses the criterion of the number of conflicts to decide the next variable to instantiate. Max-conflicts tries first the variable involved in the fewer conflicts (restricted pairs of values).

dom/deg. This heuristic combines two criteria to decide which heuristic to instantiate first. It considers both the domain size and the forward degree of the variables. The forward degree of a variable is the number of uninstantiated variables connected to it. Then, dom/deg selects first the variable that maximizes the quotient of the domain size over the forward degree of the variable.

kappa. The kappa heuristic uses the value of κ as a measure of the difficulty of the remaining instance [34]. Then, it selects the variable which instantiation maximizes:

$$\kappa(x) = \frac{-\sum_{c_i \in C_x} \log_2(1-p_{c_i})}{\log_2(m_x)} \tag{1}$$

where c_i is a constraint where variable x is involved, m_x is the domain size of variable x and p_{c_i} is the fraction of unfeasible tuples on constraint c_i.

Along with these four variable ordering heuristics, to order the values of the selected variable we used Min-conflicts [2], a heuristic that orders the values in an ascendant way according to the number of conflicts where each value in the domain of the selected variable is involved. In this way, Min-conflicts tries first those values which are more likely to lead to one solution, because they are less restricted.

In all cases, ties are broken by using the lexical ordering.

IV. Experiments

We created a set of 300 random instances with 20 variables, each variable with 10 values in its domain. The number of constraints and the number of conflicts within each constraint was decided randomly for each instance. All the random instances were generated with model B [36]. With model B, CSP instances are generated in two stages. In the first stage, a constraint graph G with n nodes is randomly constructed and then, in the second stage, the incompatibility graph C is formed by randomly selecting a set of edges (incompatible pairs of values) for each edge (constraint) in G. The parameter p_1 determines how many constraints exist in a CSP instance and it is called constraint density, whereas p_2 determines how restrictive the constraints are and it is called constraint tightness. For the purpose of this investigation, using only one generator is enough to develop the ideas that two or more heuristics can be applied at distinct times during the search and reduce its cost.

The set of 300 random instances will be referred to as set A in the rest of the investigation. It is important to stress that set A contains a mixture of instances, both satisfiable and unsatisfiable. Regarding the hardness of the instances, set A contains both hard and easy ones according to their location with respect to the transition phase [37]. Thus, some of the instances are solved with just a few consistency checks, while others will require considerably more effort. The number of consistency checks is used in this investigation as a reference to measure the cost of the search. Every time a constraint has to be revised, a constraint check occurs. Thus, the fewer the consistency checks, the lower the cost of the search. For the first set of experiments we considered the evaluation of the variable ordering heuristics in pairs: (Min-domain, Max-conflicts), (Min-domain, dom/deg), (Min-domain, kappa), (Max-conflicts, dom/deg), (Max-Conflicts, kappa) and (dom/deg, kappa).

A. *Hyper-heuristic Representation*

With selective hyper-heuristics we believe that applying distinct heuristics at different levels of the search tree may achieve a better performance than with the use of one single heuristic during all the search. Evidence has been gathered before that proves that some heuristics are better than others under certain regions of the space of instances [38]. Then, it seems reasonable to think that we can make two or more heuristics to collaborate to improve the performance of the search. When combining heuristics during the search by applying distinct heuristics among the decision points of the search tree, three scenarios can result from this combination. Then the hyper-heuristic may: (1) reduce the cost of the search, leave unaffected the cost of the search or (3) increase the cost of the search. In this investigation, the cost of the search is defined as the number of consistency checks required for a given method to solve an instance. We expect that good combinations of heuristics often reduce the cost of the search.

The space of heuristics is, the associated space where hyper-heuristics work [26]. Hyper-heuristics look for combinations of heuristics and by doing so, indirectly facilitate finding a solution to a CSP. The size of the heuristic space depends on the number of heuristics and the number of decision points. As we mentioned before, the number of variables imposes a maximum number of decision points during the search. Then, given a hyper-heuristic that selects among k heuristics over instances with up to n variables, the size of the heuristic space is up to k^n states. Because of its size, we cannot exhaustively search the heuristic space for large instances, like most of the real problems are. Also, one specific sequence of heuristics may be efficient for one particular instance but perform poorly on others. Then, even though we could completely explore the heuristic space looking for the best sequence, that sequence is unlikely to work well on all the instances.

