Branching Schemes and Variable Ordering Heuristics for Constraint Satisfaction Problems: Is there Something to Learn?

José Carlos Ortiz-Bayliss[1], Hugo Terashima-Marín[2] and Santiago Enrique Conant-Pablos[2]

[1] Automated Scheduling, Optimisation and Planning (ASAP)
School of Computer Science, University of Nottingham, UK
Jose.Ortiz_Bayliss@nottingham.ac.uk
[2] Tecnológico de Monterrey
Campus Monterrey, Mexico
{terashima, sconant}@itesm.mx

Abstract. When solving a constraint satisfaction problem by using systematic algorithms it is needed to expand and explore a search tree to find a solution. In this work we study both binary and k-way branching schemes while they interact with various variable ordering heuristics, and how those interactions affect the cost of finding a solution to different instances. Both branching schemes have been used in previous investigations and it is not straight forward to determine the conditions that make one branching scheme better than the other. But we provide evidence that, in order to decide, variable ordering heuristics play a major role in the performance of these branching schemes. This study is intended to work as a preliminary study to develop hyper-heuristics for branching schemes in combination with variable ordering heuristics. The final part of the analysis presents a very simple naive hyper-heuristic that randomly applies binary and k-way branching as the search progresses in combination with some well known variable ordering heuristics. The scope of this paper is to explore the interactions between different variable ordering heuristics and these two branching schemes, in order to produce some relations between their performance. We expect these relations to be used in further studies as the basis for more robust hyper-heuristics that take into consideration the information gathered in this investigation.

Keywords: Constraint Satisfaction, Hyper-heuristics, Branching Schemes, Variable Ordering Heuristics

1 Introduction

A constraint satisfaction problem (CSP) is defined by a set of variables X, where each variable is associated a domain D_x of values subject to a set of constraints C [23, 39]. The goal is to find a consistent assignment of values to variables in such a way that all constraints are satisfied, or to show that such

consistent assignment does not exist. CSP belong to the NP-Complete class [18] and there is a wide range of theoretical and practical applications like scheduling, timetabling, cutting stock, planning, machine vision, temporal reasoning, among others (see for example [17], [27], [13] and [3]).

Several modern methods to solve CSP exist [14, 37], and solutions are found by searching systematically through the possible assignments to variables or by slightly modifying an initial complete and unfeasible solution. In all cases the solution process is guided by heuristics. It is a common practice to use depth first search (DFS) to solve CSPs. When using DFS to solve a CSP, every variable represents a node in the tree and the deeper we go in that tree, the larger the number of variables that have already been assigned a feasible value. Every time a variable is instantiated, a consistency check occurs to verify that the current assignment does not conflict with any of the previous assignments given the constraints within the instance. When an assignment produces a conflict with one or more constraints, the instantiation must be undone, and a new value must be assigned to that variable. When all the feasible values for a variable have been tried and failed, the value of a previously instantiated variable must be changed (this is known as backtracking [5]). Backtracking always goes up one single level in the search tree when a backward move is needed. One of the many ways to reduce the search space is using constraint propagation, where the idea is to propagate the effect of one instantiation to the rest of the variables due to the constraints among them. Thus, every time a variable is instantiated, the values of the other variables that are not allowed due to the current instantiation are removed from the respective domains.

The two most used approaches used to expand the search tree are binary and k-way branching. Binary branching is in fact, a particular case of domain splitting branching [30] when the pivot is selected to be the first value in the domain of the selected variable. The performance of domain splitting for different pivots has not properly been studied and is a topic beyond the scope of this investigation. Here we will use binary branching instead of domain splitting because binary branching is easier to explain, more used in practice, and requires fewer parameters to be tuned.

With k-way branching an instance P is solved as follows. Select a variable x with domain $D_x = \{v_1, v_2, \ldots, v_m\}$. For each $v \in D_x$, we restrict P by setting $x = v$, and recursively try to solve the remaining instance. P has no solution if and only if none of the m possible values for variable x produces a feasible solution given the current instantiated variables. In binary branching, the first choice point creates two alternatives, $x = v_1$ and $x \neq v_1$. The left branch is explored; if the branch fails, or if all solutions are required, the search backtracks to the choice point, and the right branch is followed instead. Crucially, the constraint $x \neq v_1$ is propagated, before a second point is created between $x = v_2$ and $x \neq v_2$, and so on.

