Multi-Level Variable Ordering Heuristics for the Constraint Satisfaction Problem

Christian Bessière[1], Assef Chmeiss[2], and Lakhdar Saïs[2]

[1] Member of the Coconut group
LIRMM-CNRS (UMR 5506), 161, rue Ada,
34392 Montpellier Cedex 5, France
bessiere@lirmm.fr
[2] CRIL - Université d'Artois - IUT de Lens
Rue de l'université - SP 16
62307 LENS Cedex, France
{chmeiss, sais}@cril.univ-artois.fr

Abstract. The usual way for solving constraint satisfaction problems is to use a backtracking algorithm. One of the key factors in its efficiency is the rule it will use to decide on which variable to branch next (namely, the variable ordering heuristics). In this paper, we attempt to give a general formulation of dynamic variable ordering heuristics that take into account the properties of the neighborhood of the variable. An empirical evaluation on random CSPs and a sample of real instances shows that the obtained heuristics can improve significantly the current best ones.

1 Introduction

Constraint satisfaction problems (*CSPs*) are widely used to solve combinatorial problems appearing in a variety of application domains. They involve finding solution in a *constraint network*, i.e., finding *values* for network *variables* subject to *constraints* on which combinations are acceptable.

The usual technique to solve CSPs is systematic backtracking. It repeatedly chooses a variable, attempts to assign it one of its values, and then goes to the next variable, or backtracks in case of failure. This technique is at the basis of almost all the CSP solving engines. But if we want to tackle highly combinatorial problems, we need to enhance this basic search procedure with clever improvements.

A crucial improvement to be added is look-ahead value filtering, which consists in removing in future domains values that cannot belong to a solution extending the current partial instantiation. Many works have studied the different levels of filtering that can be applied at each node of the search tree. In the following, we will use maintaining arc consistency (MAC) as our search algorithm [13, 1].

Another improvement that has shown to be of major importance is the ordering of the variables, namely, the criterion under which we decide which variable will be the next to be instantiated. Many variable ordering heuristics for

solving CSPs have been proposed over the years. However, the criteria used in those heuristics to order the variables are often quite simple, and concentrate on characteristics inherent to the variable to be ordered, and not too much on the influence its neighborhood could have. Those that use more complex criteria, essentially based on the constrainedness or the solution density of the remaining subproblem, need to evaluate the tightness of the constraints, and so, need to perform many constraint checks.

The goal of this paper is to propose heuristics that take into account properties of the neighborhood in the criterion of choice of a variable, while remaining free of any constraint check. After a recall on the existing variable ordering heuristics for the CSP, and a brief discussion on their efficiency, we take a look at the branching rules used in SAT. Then, following the ideas developed for SAT, we define a general criterion that can be used on CSPs. We give instantiations of this general criterion, which depend on the choice of some operators and parameters to approximate the constrainedness of variables and constraints. Finally, we take some simple cases for that criteria (fixing the parameters), that we experiment and compare to existing heuristics.

The rest of the paper is organized as follows. Section 2 gives the necessary definitions and notations. It summarizes existing work on this topic, and shows experimentally how the best already known heuristics relate to each other. Section 3 presents our multi-level heuristics. In Section 4, we compare the best existing heuristics to some of those proposed in Section 3. Sections 5 and 6 discuss possible future work, and conclude the paper.

2 Preamble

2.1 Definitions and notations

A constraint network is defined by a set of variables, each taking values in its finite domain, and a set of constraints restricting the possible combinations of values between variables. For simplicity we restrict our attention here to *binary* constraint networks, where the constraints involve two variables.[1] Binary constraints are binary relations. If a variable X_i has a domain of possible values $D(X_i)$ and a variable X_j has a domain $D(X_j)$, the constraint on X_i and X_j, R_{ij}, which specifies the allowed pairs of values for (X_i, X_j), is a subset of the Cartesian product $D(X_i) \times D(X_j)$. Asking whether a pair of values is allowed by a constraint is called a *constraint check*.

Any constraint network can be associated with a *constraint graph* in which the nodes are the variables of the network, and an edge links a pair of nodes if and only if there is a constraint on the corresponding variables. $\Gamma_{init}(X_i)$ denotes the set of nodes sharing an edge with the node X_i (its initial *neighbors*). We define the set $\Gamma(X_i)$ as the *current* neighborhood of X_i, namely, the neighbors remaining uninstantiated once a backtracking search procedure has instantiated

[1] However, the techniques presented in the following of this paper can easily be extended to general non-binary constraint networks.

the set $Y = \{X_{i_1}, \ldots, X_{i_k}\}$ of variables, i.e., $\Gamma(X_i) = \Gamma_{init}(X_i) \setminus Y$. The size of $\Gamma(X_i)$ (resp. $\Gamma_{init}(X_i)$) is called the *degree* (resp. initial degree) of X_i.