From this point on, we will use the term hyper-heuristic to refer to selective ones only.

B. *Combining Heuristics*

A variable ordering heuristic is the immediate output of a hyper-heuristic. As a result of multiple calls to the hyper-heuristic (at different decision points of the search), a sequence of heuristics is obtained. Then, we can analyse the outputs of any hyper-heuristic by indirectly analysing the sequences they produce. We will analyse the sequences of heuristics that can be produced by different sets of heuristics. The simpler case of study occurs when only two heuristics are involved. In this case, the sequences of heuristics can be represented as binary strings, where the most significant bit indicates the first heuristic to apply during the search. If during the path in the search tree to reach a node, the number of decision points exceeds the length of the string, the sequence is repeated until the instance is solved or proved unsatisfiable. For example, suppose we have a sequence of heuristics defined by the string 001101 and a set of heuristics H = {Min-domain, Max-Conflicts}. During the search, the first two decision points will use Min-domain, followed by two decision points where Max-Conflicts will be used. The fifth and sixth decision points will be guided by Min-domain and Max-conflicts, respectively. In case the search continues, the sequence will be repeated until the instance is solved or proved to be unsatisfiable.

In this experiment, we explored the 8-bit heuristic space for set A. Then, 256 hyper-heuristics were generated and their performance evaluated on set A. This number of hyper-heuristics cover the whole 8-bit heuristic space, allowing us to map every permutation of heuristics to one cost. Even though we are exhaustively exploring an 8-bit heuristic space representation, it is not the only way to code sequences of heuristics for CSPs. For example, we may use any α-bit representation ($\alpha > 8$) and the results may be slightly different. The most important consideration for deciding the resolution of the search space in this investigation is the time needed to solve the instances. For example, if we have instances with 20 variables, a sequence of 20 bits is needed to represent all the possible sequences of heuristics that can be formed. 20 bits correspond to 1048576 sequences per instance, which require significantly more time to run than the sequences we have used for this experiment.

With the cost associated to every point in the heuristic space, we can produce a graphical representation of the utility function under exploration. Because we have used binary code to represent the sequences of heuristics, each number in the x-axis corresponds to its equivalent binary-coded value. For example, value 200 in the x-axis corresponds to the sequence 11001000, which should be interpreted as explained before. It is possible that some other conversion from the sequence to an x-axis value would clarify the results and this will be explored in future work.

1) (Min-domain, Max-Conflicts): Figure 1 presents the cost of each of the 256 sequences of heuristics resulting from the combination of Min-domain and Max-conflicts on set A. The cost of solving set A is defined as the sum of the consistency checks required to solve each instance in the set. On set A, Min-domain has a lower cost than Max-conflicts. Some of the sequences produce a cost which is between the cost of these two heuristics: these sequences are better than Max-conflicts but not better than Min-domain. 14.84% of the sequences represent larger costs than any of the single heuristics. Finally, 12.11% of the sequences reduces the cost of both Min-domain and Max-conflicts. For this pair of heuristics, 01111000 was the best sequence of heuristics (0 for Min-domain, 1 for Max-conflicts). The cost of this sequence represents a saving of 3.49% in the cost of solving set A with respect to Min-domain, that produced the lowest cost from both heuristics on this set.

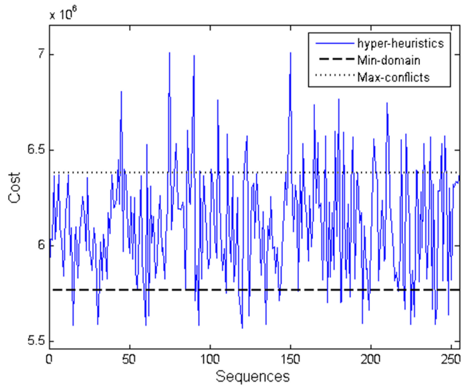

Fig. 1: Analysis of the 8-bit heuristic space (Min-domain, Max-Conflicts) on set A