With binary branching, the subtrees resulting from successive assignments to a variable are not explored independently; propagating the removal of a value from the current domain of the variable on the right branch can lead to further

domain reductions. This propagation affects the search when future values of the variable are considered. Hence, the order in which values are assigned has more effect in the search compared to k-way branching [35]. Nevertheless the apparent difference in the way they work, most of the research performed on CSPs has used k-way branching, and only a few extra studies on the comparison between some branching strategies have been reported [2, 22, 30, 36].

In some cases, because of the ordering heuristics used, binary branching ends up simulating k-way branching [30]. To see why, consider a pure binary backtracking search with variable ordering based on the MRV heuristic [21] (a heuristic that selects the variable with the smallest domain size). If we select variable x because it has the smallest domain size m, the right branch will produce an instance where x has also the minimum domain size $(m-1)$. Although, this may be the case, the cost of the search may still be different because of the way the values are to be removed from the domains. Thus, we need a way to analyse if these branching schemes produce different search trees or not, and under which circumstances the differences are statistical significant.

The effect of the value ordering heuristics in binary branching, comparing the performance to k-way branching was already studied in [34, 36]. According to those results, binary branching is not, even with the worst value ordering heuristic, worse than k-way branching [34, 36]. Thus, it seems that we should always prefer binary branching over k-way branching; but this may not be correct. In this investigation we have explored the other part of the ordering problem in CSPs, the one related to variable ordering. We provide evidence that there are cases where the use of one branching scheme is preferable to the other when used with a specific variable ordering heuristic, but we have found no evidence that supports that one branching scheme is always better than the other for all variable ordering heuristics.

The research described in this paper is closely related to the work done by Balafoutis and Stergiou [1] and the one conducted by Lagoudakis and Littman [24]. Balafoutis and Stergiou [1] proposed two adaptive branching heuristics to modify the behaviour of binary branching once a variable ordering heuristic has been invoked. Their branching heuristics accept or reject the advise of the variable ordering heuristics once the right branch is to be evaluated. They considered some variable ordering heuristics and compared their branching heuristics on them proving that it is possible to obtain reductions in the cost by modifying the branching strategy as the search progresses. Lagoudakis and Littman [24] described an approach to select branching rules for the Davis-Putnam-Logemann-Loveland procedure for SAT [12]. The results obtained in [24] suggest that it is possible to improve traditional search methods by introducing some decision making and reasoning on top of them, to produce more robust branching rules.

This paper is organized as follows. Section 2 provides information about the variable ordering heuristics used in this investigation, which along with the branching scheme, determine the way the tree is expanded and the cost associated to the search. Later, in Sec. 3 we present the experiments that support the idea that binary and k-way branching are good for different regions of the search

space and we explore the idea of producing a simple hyper-heuristic that uses both branching schemes during the search to analyse its performance. Finally, in Sec. 4 we present the conclusion and the future work of this investigation.

2 Background

Before moving onto the variable ordering heuristics used in this investigation, we need to explain how the CSP instances will be characterized in this document. There are many features that can be used to describe the instances (see for example [20]) but the most important properties used for this purpose are the constraint density p_1 and the constraint tightness, p_2.

The constraint density is a measure of the proportion of constraints within the instance; the closer the value of p_1 to 1, the larger the number of constraints in the instance. The constraint tightness (p_2) represents a proportion of the conflicts within the constraints. A conflict is a pair of values $\langle x, y \rangle$ that is not allowed for two variables at the same time [3]. The higher the number of conflicts, the more unlikely an instance has a solution.

2.1 Variable and Value Ordering Heuristics

This investigation includes a small set of ordering heuristics, four for variable ordering and one for value ordering. For variable ordering we include Minimum Remaining Values (MRV) [21], Kappa (K) [20], degree (DEG) [4] and Max-Conflicts (MXC). For value ordering we use Min-Conflicts (MNC) [26]. In all cases, the tie breaking strategy used is the lexical ordering of the variables and values. In the next lines we briefly describe each one of these heuristics.

Minimum Remaining Values (MRV) [21, 32]. MRV selects the variable with the smaller number of available values in its domain.