2.2 Overview on the existing variable orderings

Ordering variables prior to or during search of solution in a CSP has been understood of prime importance for a long time. The first kind of criteria used for ordering variables are based on the structure of the network. Since in classical search procedures the structure of the network does not change during search, these variable orderings (VOs) keep the same ordering of the variables during the whole search. They are called static VOs (SVOs), and simply replace the lexicographic ordering by something more appropriate to the structure of the network before starting search. Some examples of these heuristics are `minwidth`, which chooses an ordering minimizing the width of the constraint network [5], `maxdeg`, which orders the variables in decreasing size of their neighborhood [4], `bandwidth`, which minimizes the bandwidth of the constraint graph [16], etc.

The second type of VOs, adapted to the technique of look ahead value pruning, takes into account the size of the domains to choose the next variable to be instantiated. As soon as we put some filtering into a search procedure, the domains have no reason to keep the same size from one branch to another. So, heuristics keeping the size of the domains as a criterion of selection have no reasons to take the variables in the same order from one branch to the other. This is why this kind of orderings, introduced by Haralick and Elliott in 1980 are called dynamic VOs (DVOs). The one introduced by Haralick and Elliott in [9] is `dom`, the heuristic choosing as next variable the one with the smallest remaining domain.

Since `dom` can be completely fooled by the structure, especially at the beginning of the search, when domains have more chances to be of equal size, `dom+deg` has been proposed, which breaks ties in `dom` by preferring the variable with the highest initial degree [6]. Another heuristic, close to `dom+deg`, is the one derived from the Brélaz heuristic (proposed for graph coloring) [2]. We note it `dom+futdeg`. It breaks ties in `dom` by preferring the variable with highest future degree [15]. Smith also improved `dom+futdeg` by adding to it a second and a third tie breakers, namely, the size of the smallest neighbor, and the number of triangles in which the first chosen variable is involved. She called this DVO `BZ3`.

However, both `dom+deg`, `dom+futdeg`, and `BZ3` use the domain size as the main criterion. The degree of the variables is considered only in case of ties. Imagine a case where a variable X_i has a domain size of 20 and a degree of one, while X_j has a domain size of 21 and a degree of 10. `dom+futdeg` or `BZ3` will prefer X_i to X_j while there are many chances that X_j would have been a better choice. This is why combined heuristics were proposed ([1]), which do not give priority to the domain size or degree of variables, but use them equally in the criterion. `dom/deg` chooses as the next variable, the variable X_i minimizing the ratio $|D(X_i)|/|\Gamma_{init}(X_i)|$, while `dom/futdeg` minimizes $|D(X_i)|/|\Gamma(X_i)|$. `dom/futdeg` has extensively been studied in [14] under the name `DD`.

Finally, there is another kind of DVOs which it is worth mentioning: those taking into account the tightness of the constraints. They look how much a given constraint restricts the remaining of the problem instead of considering all of them as equivalent. Examples of such heuristics can be found in [11, 7, 8]. The drawback of these heuristics is the need of checking the tightness of a constraint, which is very constraint checks consuming (and thus in cpu time also). None of these heuristics has been shown as better than those dealing only with the constraint graph structure.

2.3 A few words about experiments

In the following of this paper, the performances of the different heuristics are evaluated both on real instances and on uniform random CSPs generated according to the model B [12]. Such a generator admits four parameters: the number N of variables, the common size of the initial domains D, the proportion $p1$ of constraints in the network (or the number $C = p1 * N \cdot (N-1)/2$ of constraints), and the proportion $p2$ of forbidden pairs of values in a constraint (or the number $T = p2 * D * D$ of forbidden pairs). In the following, we will use C and T instead of $p1$ and $p2$. C is called the *density* of the constraint graph and T the *tightness* of the constraints. The real instances are taken from the FullRLFAP archive,[2] which contains instances of radio link frequency assignment problems (RLFAPs). They are described in [3]. In all our experiments, we stop search after the first solution is found. The search procedure used is MAC.

2.4 Observations

The strong relationship between the instantiation ordering of variables and the size of the search tree has motivated many works on the design of "good" heuristics. However, an empirical comparison of such heuristics is not an obvious task. When comparing variable orderings on a sample of instances of CSPs, it can arise that we see little difference between two given VOs, giving us the temptation to select the simplest one. However, what is important to see is the evolution of their differences when problems become larger and harder. Indeed, a small difference on problems of small size can become huge when the size of the problems increases.

To give an insight into the state of the art, we present in Figure 1 results for several well-known heuristics: `dom` is the oldest and most well known DVO, and `dom/futdeg` and `BZ3` can be considered as the best current VOs for CSPs [14, 15]. In this experiment, we consider random CSPs $< N, D, C, T >$, where the number of variables N grows from 20 to 200,[3] while the domain size and the average degree of the variables are kept constant ($|D| = 10$, and $2 \cdot C/N = 5$). The tightness remains at the cross-over point ($T = T_{crit}$), which remains stable

[2] We thank the Centre d'Electronique de l'Armement (France).

[3] For `dom` (resp. `BZ3`) N grows until 140 (resp. 190), because of the huge time needed to find a solution.

at the value 55 since the networks generated all have the same average degree for their variables.