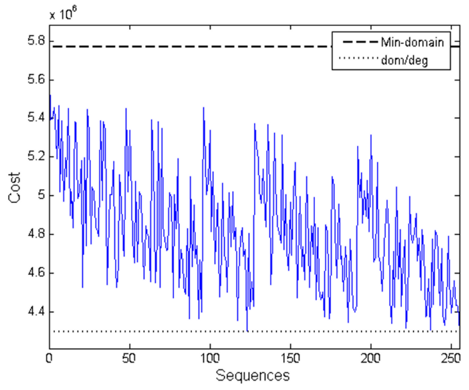

Fig. 2: Analysis of the 8-bit heuristic space (Min-domain, dom/deg) on set A

2) (Min-domain, dom/deg): This case is completely different to the first case described. Figure 2 shows that, in the 8-bit heuristic space, it is not possible to combine Min-domain with dom/deg through an 8-bit length sequence to improve the performance of dom/deg on set A. But, all the sequences of heuristics produce a cost below the one of Min-domain. We can say that dom/deg *improves* Min-domain.

3) (Min-domain, kappa): In the case of the pair (Min-domain, kappa) (Fig. 3), 2.34% of the sequences produced represent a lower cost than kappa on this set, which was the best heuristic for this pair. The sequence 01101011 obtains the lowest cost (0 for Min-domain, 1 for kappa) for this pair of heuristics. This represents a saving of 3.77% with respect to kappa. No cases where the combination of these heuristics resulted in a cost larger than the cost of Min-domain, which is used as upper-bound for this pair of heuristics. Thus. we can also conclude that kappa improves dom/deg.

4) (Max-Conflicts, dom/deg): When we combined Max-conflicts and dom/deg (Fig. 4), we observed that it was not possible to obtain a sequence of heuristics with a better performance than dom/deg. The same phenomena was observed in the pair Max-conflicts and kappa (Fig. 5). Because in both cases the cost of Max-conflicts was never increased by any of the sequences, we can conclude that both dom/deg and kappa

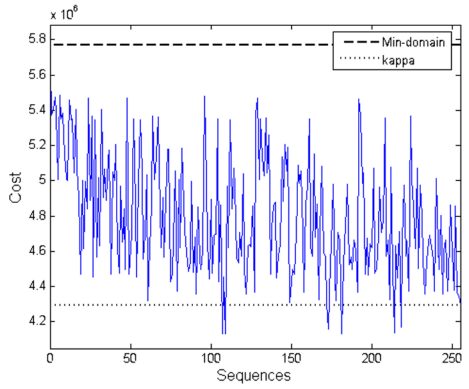

Fig. 3: Analysis of the 8-bit heuristic space (Min-domain, kappa) on Set A

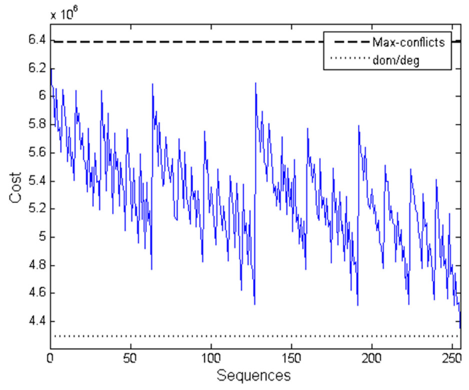

Fig. 4: Analysis of the 8-bit heuristic space (Max-Conflicts, dom/deg) on Set A

improve Max-conflicts.

5) (dom/deg, kappa): A very particular case occurs for the pair (dom/deg, kappa) in Figure 5. Both heuristics show a similar performance, being their costs practically equal. Nevertheless, when these heuristics are used together in a sequence, most of the times they produce reductions in the cost