Kappa (K) [20]. K selects the variable that minimizes the value of κ of the remaining subproblem. κ is a measure of constrainedness which serves as an indicator of the hardness of the instances with respect to their sizes. For example, instances with $\kappa \ll 1$ have many solutions while instances with $\kappa \gg 1$ are likely to be unsatisfiable. κ is calculated as follows:

$$\kappa = \frac{-\sum_{c \in C_x} \log_2(1-p_c)}{\log_2(D_x)} \tag{1}$$

where D_x represents the domain size of variable x, C_x all the constraints where variable x participates and p_c the fraction of unfeasible tuples of values in constraint c.

[3] In this investigation we have used conflicts that involve only pairs of values because we are using binary CSPs. For constraints of arity a, an a-tuple has to be used. The approach is able to work with constraints of any arity without additional modifications.

Degree (DEG) [16, 38]. DEG prefers the variables connected to the maximum number of uninstantiated variables (forward degree of the variable).

Max-Conflicts (MXC) is a very simple and fast heuristic, and the main idea is to select the variable that is involved in the largest number of conflicts among the constraints in the instance. This instantiation will produce a subproblem that minimises the number of conflicts among the variables left to instantiate.

The value ordering heuristic Min-Conflicts (MNC) [26], is one simple heuristic that prefers the value involved in the minimum number of conflicts. This heuristic is trying to leave the maximum flexibility for subsequent variable assignments. If we select the value that is involved in the minimum number of conflicts, we can suppose that the resulting subproblem will have more solutions than the other subproblems. This heuristic is the direct implementation of the 'most promising' principle for value ordering [19].

2.2 Hyper-heuristics

Hyper-heuristics are motivated with the goal of automating the design of heuristic methods to solve hard computational search problems [6]. Although 'hyper-heuristic' is a relatively new term [10], the idea of automating the design/selection of heuristics can be traced back to the early 1960's, when Fisher and Thompson [15] suggested that combining priority dispatching rules would produce a superior performance than using any of the rules in isolation. According to Burke et al. [8], hyper-heuristics can be divided into two main categories: methodologies that select from a fixed set of heuristics and generate new heuristics. Regarding hyper-heuristics that select among existing heuristics, they produce a mapping between the states of the problem and a feasible heuristic. These methodologies maintain a set of heuristics and then, as the problem changes, decide which heuristic to apply. Examples of these methodologies (although not all of them use the term 'hyper-heuristic' to refer to their approaches) include [31],[28],[11] and [29]. On the other hand, hyper-heuristics that produce heuristics identify critical parts of existing heuristics to create new ones [7]. This study intends to obtain information about the interactions between branching schemes and variable ordering heuristics that could be used in the future to produce hyper-heuristics for branching and variable ordering within CSPs.

3 Experiments and Results

In this section we present the set of experiments conducted during this investigation. These experiments were designed to observe the behaviour of binary and k-way branching on different instances and working together with distinct variable ordering heuristics. The first experiment is designed to identify whether there are significant differences in the performance of the two branching schemes in different regions of the search space, when using distinct variable ordering

heuristics. The second experiment explores the impact of the variable ordering strategy in the performance of each branching scheme when tested on more specific and larger instances. In the final experiment we study the effect of combining the two branching schemes as the search progresses by using a very simple hyper-heuristic approach.

3.1 Is One Branching Scheme Better than the Other?

Our first experiment is designed to confirm that there are differences in the performance of the two branching schemes according to the initial values of p_1 and p_2 of the instance to solve. For this reason, we produced a grid of random instances where each instance contains 20 variables, and each variable has 10 values in its domain. Each cell in the grid covers a region of 0.05×0.05 in the space $p_1 \times p_2$ (each grid contains 20×20 cells). The purpose of such division is to cover all the space $p_1 \times p_2$ in such a way that we can map the performance of some regions of the space to one suitable branching scheme given one variable ordering heuristic. The resolution of the grid is enough to identify these regions and it can also be explored in an acceptable time.

For each cell, 30 instances were generated with random values of p_1 and p_2 within the range of the cell. In this way, 12000 instances were generated and solved for each grid in this experiment. Because four variable ordering heuristics are analysed in this investigation, four grids were produced and solved. In all cases, the cost of the search is measured by the number of consistency checks required to find the first solution or to prove than none exists.

Random model B [33] was used to generate the instances. The random 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 the CSP instance and corresponds to the constraint density, whereas p_2 determines how restrictive the constraints are, and corresponds to constraint tightness of the resulting instance. In model B, there should be exactly $p_1 n(n-1)/2$ constraints (rounded to the nearest integer), and for each pair of constrained variables, the number of inconsistent pairs of values should be exactly $m^2 p_2$ (where m is the uniform domain size of the variables). This generation model was selected because it provided the flexibility to produce instances in the exact region of the space $p_1 \times p_2$ where we needed them.