We see that on small sizes, `dom` is already significantly worse than `BZ3` and `dom/futdeg`. These two last have very close performances until 60 variables. When N becomes greater than 60, the benefit of `dom/futdeg` becomes bigger and bigger, exceeding one order of magnitude for $N > 150$.

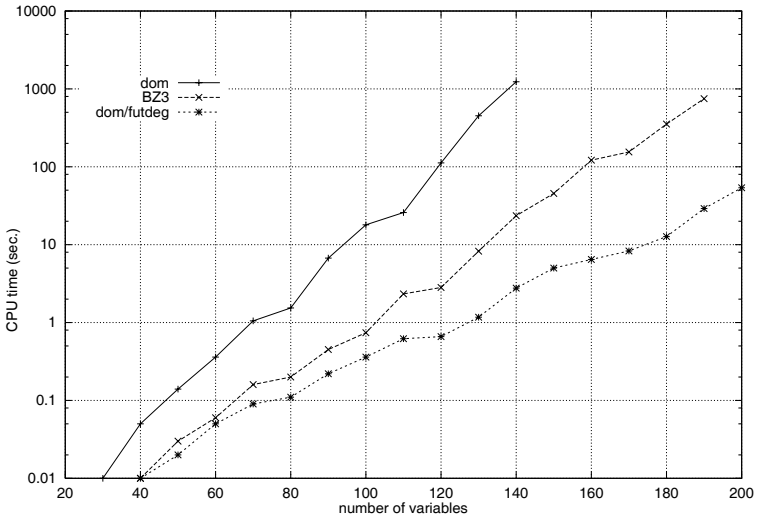

Fig. 1. $< N, D = 10, C = 5 * N/2, T_{crit} >$: cpu time needed (log scale).

3 Multi-level DVOs

One of the key features for the efficiency of a backtrack search method lies in its branching (or splitting) strategy. At each step of the search process, a problem P is reduced into a finite number of sub-problems $(P_1, P_2, \ldots, P_{|D(X_i)|})$, where X_i is the chosen variable. As stated by Hooker and Vinay [10] in the context of satisfiability problems (SAT): "a branching rule works better when it creates simpler subproblems".[4] The performance of branching rules is better explained by the previous simplification hypothesis. Following this idea, and since contrary to SAT problems the domains are not all binary, we think that a good DVO should reduce both the *number* and the *difficulty* of such subproblems. In other words, a branching rule performs well when it creates less and simpler subproblems.

Indeed, choosing the variable with minimum domain (resp. with maximum degree) reduces the number (resp. difficulty) of such subproblems, whereas combining both, such as `dom/futdeg` does, tends to satisfy both objectives. This

[4] called a simplification hypothesis

explains the importance of these two syntactical parameters on the design of good heuristics.

To go further in that direction, we propose a general formulation of DVOs which integrates in the selection function a measure of the constrainedness of the given variable. The constrainedness of a variable can be defined as a function of the constraints involving the variable. One could choose semantical constraints-based measures (e.g., number of allowed tuples) or syntactical ones (e.g., size of the Cartesian product of the domains). Choosing the most constrained variable should have a great impact on the search space, leading the search to the most constrained parts of the CSP, and thus provoking early detection of local inconsistencies.

As recalled in Section 2.2, this is not a new idea in CSPs, but what we propose below is a new and general formulation of DVOs, which remains free of any constraint check.

3.1 A general criterion free of constraint checks

Let us first define $W(R_{ij})$ as the weight of the constraint R_{ij} and,

(1) $W(X_i) = \frac{\sum_{X_j \in \Gamma(X_i)} W(R_{ij})}{|\Gamma(X_i)|}$

as the mean weight of the constraints involving X_i. In order to maximize the number of constraints involving a given variable and to minimize the mean weight of such constraints, the next variable to branch on should be chosen according to the minimum value of

(2) $H(X_i) = \frac{W(X_i)}{|\Gamma(X_i)|}$

over all uninstantiated variables (numerator to minimize the weight, and denominator to maximize the number of constraints).

For complexity reasons we already mentioned above, the weight we will associate to a constraint must be something cheap to compute (e.g., free of constraint checks). It can be defined by $W(R_{ij}) = \alpha(X_i) \circledcirc \alpha(X_j)$, where $\alpha(X_i)$ is instantiated to a simple syntactical property of the variable such as $|D(X_i)|$ or $\frac{|D(X_i)|}{|\Gamma(X_i)|}$, and $\circledcirc \in \{+, \times\}$. For $\alpha(X_i) = |D(X_i)|$, and $\circledcirc = \times$, the weight associated to a given constraint R_{ij} represents an upper bound of the number of tuples allowed by R_{ij}.

We obtain the new formulation of (2):

(3) $H^{\circledcirc}_{\alpha}(X_i) = \frac{\sum_{X_j \in \Gamma(X_i)} (\alpha(X_i) \circledcirc \alpha(X_j))}{|\Gamma(X_i)|^2}$

3.2 Multi-level generalization

In the formulation of the DVOs presented above, the evaluation function $H(X_i)$ considers only the variables at distance one from X_i (first level or neighborhood). However, when arc consistency is maintained (MAC), the instantiation of a value

to a given variable X_i could have an immediate effect not only on the variables of the first level, but also on those at distance greater than one.