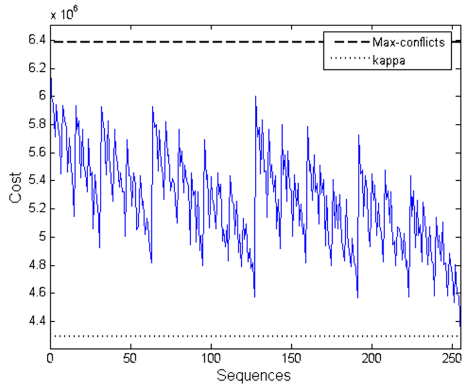

Fig. 5: Analysis of the 8-bit heuristic space (Max-Conflicts, kappa) on set A

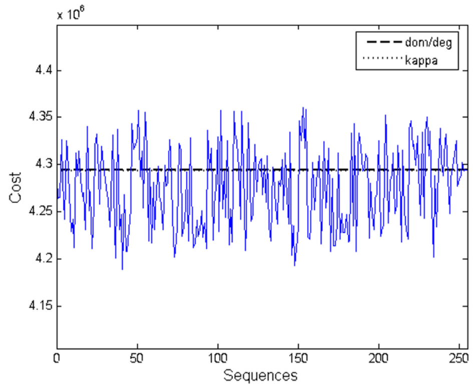

Fig. 6: Analysis of the 8-bit heuristic space (dom/deg, kappa) on set A

of solving set A. In our experiment, 62.11% of the sequences of heuristics resulted in a reduction of the cost with respect to kappa, which produced the lowest cost for the two heuristics. Only 36.72% of the sequences resulted in an increase in the cost with respect to the upper bound, given by dom/deg. The best sequence, 00101001 (0 for dom/deg, 1 for kappa), produced a reduction of 2.43% in the cost with respect to the lower bound (that corresponds to the cost of kappa).

Even though these heuristics generated very similar orderings when applied in isolation on set A (45.33% of the orderings were exactly the same for both heuristics), when combined in a sequence, the orderings of the variables change and new behaviours emerge.

V. Analysis of the Heuristic Interactions

On set A, kappa and dom/deg provide the lowest cost with 4293858 and 4294894, respectively. The difference in the performance of these two heuristics is insignificant: the cost of dom/deg is only 0.024% above the cost of Kappa, which is the best heuristic on this set of instances. These heuristics have a very similar performance but the orderings they produce are not exactly the same. On the other hand, Min-domain and Max-conflicts are not competent heuristics on this set. The cost of Min-domain, 5769105, is 34.35% above the cost of kappa. Max-conflicts is the worst heuristic on set A with a cost of 6383514, which represents 48.66% of additional consistency checks with respect to kappa.

Figure 7 shows the box plot with the results of the six pairs of heuristics on set A. We can observe that (dom/deg, kappa) is the pair with the best performance on set A. Also, dom/deg and kappa (which have a similar performance) are a good complement for Min-domain. Max-conflicts is a bad heuristic for this set of instances: it produces the worst results when combined with the other heuristics.

A. Exploiting the Problem State

We have mentioned before that a hyper-heuristic which ignores the problem state and only applies a fixed sequence of heuristics may not be the best option. A more intelligent approach would be to apply a sequence according to the initial problem state of the instance to solve. With this idea on mind,

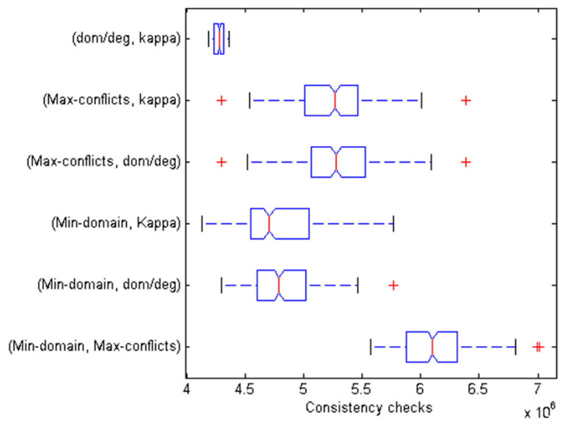

Fig. 7: Box plot with the results of the six pairs of heuristics on Set A

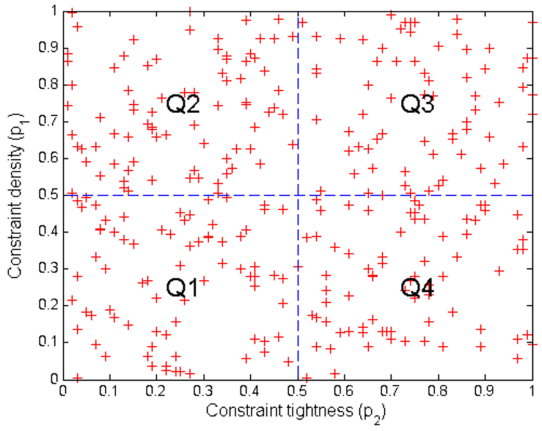

Fig. 8: Instances in set A divided by quadrants

we have slightly modified the hyper-heuristics as follows. Now, the hyper-heuristics contain more than one sequence of heuristics at the time and the selection of the sequence to apply depends on the initial problem state.