Each grid was solved with a distinct variable ordering heuristic. We solved the 30 instances in each cell in each grid by using binary branching and k-way branching. MNC was used as value ordering heuristic in all cases. The constraint propagation method used in all the experiments was AC3 [25]. Then, we statistically compared the cost of the two branching schemes on each cell in the grid to observe whether the differences in the performance were statistically significant or not (the cost is given by the number of consistency checks required by the search; this is, the number of times the constraints are revised). To compare the means at each point in the grid we used a bilateral hypothesis test based on a

normal distribution. In each test, we use the null hypothesis $H_0 : \mu_{2B} = \mu_{kB}$ and the alternative hypothesis $H_1 : \mu_{2B} \neq \mu_{kB}$, where μ_{2B} and μ_{kB} stand for the real means of binary branching and k-way branching, respectively.

Figure 1 presents the results of the the statistical test for each variable ordering heuristic by using the two branching schemes under study. The values of each cell correspond to the p-values resulting from the test. Thus, the smaller the value, the stronger the statistical evidence that indicates that both approaches differ in performance. To be consistent with the statistical notation, the cells with values below 0.05 confirm the idea that one branching scheme is better than the other with 5% of significance.

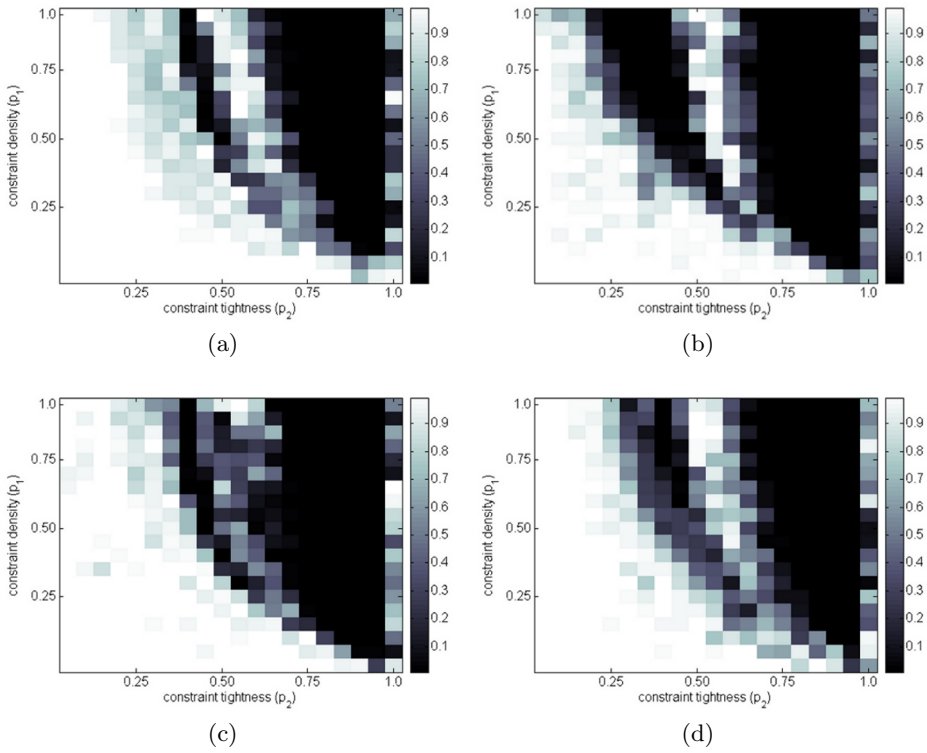

Fig. 1. p-values for the statistical test of the two branching schemes under four different heuristics: (a) MRV, (b) DEG, (c) K and (d) MXC (cells with values below 0.05 indicate significant statistical difference between both approaches with 5% of significance)

The statistical tests were performed to identify the regions where one branching scheme was statistically better than the other; not to identify which one was the best option. For this reason, once we identified the regions where there was statistical evidence that the means of the two branching schemes were differ-

ent, we conducted new tests to identify which was the better branching scheme for each region. We found that, in all the cases where the statistical evidence suggests that the schemes are different, k-way branching obtained the smallest average cost. We can conclude that, for the cases where statistical evidence was found that one approach is better than the other, k-way branching is always better than binary branching on this first experiment.