To maximize the effect of such a propagation process on the CSP, and consequently to reduce the difficulty of the subproblems, we propose a generalization of the DVO $H^{\circledcirc}_{\alpha}$ such that variables at distance k from X_i are taken into account. This gives what we call a "multi-level DVO", $H^{\circledcirc}_{(k,\alpha)}$. To obtain this multi-level DVO, we simply replace $\alpha(X_j)$ in formula (3) by a recursive call to $H^{\circledcirc}_{(k-1,\alpha)}(X_j)$. This means that to compute $H^{\circledcirc}_{(k,\alpha)}$ on a given variable, we need to compute $H^{\circledcirc}_{(k-1,\alpha)}$ on all its neighbors, and so on. The recursion terminates with $H^{\circledcirc}_{(0,\alpha)}$, equal to α. This is formally stated as follows:

(4) $H^{\circledcirc}_{(0,\alpha)}(X_i) = \alpha(X_i)$

(5) $H^{\circledcirc}_{(k,\alpha)}(X_i) = \frac{\sum_{X_j \in \Gamma(X_i)} (\alpha(X_i) \circledcirc H^{\circledcirc}_{(k-1,\alpha)}(X_j))}{|\Gamma(X_i)|^2}$

3.3 Some instantiations of the multi-level formula

We show now some of the different DVOs we can obtain from $H^{\circledcirc}_{(k,\alpha)}$ by instantiating $\circledcirc$ to $+$ or $\times$, $\alpha(X_i)$ to $|D(X_i)|$ or $\frac{|D(X_i)|}{|\Gamma(X_i)|}$, and k to 0 or 1:[5]

1. For $\circledcirc = +$:
   - and $\alpha(X_i) = |D(X_i)|$ we obtain:
     (6) $H^{+}_{(0,dom)}(X_i) = |D(X_i)|$,
     (7) $H^{+}_{(1,dom)}(X_i) = \frac{|D(X_i)|}{|\Gamma(X_i)|} + \frac{\sum_{X_j \in \Gamma(X_i)} |D(X_j)|}{|\Gamma(X_i)|^2}$
   - and $\alpha(X_i) = \frac{|D(X_i)|}{|\Gamma(X_i)|}$ we obtain :
     (8) $H^{+}_{(0,dom/futdeg)}(X_i) = \frac{|D(X_i)|}{|\Gamma(X_i)|}$,
     (9) $H^{+}_{(1,dom/futdeg)}(X_i) = \frac{|D(X_i)| + \sum_{X_j \in \Gamma(X_i)} \frac{|D(X_j)|}{|\Gamma(X_j)|}}{|\Gamma(X_i)|^2}$.
2. For $\circledcirc = \times$:
   - and $\alpha(X_i) = |D(X_i)|$ we obtain:
     (10) $H^{\times}_{(0,dom)}(X_i) = |D(X_i)| = H^{+}_{(0,dom)}(X_i)$,
     (11) $H^{\times}_{(1,dom)}(X_i) = \frac{|D(X_i)| \times \sum_{X_j \in \Gamma(X_i)} |D(X_j)|}{|\Gamma(X_i)|^2}$
   - and $\alpha(X_i) = \frac{|D(X_i)|}{|\Gamma(X_i)|}$ we obtain :
     (12) $H^{\times}_{(0,dom/futdeg)}(X_i) = \frac{|D(X_i)|}{|\Gamma(X_i)|} = H^{+}_{(0,dom/futdeg)}(X_i)$,
     (13) $H^{\times}_{(1,dom/futdeg)}(X_i) = \frac{|D(X_i)| \times \sum_{X_j \in \Gamma(X_i)} \frac{|D(X_j)|}{|\Gamma(X_j)|}}{|\Gamma(X_i)|^3}$.

In the following, $H^{\circledcirc}_{(0,dom)}$ and $H^{\circledcirc}_{(0,dom/futdeg)}$ are denoted by their classical name, `dom` and `dom/futdeg`, respectively. $H^{\circledcirc}_{(k,dom)}$ and $H^{\circledcirc}_{(k,dom/futdeg)}$ are denoted by `H_k_dom_`$\circledcirc$ and `H_k_dom/futdeg_`$\circledcirc$ respectively.

[5] In the same way, other instantiation can be obtained for $k > 1$.

4 Experiments

We will now compare experimentally the behavior of the new DVOs defined above on several classes of random CSPs (Section 4.1). In Section 4.2 results on real instances from the FullRLFAP archive are presented. To show the feasibility of the multi-level approach, we present experiments obtained with the second level instantiations ($k > 1$) on a sample of random CSPs (Section 4.3).

We compare our DVO heuristics with the most efficient one previously known, `dom/futdeg` (see Section 2.4). In our tests we have used different kinds of measures of performance for the DVOs tested: cpu time in seconds (time),[6] number of constraint checks (#ccks), and number of visited nodes (#nodes). The different curves obtained using these different measures are very similar, so we just present those with cpu time. When necessary, the other measures (#ccks and #nodes) are also given.

