MAC and Combined Heuristics: Two Reasons to Forsake FC (and CBJ?) on Hard Problems

Christian Bessière[1] and Jean-Charles Régin[2]

[1] LIRMM (UMR 5506 CNRS), 161 rue Ada, 34392 Montpellier cedex 5, France
e-mail: bessiere@lirmm.fr
[2] ILOG S.A., 9 rue de Verdun BP 85, 94253 Gentilly Cedex, France
e-mail: regin@ilog.fr

Abstract. In the last twenty years, many algorithms and heuristics were developed to find solutions in constraint networks. Their number increased to such an extent that it quickly became necessary to compare their performances in order to propose a small number of "good" methods. These comparisons often led us to consider FC or FC-CBJ associated with a "minimum domain" variable ordering heuristic as the best techniques to solve a wide variety of constraint networks.
In this paper, we first try to convince once and for all the CSP community that MAC is not only more efficient than FC to solve large practical problems, but it is also really more efficient than FC on hard and large random problems. Afterwards, we introduce an original and efficient way to combine variable ordering heuristics. Finally, we conjecture that when a good variable ordering heuristic is used, CBJ becomes an expensive gadget which almost always slows down the search, even if it saves a few constraint checks.

1 Introduction

Constraint satisfaction problems (*CSPs*) occur widely in artificial intelligence. They involve finding *values* for problem *variables* subject to *constraints* on which combinations are acceptable. For simplicity we restrict our attention here to *binary* CSPs, where the constraints involve two variables.

Binary constraints are binary relations. If a variable i has a *domain* of potential values D_i and a variable j has a domain of potential values D_j, the constraint on i and j, R_{ij}, is a subset of the Cartesian product of D_i and D_j. If the pair of values a for i and b for j is acceptable to the constraint R_{ij} between i and j, we will call the values *consistent* (with respect to R_{ij}). Asking whether a pair of values is consistent is called a *constraint check*.

The entity involving the variables, the domains, and the constraints, is called *constraint network*. Any constraint network can be associated to 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. $\Gamma(i)$ represents the set of nodes sharing an edge with the node i.

In the last twenty years, many algorithms and heuristics were developed to find solutions in constraint networks [16], [21], [22]. Their number had increased to such an extent that it quickly became necessary to compare their performances in order to designate some of them as being the best methods. In the recent years, many authors worked in this way [23], [5], [10], [1]. The general inference drawn from these works is that forward checking [16] (denoted by FC) or FC-CBJ (CBJ: conflict directed backjumping [22]) associated with a "minimum domain" variable ordering heuristic is the most efficient strategy to solve CSPs. (It has been so repeated that hard problems are often considered as those that FC-CBJ cannot solve [24]). This can be considered as a surprising conclusion when we know that the constraint programming community uses full arc consistency at each step of the search algorithms [32] and claims that it is the only practicable way to solve large real world problems in reasonable time [26]. The contradiction comes from the fact that in the CSP community, the sample problems used for the comparisons were often very particular (especially small or easy [16], [21], [23]), and the way the algorithms were compared was sometimes incomplete (no procedure maintaining full arc consistency involved in the comparisons [5], [10]) or unsatisfactory [1]. But, this apparent contradiction did not give rise to other questions or comments than Sabin and Freuder's paper [28], in which it was pointed out that a procedure Maintaining Arc Consistency during the search (*MAC*) could outperform FC on random problems around the cross-over point.

In this paper, we try to convince the reader that MAC is not only more efficient than FC to solve large practical problems, but it is also really more efficient than FC on hard and large random problems. Afterwards, we introduce an original[3] way to really *combine* different variable ordering heuristics (instead of just using a secondary heuristic to break ties in the main one) and show its efficiency. Finally, we conjecture that when a good variable ordering heuristic is used, CBJ becomes an expensive gadget which almost always slows down the search, even if it saves a few constraint checks.

The paper is organized as follows. Section 2 contains an overview of the main previous works on algorithms and heuristics to solve CSPs. Section 3 describes the instance generator and the experimental method used in the rest of the paper. We show the good behavior of MAC in Sect. 4. The new way to combine variable ordering heuristics is presented in Sect. 5. Section 6 shows that CBJ loses its power when high level look-ahead is performed during the search. Finally, Sect. 7 summarizes the work presented in this paper.

2 Previous Work

It has been noted by several authors (e.g. [15]) that there are four choices to be made when searching solutions in constraint networks: what level of filtering to do, which variable to instantiate next, what value to use as the instantiation, what kind of look-back scheme to adopt.