Trying to understand the differences in the performance of both branching schemes, we can observe that in general (and regardless of the heuristic used), the region where unsatisfiable instances take place, k-way branching is a better option than binary branching for any of the four variable ordering heuristics. Even though there is no statistical evidence that supports that binary branching is better than k-way branching, we observed that there are regions where binary branching obtains a better mean performance than k-way branching. These regions are located just before the well known transition phase [9]; the region where the instances abruptly change from being satisfiable to being unsatisfiable. Nevertheless, no statistical evidence was found that these differences are significant. Also, it is important to mention that, in the region where loose constraints take place (low values of p_1 and p_2), both branching schemes obtain the same cost for most of the instances.

3.2 The Effect on Larger Instances

We have studied the effect of binary and k-way branching when combined with four variable ordering heuristics in the space $p_1 \times p_2$. In this experiment we study the performance of these branching schemes on a different set of instances taken from the literature. The set contains 10 random instances produced with model RB [40] (which is a revision of model B). The set includes 10 satisfiable instances, each one with 30 variables and 15 values in each domain. This instances can be obtained from `http://www.cril.univ-artois.fr/~lecoutre/research/benchmarks/frb30-15.tgz`. The idea behind this new experiment is to focus on a smaller set of larger instances with different features than the ones used before. Also, it is interesting to observe that in the previous experiment, k-way branching proved to be a better option for unsatisfiable instances and, in this case, all the instances have at least one solution.

For these instances, the interactions between variable ordering heuristics and branching schemes produce interesting behaviours. Only DEG seems to be statistically sensitive to the choice of the branching scheme. For the remaining variable ordering heuristics, the differences in the performance do not represent statistical differences with 5% of significance (this time we used a Welch's test, which is a hypothesis test based on t-distribution for small samples). These results are presented by using the box plot in Fig. 2.

The analysis of the medians confirms our conclusions obtained from the analysis of the means. For MRV, K and MXC, the statistical evidence suggests that the medians of binary and k-way branching are equal, with 95% of confidence. This can be concluded because the notches in the box plot overlap. The statisti-

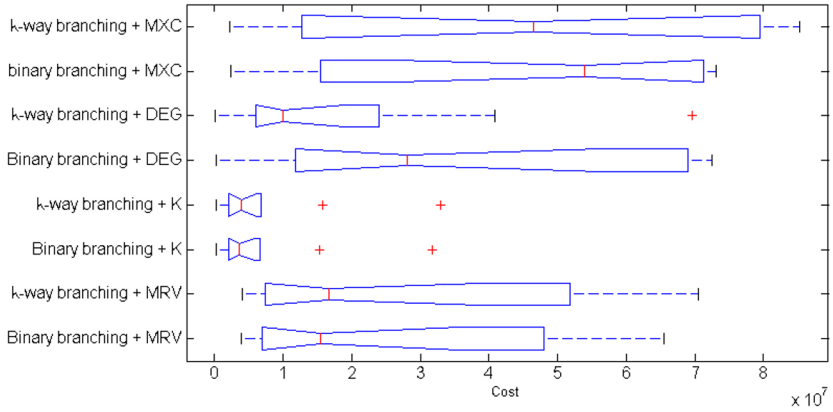

Fig. 2. Boxplot of the results obtained by each branching scheme on the set of large satisfiable instances produced with model RB when combined with each of the four variable ordering heuristics

cal evidence for DEG indicates that the medians of the two branching schemes may not me equal.

Once again, as in the previous experiment, we observed differences in the performance of the distinct methods which are not statistical significant, but this does not mean that they may not be significant in practice. For this reason, and using only the values from the samples, we can conclude that on this set of instances:

- Binary branching is always better than k-way branching when using MRV.
- When K is used as variable ordering heuristic, binary branching always obtains the lowest costs.
- Binary branching is never worse than k-way branching when using DEG.
- When MXC is used as variable ordering heuristic, four of the 10 instances are best solved by using binary branching and the remaining ones are solved more efficiently by using k-way branching.
- We found no case where the both branching schemes produce the same cost.

Thus, we have identified that there are cases where, in combination with the variable ordering heuristic, one branching scheme should be preferred above the other. Now, the question is what occurs when we try to apply these schemes at different stages of the search. This is the purpose of the following experiment.