4.1 Random instances

To get an accurate view of the behavior of the above DVO heuristics on random CSPs, several classes are considered:

1. the class of CSPs presented and described in Section 2.4, with a number of variables growing from 60 to 230 (Figure 2).
2. CSPs for which we vary the tightness when N, D, and C are kept constant. Figures 3 and 4 give results for $N = 80$, $D = 10$, and three different densities: $C = 200$ (Figure 3), $C = 400$ (Figure 4), and $C = 800$ (Table 1).
3. CSPs with a greater number of variables and/or a greater domain size. The results are summarized in Table 2. They are all at the crossover point.

We present the performance of the first level DVOs (`H_1_dom_`$\circledcirc$ and `H_1_dom/futdeg_`$\circledcirc$) with respect to the most efficient DVO previously known, `dom/futdeg`. Each point in the curves represents the average time for 100 generated problems.

In Figure 2, N grows, while the domain size and the average degree remain constant. The tightness is always at the cross-over point. As we can see, except for `H_1_dom_`$\times$, all the first-level DVOs outperform the best known DVO `dom/futdeg` on this class of CSPs. `H1_dom/futdeg_`$+$ has the best performance. It is interesting to note that the gain increases as the number of variables increases.

Figures 3 and 4 confirm the improvements obtained with the first-level instantiations. As we can see in Figure 4, when density grows ($C = 400$, i.e., average degree equal to 10), `H_1_dom_`$\times$ becomes slightly better than `dom/futdeg`, and `H_1_dom/futdeg_`$\times$ becomes better than `H_1_dom/futdeg_`$+$. The instantiations obtained with $\alpha(X_i) = \frac{|D(X_i)|}{|\Gamma(X_i)|}$ are more efficient than those with $\alpha(X_i) = |D(X_i)|$.

[6] Experiments on random (resp. real) instances have been run on a PC Pentium III 667 MHz (resp. Pentium II 300MHz) under Linux.

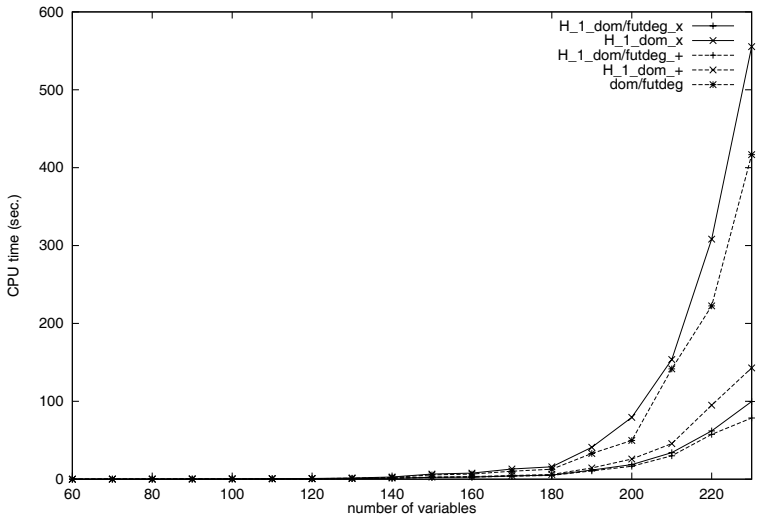

Fig. 2. $< N, D = 10, C = 5 * N/2, T_{crit} >$: cpu time.

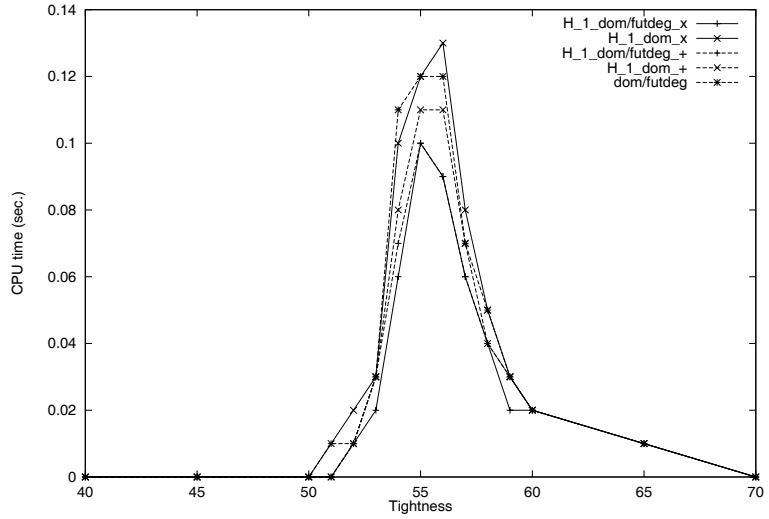

Fig. 3. $< N = 80, D = 10, C = 200, T >$: cpu time.