This new hyper-heuristic divides the space $p_1 \times p_2$ in four quadrants of the same size. These quadrants, and the instances in set A which correspond to each one of them are shown in Fig 8. Even though these features seem very simple, there is evidence that they can be used to properly map one the problem state to some suitable heuristics [39].

The hyper-heuristic works in two steps: (1) It reads the initial problem state and then, it assigns the instance to one of the four quadrants, (2) Finally, the hyper-heuristic applies the sequence assigned to that quadrant. As we can observe, the hyper-heuristic will produce a sequence of heuristics based on the initial values of p_1 and p_2; it no longer returns a simple sequence that ignores the problem state.

The new hyper-heuristic was applied to the pair (dom/deg, kappa). This was done because this pair provided the best results with the analysis of sequences and we are interested in observing if the the change in the hyper-heuristics can improve the results obtained by the first versions. To produce the sequences for each quadrant we conducted a similar analysis to the one presented in previous sections. Set A was divided into quadrants as described before, and the instances in each quadrant were solved with each of the 256 sequences that can be produced with an 8-bit representation. The best sequence of heuristics from each quadrant was kept to be part of the hyper-heuristic.

Even though the best sequence obtained for the pair (dom/deg, kappa) already produced promising results, we tried the new hyper-heuristic representation (the one that considers the initial problem state) and we were able to improve the results obtained before. The sequences applied per sectors, on set A, produced a cost of 4160582, which represents a reduction of 3.1% with respect to the best heuristic on set A, kappa. The results confirm our idea that using the problem state to discriminate among sequences is a better approach than simply using the same sequence for all the instances. As we expected, some sequences work better for some instances than others.

In this case, we have divided the space $p_1 \times p_2$ in an arbitrary way just to make an example of how the hyper-heuristics can be applied if the initial problem state is used to determine the sequence of heuristics. But more sophisticated methods of machine learning can be used to create more accurate maps between problem state and hyper-heuristics to obtain better results. In this experiment we are only interested in showing that a unique sequence of heuristics, applied to all the instances is unlikely to perform better than different sequences applied according to the features of the instance at hand. At this point we should also emphasize the importance of a good set of features to characterize the space. For the purpose of this investigation p_1 and p_2 seem to work, but for other applications where the instances are more diverse, the same features may not be a good option.

B. *Choosing Among More Heuristics*

So far we have analysed the effect of combining pairs of heuristics. We have identified pairs that work well together and heuristics that seem to be unable to increase the performance of others (for example, Max-conflicts). In this section we will discuss what happens when we increase the number of heuristics the hyper-heuristic can select from during the search.

When we combine the four heuristics, the space representation needs to be adjusted. This time, the sequences will not be represented as binary strings, but strings of base 4. Thus, more sequences can be generated with shorter strings. For example, an 8-digit string of base 4 is enough to represent 65536 distinct sequences of heuristics. For this reason, the length of the strings for this experiment was reduced to a 6-digit representation, which generates 4096 sequences of heuristics.

The best sequence, 233330 (0 for MRV, 2 for dom/deg and 3 for kappa) produces a cost of 3924391, which represents a saving of 8.60% with respect to the best heuristic for the set, kappa. As we can see, Max-conflicts is not part of the best sequence, which confirms our observations with the analysis per pairs of the heuristics. From all the hyper-heuristic produced in the 6-digit space representation, only 0.44% of them were worse than Max-conflicts, while 4.39% of the