3.3 A Naive Random Hyper-heuristic: How Difficult is it to Improve the Branching Schemes?

We have observed that the performance of the branching schemes is affected by the selection of the variable ordering heuristic. Then, we cannot decide when one

branching scheme is better than the other without considering the variable ordering heuristic used in the search. In this experiment, we try a very simple way to analyse the probabilities of success of a naive hyper-heuristic that expands the search tree by using binary branching at some stages, and k-way branching at others. For this analysis we propose a naive random hyper-heuristic, which is defined as follows. There is a probability α that binary branching is applied (and a probability $1 - \alpha$ that k-way branching is used instead). At each stage of the search (every time a variable is to be selected) a random decision is made based on a uniform random distribution and the current value of α. For small values of α, it is very likely that k-way branching is used. Thus, when $\alpha = 0$ the search imitates k-way branching and, when $\alpha = 1$, it matches the tree produced by binary branching. This simple approach is used only to measure the probability of success of random combinations of binary and k-way branching. This is not intended to be a robust method for producing a real hybrid application; its purpose is only to show what can occur when we combine these branching approaches as the search progresses.

For each heuristic, nine values of α were tested: 0.1 to 0.9, by increments of 0.1 (values of 0 and 1.0 were taken from the results obtained in Sec. 3). For each value of α, ten runs were conducted and the average cost of these ten runs on the set of instances taken from `http://www.cril.univ-artois.fr/~lecoutre/research/benchmarks/frb30-15.tgz` is reported. We considered that ten runs was enough to give us an idea of the behaviour of a naive random hyper-heuristic on this set of instances. This process is performed for each of the four variable ordering heuristics. Figure 3 presents the results of this experiment.

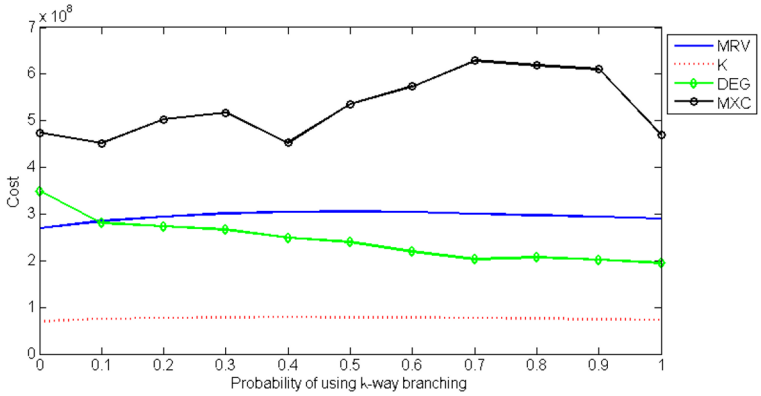

Fig. 3. Average costs of different values of α for the naive random hyper-heuristic

Before conducting this experiment, we had the hypothesis that there should be a value of α for which the average cost of solving the set of instances will be below the cost of both binary and k-way branching. Considering the results on Fig. 3 we have not found evidence to support our hypothesis. It seems that

using the two branching schemes at different stages of the search by using a naive random hyper-heuristic that does not consider the current state under exploration does not help to reduce the cost. On the contrary, the results suggest that this only increases the cost of the search. We think that the selection of the branching scheme at different stages of the search requires a more refined mechanism that uses the information from the current state of the space under exploration to decide the most suitable branching scheme. The evidence suggests that relying the selection of the branching scheme to be used only on naive random decisions is not a good idea. What we can conclude from this experiment is that, if we randomly try to select between the two branching schemes as the search progresses, we are likely to increase the search cost rather than reducing it (see for example the performance of MXC on Fig. 3, where the cost of the search for $\alpha = 0.7$ is larger than the cost of both binary and k-way branching).

The development of a hyper-heuristic that exploits information from the instances to decide which branching scheme to use at each stage of the search will be part of a future investigation. At the moment, we have provided evidence that the use of different branching schemes during the search affects the search cost. The goal for future research is now to propose a mechanism that produces a mapping from problem states to branching schemes and variable ordering heuristics, similar to the work done on hyper-heuristics for variable ordering [11, 29].

4 Conclusion and Future Work

In this investigation we have analysed binary and k-way branching for CSPs together with some important and widely used variable ordering heuristics. Our analysis considers the effect of variable ordering during the search on the performance of binary and k-way branching. Based on our observations, there is no way to state that one of the branching scheme is to be preferred above the other in all cases. Not only there is much to learn from the interactions between branching schemes and variable ordering heuristics, but about the ways to exploit when these interactions produce reductions in the search cost.