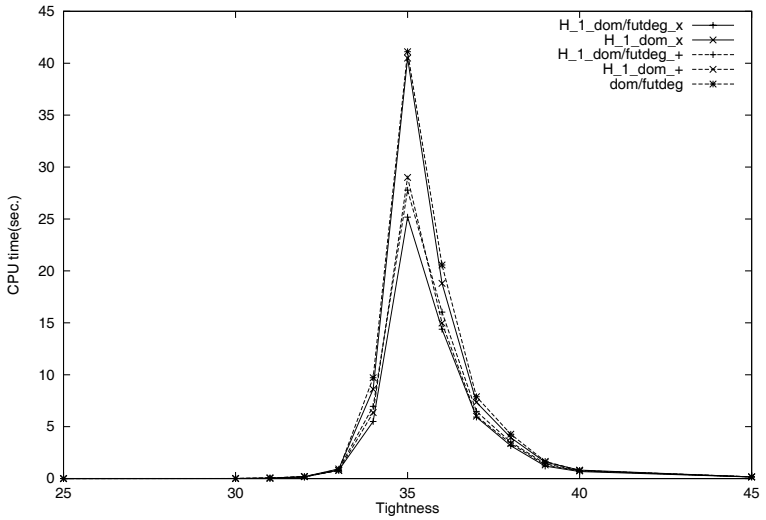

Fig. 4. $< N = 80, D = 10, C = 400, T >$: cpu time.

Table 1 shows additional results on denser and harder CSPs with 20 as the average degree: $< N = 80, D = 10, C = 800, T >$. Because of the huge execution time, only 10 instances were tested. They are generated at the cross-over point ($T_{crit} = 20$). On these dense CSPs, all the first level DVOs noticeably outperform `dom/futdeg`. And, `H_1_dom/futdeg_`$\times$ confirms that it becomes better and better when density grows.

Table 1. $< N = 80, D = 10, C = 800, T_{crit} = 20 >$ at the cross-over point : checks, visited nodes and cpu time

	#ccks (million)	#nodes (million)	time (sec.)
`dom/futdeg`	160,338	54.10	20,713.78
`H_1_dom/futdeg_+`	141,169	39.38	18,097.21
`H_1_dom_+`	121,616	37.27	16,298.56
`H_1_dom/futdeg_`$\times$	108,978	34.74	14,221.82
`H_1_dom_`$\times$	145,496	49.17	17,917.18

In its first three first lines, Table 2 gives details on the number of constraint checks (#ccks) and cpu time for the cross-over point of some classes described only by cpu time graphs in Figures 2, 3, and 4. The last four lines show that if we increase even more the number of variables ($< \mathbf{300}, 6, 1200, 12 >$, $< \mathbf{100}, 10, 500, 35 >$), or if we increase the domain size ($< 150, \mathbf{15}, 375, 139 >$),

Table 2. A sample of random CSPs at the cross-over point : checks & cpu time

	dom/futdeg		H_1_dom_+		H_1_dom/futdeg_+		H_1_dom_x		H_1_dom/futdeg_x	
Problems $< N, D, C, T >$	#ccks (million)	time (sec.)	#ccks (million)	time (sec.)	#ccks (million)	time (sec.)	#ccks (million)	time (sec.)	#ccks (million)	time (sec.)
$< 80, 10, 200, 55 >$	0.79	0.12	0.71	0.11	0.67	0.10	0.81	0.12	0.63	0.10
$< 80, 10, 400, 35 >$	404.45	41.11	256.98	29.00	255.06	27.78	367.89	40.49	222.20	25.18
$< 200, 10, 500, 55 >$	480.25	48.62	232.97	24.68	136.68	14.68	720.64	74.97	171.89	18.60
$< 100, 10, 250, 55 >$	2.64	0.36	1.86	0.27	1.45	0.23	2.74	0.39	1.36	0.22
$< 100, 10, 500, 35 >$	3,713.87	390.37	2,195.44	239.71	2,240.78	253.08	3,340.33	362.12	1,916.73	216.86
$< 150, 15, 375, 139 >$	2,267.35	204.22	939.98	87.65	516.72	48.85	2,839.49	260.51	825.97	78.35
$< 300, 6, 1200, 12 >$	9,158.42	1,351.39	3,989.83	626.06	4,338.20	709.32	7,938.12	1,243.61	2,968.28	495.35

the gain of the `H_1` heuristics continue to grow compared to `dom/futdeg`. These are promising observations.

As a synthesis of the results on different classes of random CSPs, we can say that, except `H_1_dom_x`, the first level DVOs improve significantly the well known DVO `dom/futdeg`. Furthermore, in general, `H_1_dom/futdeg_`$\circledcirc$ are better than `H_1_dom_`$\circledcirc$. This is not surprising because the former take into account the connectivity of the neighborhood of the chosen variable. Finally, `H_1_dom/futdeg_`$\times$ is more efficient than `H_1_dom/futdeg_`+ on CSPs with higher densities. On the contrary, `H_1_dom/futdeg_`+ is more efficient on sparse ones.