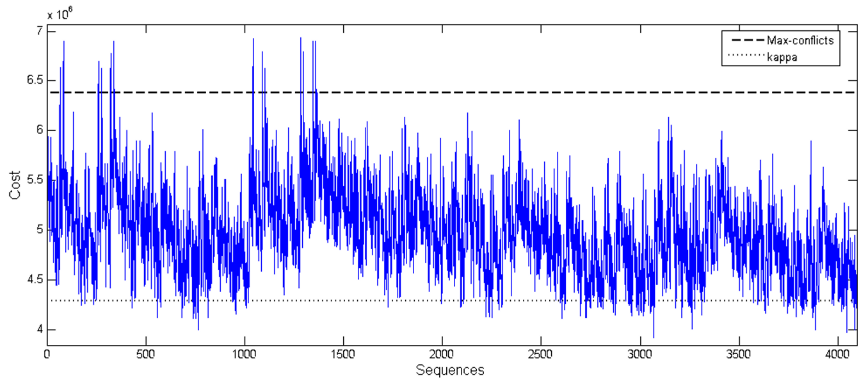

Fig. 9: Analysis of the 6-digit heuristic space (all heuristics) on set A

sequences solved the set with fewer consistency checks than the best single heuristic, kappa. The cost of each one of the 4096 sequences of heuristics is shown in Fig. 9.

VI. Conclusions and Future Work

The results confirm the idea that by combining some heuristics it is possible to achieve a better performance than with the isolated usage of heuristics. Nevertheless, the results also show that not all the heuristics work well together and, sometimes it is better to stick to one single variable ordering heuristic and avoid combining them. Among the four variable ordering heuristics analysed in this investigation, the performance of kappa and dom/deg is outstanding. These heuristics, applied in isolation produced very competitive results. More importantly, these two heuristics work particularly well when applied in sequence. Their combination produced changes in the orderings of the variables and then, important reductions in the cost of solving a complete set of instances. When we changed the hyper-heuristic representation to exploit the initial problem features, the results were even better. The selective application of sequences that combine kappa and dom/deg, based on the initial values of constraint density and tightness of the instances, resulted in a larger reduction that the one obtained with the simple sequences that do not consider the problem state.

We also observed that not all the heuristics work well together. For example, Max-conflicts is improved by kappa or dom/deg. Then, Max-conflicts gets a benefit when combined with such heuristics, but our experiments suggests that it is better to apply kappa or dom/deg by themselves and exclude Max-conflicts to achieve the best performance.

One important consideration regarding which heuristics work well together comes from the case of Min-domain and dom/deg. Given the results, dom/deg was better than Min-domain when applied in isolation on set A. When combined in sequences, none of the sequences between Min-domain and dom/deg in the 8-bit space were able to outperform dom/deg. We can conclude that Min-domain is not a good heuristic to be combined with dom/deg. Nevertheless, the pair (Min-domain, kappa) produced some sequences that reduced the cost of kappa on set A. Actually, the reduction achieved by the sequence 01101011 (0 for Min-domain, 1 for kappa), 3.77% with respect to kappa, is larger than the reduction produced by the best sequence of the pair (dom/deg, kappa), 00101001 (0 for dom/deg, 1 for kappa), with 2.43%. When we combined the four heuristics, the fact that Min-domain can be combined with kappa makes it possible that, even though the pair (Min-domain, dom/deg) was not successful, the combination (Min-domain, dom/deg, kappa) improves the search with a saving of 8.60% with respect to the cost of the best heuristic applied in isolation, kappa.

The heuristic space representation is very important in any study; specially for those that will partially explore the heuristic space. In this investigation, we have used a string representation to map every state of the space to one list of heuristics. We are aware that other representations may be more suitable for some methods to explore the space. For other methodologies, it may be necessary to change to a complete different representation. Deciding which representation to use for the heuristic space is always a complex task inherent to any hyper-heuristic study. We would like to extend our results by comparing the effect of other representations and see if they allow us to better understand the relations between heuristics.

Finally, we are interested in testing our findings on new sets of instances, both random with different features and real ones taken from public repositories. Is the performance of the pair (dom/deg, kappa) also a good option in other types of instances? To answer questions like this, we need to incorporate new instances with different properties. Also, we would like to include more variable ordering heuristics, and value ordering ones. We consider that a similar analysis about how well variable and value ordering heuristics interact with each other would be particularly helpful to the understanding of more general solution methods with better results on a large scale.

VII. Acknowledgments

This research was supported in part by ITESM Strategic Project PRY075 “Intelligent Learning for Pattern Recognition and its Application in Medicine and Logistics” and the CONACyT Project under grant 99695.

References

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

[2] S. Minton, A. Phillips, and P. Laird, "Solving large-scale CSP and scheduling problems using a heuristic repair method," in *Proceedings of the 8th AAAI Conference*, 1990, pp. 17–24.