One important contribution of this research is the fact that, contrary to what happens in value ordering (where binary branching is never worse than k-way branching [34, 36]), variable ordering heuristics require a deeper analysis to understand their influence on the branching schemes. At the moment, we can conclude that binary and k-way branching are variable ordering dependant and, none of them can be stated to be superior to the other for every instance. In other words, we can conclude that we cannot analyse the performance of the branching schemes without considering the variable ordering heuristic to be used with them.

We have identified regions where some heuristics are more suitable to work with one of these branching approaches. It seems that in general, it looks like for unsatisfiable instances, it is better to use k-way branching. But, more investigation is still needed to confirm whether these results apply to other classes of instances. With regard to the classes of instances used, as future work we are

interested in extending the analysis of the implications in the performance of the branching schemes by considering real instances to observe how these branching schemes behave when tested on structured instances.

This investigation will be used as the basis for developing a more challenging idea, which is the generation of hyper-heuristics that control not only the heuristics used, but the branching schemes at different stages of the search. In order to produce a hyper-heuristic for this problem, we needed to justify that there are differences in the performance of the methods to be combined and that one branching scheme does not dominate the other, which is the case for the branching schemes studied when they interact with the variable ordering heuristics described in this investigation.

5 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.

Bibliography

[1] Balafoutis T, Stergiou K (2010) Adaptive branching for constraint satisfaction problems. In: Proceedings of the 2010 conference on ECAI 2010, IOS Press, Amsterdam, The Netherlands, The Netherlands, pp 855–860

[2] Balafoutis T, Paparrizou A, Stergiou K (2010) Experimental evaluation of branching schemes for the csp. CoRR abs/1009.0407

[3] Berlier J, McCollum J (2010) A constraint satisfaction algorithm for microcontroller selection and pin assignment. In: Proceedings of the IEEE SoutheastCon 2010 (SoutheastCon), pp 348–351

[4] Bessière C, Régin JC (1996) 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, pp 61–75

[5] Bitner JR, Reingold EM (1975) Backtrack programming techniques. Communications of the ACM 18(11):651–656

[6] Burke E, Hart E, Kendall G, Newall J, Ross P, Shulenburg S (2003) Hyper-heuristics: an emerging direction in modern research technology. In: Handbook of metaheuristics, Kluwer Academic Publishers, pp 457–474

[7] Burke EK, Hyde MR, Kendall G, Ochoa G, Özcan E, Woodward JR (2009) Exploring hyper-heuristic methodologies with genetic programming. In: Mumford C, Jain L (eds) Computational Intelligence, Intelligent Systems Reference Library, vol 1, Springer Berlin Heidelberg, pp 177–201

[8] Burke EK, Hyde M, Kendall G, Ochoa G, Özcan E, Woodward JR (2010) A classification of hyper-heuristic approaches. In: Gendreau M, Potvin JY (eds) Handbook of Metaheuristics, International Series in Operations Research & Management Science, vol 146, Springer US, pp 449–468