4.2 Real instances

In this subsection, we compare the behavior of the DVOs used in Section 4.1 on the real instances of the FullRLFAP archive. It contains 11 real instances, scen01 to scen11, and 14 artificially generated instances, graph01 to graph14. These instances all contain an optimization criterion, and so, are not pure satisfiability problems. The consequence is that most of these instances (except scen11 and graph10) are trivially solved by MAC with any of the discussed DVOs, finding a solution (not necessarily optimal!), or detecting inconsistency easily. To produce harder instances, we took the 5 trivially satisfiable real instances, scen01 to scen05, and reduced the available frequencies step by step, leading to a series of problems scenXX-k, where k is the number of frequencies removed from the problem (starting from the largest frequency to the smaller ones). There always exists a value k_0 for which scenXX-k_0 is satisfiable, and scenXX-$(k_0 + 1)$ is inconsistent.

For the real instances scen06 to scen10, which are trivially inconsistent, the constraints are partitioned into 5 sets, depending on their importance in the problem. The set #0 contains the hard constraints, the set #4 contains the least important constraints. We used that to produce a series of five different problems from a scenXX. For instance, scenXX-012 denotes the instance containing the constraints at levels 0, 1, and 2. Here again, for a given scenXX, we can go from an over-constrained problem (scenXX-01234) to an under-constrained one (scenXX-0) by removing levels of constraints.

These two protocols do not respect exactly the optimization criteria defined in the archive, but permit us to build hard instances around the satisfiability

limit. Even with these protocols, many instances generated are either easy (less than 2 seconds of cpu time), or too difficult (more than one hour of cpu time). In Table 3, we report results for those instances on which a significant difference has been observed among the DVOs tested. The cpu time limit was put to one hour.

Table 3. A sample of RLFAPs solved on a PC PentiumII 300MHz under Linux.

	scen11-01234 (sat)		scen06-012 (unsat)		scen02-24 (sat)		scen02-25 (unsat)	
	#nodes	time	#nodes	time	#nodes	time	#nodes	time
dom/futdeg	6,019	8.43	—	> 1 h.	31,308,876	2,296.61	—	> 1 h.
H_1_dom_+	21,156	29.68	41	0.40	663	0.32	—	> 1 h.
H_1_dom/futdeg_+	—	> 1 h.	41	0.41	—	> 1 h.	11,668	10.18
H_1_dom_x	12,517	16.99	—	> 1 h.	677	0.32	—	> 1 h.
H_1_dom/futdeg_x	226,011	337.55	41	0.41	—	> 1 h.	8,529	6.99

The two non trivial original instances are graph10 and scen11. On scen11, we can see that H1_dom/futdeg_+ and H_1_dom/futdeg_× do not perform as well as the others, especially H1_dom/futdeg_+, which does not solve the problem under the time limit. None of the five tested DVOs could solve graph10 in one hour. It can be pointed out that dom/deg could solve it in 2.84 seconds.

If we take a look at the instances derived from scen01 to scen10, we can find three of them on which significant differences can be observed. scen06-012, which is inconsistent, is almost trivially solved by three of the H_1 heuristics. dom/futdeg and H_1_dom_× could not solve it in one hour. Interestingly, we could find and prove the optimal number of frequencies that can be removed in scen02. Indeed, scen02-24 was found satisfiable, and scen02-25 inconsistent. On scen02-24, H_1_dom_+ and H_1_dom_× quickly find a solution, while the others are very slow or out of the limit. On scen02-25, this is exactly the opposite since H1_dom/futdeg_+ and H_1_dom/futdeg_× are the only two able to prove inconsistency in the allowed time.

If no conclusion can be drawn on a so small number of pertinent instances, we can at least give some observations. First, we see that dom/futdeg is significantly outperformed on those real problems. This confirms results of Section 4.1. Second, it seems that H1_dom/futdeg_+ and H_1_dom/futdeg_× are better on inconsistent problems, and H_1_dom_+ and H_1_dom_× on satisfiable ones. But, more extensive tests should be run to draw definite conclusions since this last fact doesn't appear as clearly as that on our experiments on random CSPs.

4.3 What about the second level?

To get an idea on the behavior of the second level instantiations (H_2), we have conducted experiments on the class presented in Figure 2. In Figure 5, we show the results of the first and the second level DVOs using (H_1_dom/futdeg_× and H_2_dom/futdeg_×). The other instantiations have a similar behavior. We can

see that the second level DVO improves the first one with respect to constraint checks (right hand curve). Regarding cpu time (left hand curve), the first level is slightly better than the second for instances with $N \leq 220$. When the number of variables is greater ($N = 230$), the second level DVO also becomes better wrt cpu time. These results are promising and show the feasibility of the multi-level approach with $k > 1$.