[3] W. W. Chu and P. Ngai, "A dynamic constraint-directed ordered search algorithm for solving constraint satisfaction problems," in *Proceedings of the 1st international conference on Industrial and engineering applications of artificial intelligence and expert systems (IEA/AIE'88)*, vol. 1. ACM Press, 1988, pp. 116–125.

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

[5] J. D. Schaffer and L. J. Eshelman, "Combinatorial optimisation by genetic algorithms: The value of the genotype/phenotype distinction," in *First International Conference on Evolutionary Computation and its applications (EvCa'96)*. Springer, 1996, pp. 110–120.

[6] P. Cowling, G. Kendall, and E. Soubeiga, "Hyperheuristics: A robust optimisation method applied to nurse scheduling," in *Seventh International Conference on Parallel Problem Solving from Nature (PPSN'02)*, ser. Lecture Notes in Computer Science. Springer, 2002, pp. 851–860.

[7] J. Denzinger, M. Fuchs, M. Fuchs, F. F. Informatik, and T. Munchen, "High performance atp systems by combining several ai methods," in *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI'97*. Morgan Kaufmann, 1997, pp. 102–107.

[8] E. Özcan, B. Bilgin, and E. E. Korkmaz, "A comprehensive analysis of hyper-heuristics," *Intelligent Data Analysis*, vol. 12, no. 1, pp. 3–23, 2008.

[9] E. K. Burke, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, and J. R. Woodward, "A classification of hyper-heuristic approaches," in *Handbook of Metaheuristics*, ser. International Series in Operations Research and Management Science, M. Gendreau and J.-Y. Potvin, Eds. Springer, 2010, vol. 146, pp. 449–468.

[10] K. Chakhlevitch and P. Cowling, "Hyperheuristics: Recent developments," in *Adaptive and Multilevel Metaheuristics*, ser. Studies in Computational Intelligence, C. Cotta, M. Sevaux, and K. Srensen, Eds. Springer, 2008, vol. 136, pp. 3–29.

[11] E. Burke, E. Hart, G. Kendall, J. Newall, P. Ross, and S. Schulenburg, "Hyper-heuristics: an emerging direction in modern research technology," in *Handbook of metaheuristics*. Kluwer Academic Publishers, 2003, pp. 457–474.

[12] E. Tsang and A. Kwan, "Mapping constraint satisfaction problems to algorithms and heuristics," Department of Computer Sciences, University of Essex, Tech. Rep. CSM-198, 1993.

[13] 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.

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

[15] E. O'Mahony, E. Hebrard, A. Holland, C. Nugent, and B. O'Sullivan, "Using case-based reasoning in an algorithm portfolio for constraint solving," *Proceedings of the 19th Irish Conference on Artificial Intelligence and Cognitive Science*, 2008.

[16] H. Terashima-Marín, J. C. Ortiz-Bayliss, P. Ross, and M. Valenzuela-Rendón, "Hyper-heuristics for the dynamic variable ordering in constraint satisfaction problems," in *Proceedings of the 10th annual conference on Genetic and evolutionary computation (GECCO'08)*. ACM, 2008, pp. 571–578.

[17] S. A. Bittle and M. S. Fox, "Learning and using hyper-heuristics for variable and value ordering in constraint satisfaction problems," in *Proc. of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference*. ACM, 2009, pp. 2209–2212.

[18] J. C. Ortiz-Bayliss, H. Terashima-Marín, and S. E. Conant-Pablos, "Learning vector quantization for variable ordering in constraint satisfaction problems," *Pattern Recogn. Lett.*, vol. 34, no. 4, pp. 423–432, Mar. 2013.

[19] B. Crawford, R. Soto, C. Castro, and E. Monfroy, "A hyperheuristic approach for dynamic enumeration strategy selection in constraint satisfaction," in *Proc. of the 4th international conference on Interplay between natural and artificial computation. Part II*, ser. IWINAC'11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 295–304.

[20] R. Soto, B. Crawford, E. Monfroy, and V. Bustos, "Using autonomous search for generating good enumeration strategy blends in constraint programming," in *ICCSA (3)'12*, 2012, pp. 607–617.

[21] S. Russell and P. Norvig, *Artificial Intelligence A Modern Approach*. Prentice Hall, 1995.