[9] Cheeseman P, Kanefsky B, Taylor WM (1991) Where the really hard problems are. In: Proceedings of International Joint Conferences on Artificial Intelligence (IJCAI'91), pp 331–337
[10] Cowling P, Kendall G, Soubeiga E (2000) A hyperheuristic approach to scheduling a sales summit. In: Burke EK, Erben W (eds) Practice and Theory of Automated Timetabling III : Third International Conference (PATAT'00), Springer, Lecture Notes in Computer Science, vol 2079, pp 176–190
[11] Crawford B, Soto R, Castro C, Monfroy E (2011) A hyperheuristic approach for dynamic enumeration strategy selection in constraint satisfaction. In: Proceedings of the 4th international conference on Interplay between natural and artificial computation: new challenges on bioinspired applications - Volume Part II, Springer-Verlag, Berlin, Heidelberg, IWINAC'11, pp 295–304
[12] Davis M, Logemann G, Loveland D (1962) A machine program for theorem-proving. Communications of ACM 5(7):394–397
[13] Dunkin N, Allen S (1997) Frequency assignment problems: Representations and solutions. Tech. Rep. CSD-TR-97-14, University of London
[14] Epstein SL, Freuder EC, Wallace RJ (2005) Learning to support constraint programmers. Computational Intelligence 21(4):336–371
[15] Fisher H, Thompson GL (1961) Probabilistic learning combinations of local job-shop scheduling rules. In: Factory Scheduling Conference, Carnegie Institute of Technology
[16] Freuder EC (1982) A sufficient condition for backtrack-free search. Journal of the ACM 29(1):24–32
[17] Freuder EC, Mackworth AK (1994) Constraint-Based Reasoning. MIT/Elsevier
[18] Garey MR, Johnson DS (1979) Computers and Intractability; A Guide to the Theory of NP-Completeness. W. H. Freeman
[19] Geelen PA (1992) Dual viewpoint heuristics for binary constraint satisfaction problems. In: Proceedings of the 10th European conference on Artificial intelligence (ECAI'92), John Wiley & Sons, pp 31–35
[20] Gent I, MacIntyre E, Prosser P, Smith B, TWalsh (1996) An empirical study of dynamic variable ordering heuristics for the constraint satisfaction problem. In: Proceedings of the International Conference on Principles and Practice of Constraint Programming (CP'96), pp 179–193
[21] Haralick RM, Elliott GL (1980) Increasing tree search efficiency for constraint satisfaction problems. Artificial Intelligence 14:263–313
[22] Hwang J, Mitchell DG (2005) 2-way vs d-way branching for CSP. In: Proceedings of the International Conference on Principles and Practice of Constraint Programming (CP'05), Springer, pp 343–357
[23] Kumar V (1992) Algorithms for constraint satisfaction: a survey. AI Magazine 13(1):32–44
[24] Lagoudakis MG, Littman ML (2001) Learning to select branching rules in the dpll procedure for satisfiability. Electronic Notes in Discrete Mathematics 9(0):344–359

[25] Mackworth AK (1977) Consistency in networks of relations. Artificial Intelligence 8(1):99–118
[26] Minton S, Johnston MD, Phillips A, Laird P (1992) Minimizing conflicts: A heuristic repair method for CSP and scheduling problems. Artificial Intelligence 58:161–205
[27] Montanari U (1974) Networks of constraints: fundamentals properties and applications to picture processing. Information Sciences 7:95–132
[28] O'Mahony E, Hebrard E, Holland A, Nugent C, O'Sullivan B (2008) Using case-based reasoning in an algorithm portfolio for constraint solving. Proceedings of the 19th Irish Conference on Artificial Intelligence and Cognitive Science
[29] Ortiz-Bayliss JC, Terashima-Marín H, Conant-Pablos SE (2013) Learning vector quantization for variable ordering in constraint satisfaction problems. Pattern Recogn Lett 34(4):423–432
[30] Park V (2004) An empirical study of different branching strategies for constraint satisfaction problems. PhD thesis, University of Waterloo
[31] Petrovic S, Qu R (2002) 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, pp 336–340
[32] Purdom PW (1983) Search rearrangement backtracking and polynomial average time. Artificial Intelligence 21:117–133
[33] Smith BM (1996) Locating the phase transition in binary constraint satisfaction problems. Artificial Intelligence 81:155–181
[34] Smith BM (2005) Value ordering for finding all solutions. In: International Joint Conference on Artificial Intelligence (IJCAI'05, pp 311–316
[35] Smith BM, Grant SA (1995) Sparse constraint graphs and exceptionally hard problems. In: Proceedings of the International Joint Conferences on Artificial Intelligence (IJCAI'95), pp 646–651
[36] Smith BM, Sturdy P (2004) An empirical investigation of value ordering for finding all solutions. In: Workshop on Modelling and Solving Problems with Constraints
[37] Soto R, Crawford B, Monfroy E, Bustos V (2012) Using autonomous search for generating good enumeration strategy blends in constraint programming. In: ICCSA (3)'12, pp 607–617
[38] Wallace R (2006) Analysis of heuristic synergies. In: Hnich B, Carlsson M, Fages F, Rossi F (eds) Recent Advances in Constraints, Lecture Notes in Computer Science, vol 3978, Springer, pp 73–87
[39] Williams CP, Hogg T (1992) Using deep structure to locate hard problems. In: Proceedings of AAAI'92, pp 472–477
[40] Xu K, Boussemart F, Hemery F, Lecoutre C (2007) Random constraint satisfaction: Easy generation of hard (satisfiable) instances. Artificial Intelligence 171(8-9):514–534