In fact, a wide part of the CSP community has been working for twenty years to answer these questions.

To the question of the level of filtering to perform before instantiating a variable, many papers concluded that *forward-checking* (FC) is the good compromise between the pruning effect and the amount of overhead involved ([16], [20], [19], [1]).

[3] This approach is original in the sense that it has never been published before. The only presentation we know of such an approach is given in [2].

It has been shown for a long time that in constraint networks, the order in which the variables are instantiated strongly affects the size of the search space explored by backtracking algorithms. In 1980, Haralick and Elliot already presented the "fail first principle" as a fundamental idea [16]. Following this, a variety of static variable ordering heuristics (*SVO*) were proposed to order the variables such that the most constrained variables are chosen first (thus respecting the Haralick and Elliot's principle). They calculate once and for all an order, valid during all the tree search, according to which variables will be instantiated. They are usually based on the structure of the constraint graph. The *minimum width* ordering (`minw`) is an order which minimizes the width of the constraint graph [9]. The *maximum degree* heuristic (`deg`) orders the variables by decreasing number of neighbors in the constraint graph [5]. The *maximum cardinality* ordering (`card`) selects the first variable arbitrarily, then, at each stage, selects the variable that is connected to the largest set of already selected variables [5]. The heuristic proposed by Haralick and Elliot to illustrate their principle was a dynamic variable ordering heuristic (*DVO*[4]). They proposed the *minimum domain* (`dom`) heuristic, which selects as the next variable to be instantiated a variable that has a minimal number of remaining values in its domain. It is a dynamic heuristic in the sense that the order in which variables are instantiated can vary from branch to branch in the search tree. Papers discussing variable ordering heuristics quickly found that DVO is generally better than SVO. More precisely, `dom` has been considered as the best variable ordering heuristic ([27], [15], [5]).

The question of the choice of the value to use as an instantiation of the selected variable did not catch as much researchers' attention as variable ordering. It has been explored in [15] or [6], but without producing a simple generic method proven efficient and usable in any constraint network. Even the *promise* selection criterion of Geelen [14] did not attract FC users.

The question of the kind of look-back scheme to adopt had remained an open question for a long time. Different approaches had been proposed, but none had been elected as the best one (e.g. *learning* [4], *backjumping* [13], *backmarking* [12], etc.). This state of things seems to have finished with the paper of Prosser [22], which presented *conflict-directed backjumping* (CBJ). Indeed, Prosser showed in [23] that the hybrid algorithm FC-CBJ is the most efficient algorithm (among many hybrid algorithms) to find solutions in various instances of the zebra problem.

That's why, for a few years, FC-CBJ associated with the `dom` DVO (denoted by FC-CBJ-`dom`) has been considered as the most efficient technique to solve CSPs (naturally following the FC domination of the eighties). Moreover, the numerous papers studying "really hard problems" ([24], [30], [7]) often take the implicit definition: "an hard problem is a problem hard to solve with FC-CBJ-`dom`".

Recent Work. Recently, some authors, not satisfied at all by the conclusion of the story of search algorithms in CSPs, tried to improve this winner. This leads to the paper of Frost and Dechter [11], which reveals two important ways to overcome the classical FC-CBJ-`dom` algorithm.

[4] The origin of the name DVO is in [10] to denote what we will call here `dom`. We use DVO in its general meaning, as it is proposed in [1].

First, the `dom` DVO is not as perfect as it seems. When several domains have the same minimal size, the next variable to be instantiated is arbitrarily selected. When the constraint graph is sparse, many useful information on its structure is lost by `dom`, which does not deal with the constraint graph. In [11], a solution to these shortcomings is proposed by using the `dom+deg` DVO: it consists of the `dom` DVO in which ties are broken[5] by choosing the variable with the highest degree in the constraint graph. Frost and Dechter underlined that "this scheme gives substantially better performance than picking one of the tying variables at random".

Second, in FC-CBJ-`dom`, once a variable is selected for instantiation, values are picked from the domain in an arbitrary fixed order (usually values are arbitrarily assigned a sequence number and are selected according to this sequence). In [11], Frost and Dechter presented various domain value ordering heuristics (*LVO* for look-ahead value ordering) and experimentally showed that the *min-conflicts*[6] (`mc`) LVO is the one which improves the most the efficiency of FC-CBJ-`dom+deg` (denoted by FC-CBJ-`dom+deg-mc`).