[22] J. R. Bitner and E. M. Reingold, "Backtrack programming techniques," *Communications of the ACM*, vol. 18, no. 11, pp. 651–656, 1975.

[23] E. C. Freuder, "Synthesizing constraint expressions," *Communications of the ACM*, vol. 21, no. 11, pp. 958–966, 1978.

[24] J. G. Gaschnig, "Experimental case studies of backtrack vs. waltz-type vs. new algorithms for satisficing assignment problems," in *Proceedings of the Canadian Artificial Intelligence Conference*, 1978, pp. 268–277.

[25] A. K. Mackworth, "Consistency in networks of relations," *Artificial Intelligence*, vol. 8, no. 1, pp. 99–118, 1977.

[26] J. Rodriguez, S. Petrovic, and A. Salhi, "An investigation of hyper-heuristic search spaces," in *Proceedings of the 2007 IEEE Congress on Evolutionary Computation (CEC'07)*. IEEE Press, 2007, pp. 3776–3783.

[27] G. Ochoa, J. Vazquez-Rodriguez, S. Petrovic, and E. Burke, "Dispatching rules for production scheduling: A hyper-heuristic landscape analysis," in *IEEE Congress on Evolutionary Computation (CEC'09)*, 2009, pp. 1873–1880.

[28] G. Ochoa, R. Qu, and E. K. Burke, "Analyzing the landscape of a graph based hyper-heuristic for timetabling problems," in *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, ser. GECCO '09. New York, NY, USA: ACM, 2009, pp. 341–348.

[29] I. Maden, A. S. Uyar, and E. Özcan, "Landscape analysis of simple perturbative hyper-heuristics," in *Mendel 2009: 15th International Conference on Soft Computing*, 2009.

[30] R. Wallace, "Analysis of heuristic synergies," in *Recent Advances in Constraints*, ser. Lecture Notes in Computer Science, B. Hnich, M. Carlsson, F. Fages, and F. Rossi, Eds. Springer, 2006, vol. 3978, pp. 73–87.

[31] C. Bessière and J. C. Régin, "Mac and combined heuristics: Two reasons to forsake FC (and CBJ) on hard problems," in *Proceedings of the Second International Conference on Principles and Practice of Constraint Programming*, 1996, pp. 61–75.

[32] B. M. Smith, "A tutorial on constraint programming," University of Leeds, Tech. Rep., 1995.

[33] B. M. Smith and S. A. Grant, "Trying harder to fail first," in *Thirteenth European Conference on Artificial Intelligence (ECAI'97)*. John Wiley & Sons, 1997, pp. 249–253.

[34] I. Gent, E. MacIntyre, P. Prosser, B. Smith, and T.Walsh, "An empirical study of dynamic variable ordering heuristics for the constraint satisfaction problem," in *Proc. of the International Conf. on Principles and Practice of Constraint Programming (CP'96)*, 1996, pp. 179–193.

[35] R. M. Haralick and G. L. Elliott, "Increasing tree search efficiency for constraint satisfaction problems," *Artificial Intelligence*, vol. 14, pp. 263–313, 1980.

[36] B. M. Smith, "Locating the phase transition in binary constraint satisfaction problems," *Artificial Intelligence*, vol. 81, pp. 155–181, 1996.

[37] P. Cheeseman, B. Kanefsky, and W. M. Taylor, "Where the really hard problems are," in *Proceedings of International Joint Conferences on Artificial Intelligence (IJCAI'91)*, 1991, pp. 331–337.

[38] J. C. Ortiz-Bayliss, H. Terashima-Marín, P. Ross, J. I. Fuentes-Rosado, and M. Valenzuela-Rendón, "A neuro evolutionary approach to produce general hyper-heuristics for the dynamic variable ordering in hard binary constraint satisfaction problems," in *Proceedings of the 11th annual conference on Genetic and evolutionary computation (GECCO '09)*. ACM, 2009, pp. 1811–1812.

[39] J. C. Ortiz-Bayliss, E. Özcan, A. J. Parkes, and H. Terashima-Marín, "Mapping the performance of heuristics for constraint satisfaction," in *Proceedings of the 2010 IEEE Congress on Evolutionary Computation (CEC'10)*. IEEE Press, 2010, pp. 1–8.