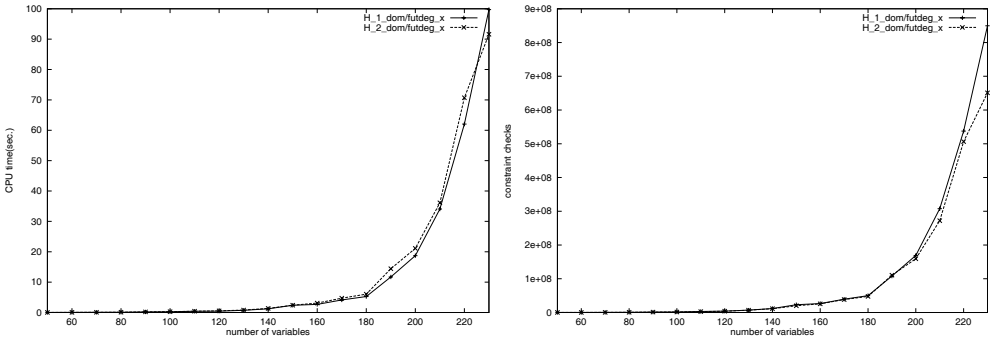

Fig. 5. $< N, D = 10, C = 5 * N/2, T_{crit} >$: cpu time & number of constraint checks.

5 Future Work

This work opens some perspectives on solving hard constraint satisfaction problems. As a future work, we plan to address the following directions:

- In the formulation given above, the weight associated with each constraint is based on simple syntactical properties of the variables which require a low computational cost. It would be interesting to investigate how constraint semantics can be integrated. Indeed, whereas the tightness of the constraint is very expensive to evaluate in random CSPs (many constraint checks to perform), it can sometimes be estimated cheaply in real-world problems where we have knowledge about the semantics. For example, it is known without any computation that if a constraint R_{ij} sets that X_i and X_j are bound by the relation "=", it will be much tighter than if it sets that they are bound by the relation "$\neq$".
- The results obtained on the second level instantiations are very promising and need to be systematically investigated. We believe that on larger and harder CSPs, greater levels DVOs might pay off.

6 Conclusion

The contribution of this paper is twofold. On the one hand, a general formulation of dynamic variable ordering heuristics has been proposed. It admits numerous advantages,

- the constrainedness of a given variable is computed without any constraint check, thanks to simple syntactical properties,
- it takes advantage of the neighborhood of the variable, with the notion of distance as a parameter,
- it can be instantiated to different known variable ordering heuristics,
- it is possible to use other functions to measure the weight of a given constraint.

On the other hand, we have shown that when instantiating the general formula with known VOs (`dom` and `dom/futdeg`), and a distance 1 for the neighborhood involved, we obtain significant improvements over the most efficient known DVOs.

References

1. C. Bessière and J.C. Régin. MAC and combined heuristics: two reasons to forsake FC (and CBJ?) on hard problems. In *Proceedings CP'96*, pages 61–75, Cambridge MA, 1996.
2. D. Brélaz. New methods to color the vertices of a graph. *Communications of the ACM*, 22:251–256, 1979.
3. C. Cabon, S. de Givry, L. Lobjois, T. Schiex, and J.P. Warners. Radio link frequency assignment. *Constraints*, 4:79–89, 1999.
4. R. Dechter and I. Meiri. Experimental evaluation of preprocessing techniques in constraint satisfaction problems. In *Proceedings IJCAI'89*, pages 271–277, Detroit MI, 1989.
5. E.C. Freuder. A sufficient condition for backtrack-free search. *Journal of the ACM*, 29(1):24–32, Jan. 1982.
6. D. Frost and R. Dechter. Look-ahead value ordering for constraint satisfaction problems. In *Proceedings IJCAI'95*, pages 572–578, Montréal, Canada, 1995.
7. P.A. Geelen. Dual viewpoint heuristics for binary constraint satisfaction problems. In *Proceedings ECAI'92*, pages 31–35, Vienna, Austria, 1992.
8. I.P. Gent, E. MacIntyre, P. Prosser, B. Smith, and T. Walsh. An empirical study of dynamic variable ordering heuristics for the constraint satisfaction problem. In *Proceedings CP'96*, pages 179–193, Cambridge MA, 1996.
9. R.M. Haralick and G.L. Elliott. Increasing tree seach efficiency for constraint satisfaction problems. *Artificial Intelligence*, 14:263–313, 1980.
10. J.N. Hooker and V. Vinay. Branching rules for satisfiability. *Journal of Automated Reasoning*, 15:359–383, 1995.
11. B. Nudel. Consistent-labelling problems and their algorithms: Expected-complexities and theory-based heuristics. *Artificial Intelligence*, 21:135–178, 1983.
12. P. Prosser. An empirical study of phase transition in binary constraint satisfaction problems. *Artificial Intelligence*, 81:81–109, 1996.

13. D. Sabin and E.C. Freuder. Contradicting conventional wisdom in constraint satisfaction. In *Proceedings PPCP'94*, Seattle WA, 1994.
14. B. Smith and S.A. Grant. Trying harder to fail first. In *Proceedings ECAI'98*, pages 249–253, Brighton, UK, 1998.
15. B.M. Smith. The Brélaz heuristic and optimal static orderings. In *Proceedings CP'99*, pages 405–418, Alexandria VA, 1999.
16. R. Zabih. Some applications of graph bandwith to constraint satisfaction problems. In *Proceedings AAAI'90*, pages 46–51, Boston MA, 1990.