Another, quite different way to improve search by reordering values (or variables and values) after a dead-end has been presented in [17]. Its features making it especially suitable to solve real world problems, we do not discuss it here.

3 A Few Words About the Experiments

Before starting the experimental comparisons between different algorithms, we say a few words about the experimental method we chose.

When we want to work on random problems, the first step is to choose an instance generator. The characteristics of the generated problems will depend on the generator used to create them. The CSP literature has presented several generators, always involving four parameters: N the number of variables, D the common size of all the initial domains, and two other parameters concerning the density of the constraint graph and the tightness of the chosen constraints. Early generators often used a probability p_1 that a constraint exists between two variables, and a probability p_2 that a value pair is forbidden in a given constraint. The number of different networks that could be generated with the same four parameters $\langle N, D, p_1, p_2 \rangle$ was really huge. Networks with quite different features (e.g. a network with a complete constraint graph and one with only one constraint) could be generated with the same set of parameters. One of the consequences of this fact was that a very large number of instances must be solved to predict the behavior of an algorithm with a good statistical validity.

Hence, a new generation of instance generators appeared (beginning with [18]), which replaced the probability p_1 to have a constraint between two variables by a fixed number C of constraints [24]. In the same way, p_2 can be replaced by a number T of forbidden value pairs [11]. In [30], p_1 and p_2 are still used, but they represent "proportions"

[5] The idea of breaking ties in SVOs and DVOs had been previously proposed in [33].

[6] *min-conflicts* considers each value in the domain of the selected variable and associates with it the number of values in domains of future variables with which it is not compatible. The values are then affected to the selected variable in increasing order of this count. This is in fact the first LVO presented in [14, page 32].

and not probabilities (i.e. if N=20 and p_1=0.1, the number of generated constraints is exactly $0.1 * (20 * 19)/2 = 19$). This new method generates more homogeneous networks and then, it is not necessary to solve a huge number of networks for each set of parameters. Nevertheless, a particular care must be taken in order to generate networks with a uniform distribution. Specifically, the distribution must be as follows: out of all possible sets of C variable pairs choose any particular set with uniform probability, and for each constrained pair out of all possible sets of T value pairs choose any particular set with uniform probability[7]. We need an algorithm that generates uniform random permutations of p elements selected among k elements without repetition. Essentially, this is just choosing which of the k elements will be the first, which of the remaining $k-1$ elements will be the second, and so forth.

When we want to perform experiments on randomly generated networks, and when the instance generator has already been chosen, a second step is to select the sets of parameters that will be used to illustrate the behavior of the algorithms tested. Each set of parameters $\langle N, D, C, T\rangle$ determines the type of the networks generated: N variables each having a domain of size D, C constraints out of the $N*(N-1)/2$ possible, and T forbidden value pairs in each constraint among the $D*D$ possible. In this paper, we did not want to make a complete study of which sets of parameters to use to illustrate our claims. Thus, we decided to use sets of parameters already presented in the literature, and quite well-known. We chose the problems presented in [11] (some of them were already used in [10]) and some of the most famous experiments used by Smith and Grant ([30], [31], [29]). In certain experiments, we propose some variations in the parameters (for example, increasing domain size to show the behavior of the algorithms on networks with larger domains). But, when we vary the density (C) or the domain size (D), we want to keep the networks generated as close as possible to the *cross-over point* (set of parameters for which approximately 50% of the problems are satisfiable and 50% are not). So, T is moved in order to stay at the value "T_{co}" which produces 50% satisfiable problems and 50% unsatisfiable. When for given values of N, D and C no value of T (which is an integer) produces exactly 50% satisfiable problems we always take as T_{co} the smallest value for which the number of unsatisfiable problems is greater than 50%. These variations of the distance between T_{co} and the effective cross–over point explain the serrated look of some of the curves reported below. The size of the problems tested in such cases is often rather small, because each point of the curves given (see Fig. 2, 3, 5) requires to solve a large number of networks just to find the right value T_{co}.

In the following sections we report different kinds of measures of performances for the algorithms tested. First, we often present what we call "number of constraint checks". The classical "number of constraint checks" measure is well-adapted for algorithms like FC, but presents some problems when used with MAC, which maintains lists where some of the past constraint checks are recorded. Hence, for MAC, what we name "number of constraint checks" is in fact the number of classical constraint checks plus the number of list checks it performs during the search. The second measure we use is cpu

[7] Prosser's generator [24] does not choose all the possible sets of C constraints with a uniform probability. Frost and Dechter's generator, while being better than Prosser's one, is not completely uniform [8]. Although it is not extensively described, Smith's generator seems to be uniform [29] (while Smith and Grant's one is not [30]).

time, and the third one is the number of backtracks performed, i.e. the number of times the algorithm goes backwards in the search tree.

In all the tables below, we generated and solved 100 instances for each tested Frost and Dechter's set of parameters. In all the figures (curves), we limited this number to 50 instances for each tested value of the varying parameter.

We always report mean performances on the number of instances solved for a set of parameters. Indeed, we think that reporting the median cost is questionable when the set of parameters is near the cross-over point: unsatisfiable instances are generally harder to solve than satisfiable ones, so the median will appear in a region where few problems fall into, involving a low representativity of this measure. In the extreme case, we can imagine a set of parameters for which 50 problems are found satisfiable in 1 second and 50 are found unsatisfiable in 10 seconds: what is the median cost of this experiment?

LVOs being outside the scope of the present paper, we just checked that `mc` was a significant improvement in our experiments compared to the versions of the algorithms written without LVO. Hence, in the results presented in the next sections, `mc` has always been used, even if on some instances the *promise* LVO of Geelen can have a slight more interesting behavior than `mc`. However, after a very rough comparison, we could not select a winner.

Finally, we want to point out that the programs used to perform the experiments of this paper are available via the ftp site `ftp.lirmm.fr`.

4 MAC is Better than FC-CBJ

We said in Sect. 2 that FC-CBJ is considered as the best algorithm to find solutions in constraint networks. In fact, in the papers that have compared algorithms with different levels of filtering during search and that have concluded that FC performs the right amount of filtering it is often specified that this claim is stated with respect to the tested problems [16], [20], [23]. The tested problems were often the n-queens, very small random problems not necessarily chosen in the phase transition, or the zebra problem. Therefore, we can conclude that on very easy or very small problems FC is probably the algorithm which performs the right amount of filtering (pure look-back algorithms are probably definitively overcome [23], [1]).

But, Dechter and Meiri already said that "it is conceivable that on larger, more difficult instances, intensive preprocessing algorithms may actually pay off" [5]. A first confirmation appeared in the paper of Sabin and Freuder [28], in which they showed that MAC can outperform FC on hard instances of CSPs. The good performances of MAC on large radio link frequency assignment problems (where FC was thrashing) provide another confirmation [3].

Recently, Smith agreed that "exceptionally hard problems ought more properly to be called problems which the particular search algorithm we are using finds exceptionally hard". This led her and Grant to study the behavior of MAC on problems found exceptionally hard with FC-`dom` [31]. Their conclusion is that "in most cases, the MAC algorithm can show that the problem is arc inconsistent, and so detects that it is insoluble without searching it".

Finally, [1] is the only paper which clearly gives the advantage to FC-CBJ against algorithms performing arc consistency at each node of the search tree after an experimentation on non-easy problems. But, after discussion with Bacchus, it appears that in his paper, the algorithm that performs arc consistency at each stage of the search uses a kind of AC-0 algorithm, i.e. an AC-1 algorithm which does not take care of the structure of the constraint graph, checking all the variable pairs, as if the network was always a complete graph. So, we cannot take these results into account.

We showed in Sect. 2 that the behavior of FC-CBJ can be improved by using a DVO proposed by Frost and Dechter, `dom+deg`, and by using a good LVO as `mc`. In this section, we will show that, even associated to the `dom+deg` DVO and the `mc` LVO, FC-CBJ can no longer be considered as the best algorithm to solve CSPs. We will experimentally show that FC has a too weak pruning effect to be the most efficient on relatively hard problems. A search procedure as MAC, with a more intensive filtering mechanism, is more efficient to find solutions on hard and large problems, in which the overhead due to arc consistency is outweighed by its gain.

The experiments of this section are limited to the comparison of FC-CBJ-`dom+deg-mc` and MAC-`dom+deg−mc`. FC-CBJ-`dom+deg−mc` is the algorithm stated to be the best in Sect. 2. MAC-`dom+deg−mc` is here a classical MAC procedure [28] in which the arc consistency algorithm used is AC-7 [3]. The DVO and the LVO used are the same in the two algorithms.

Table 1. FC-CBJ-`dom+deg−mc` and MAC-`dom+deg−mc` performances on problems generated with Frost and Dechter's sets of parameters [11]. "arc-inc" in the backtrack ratio column means that all the problems generated for a given set of parameters were arc-inconsistent, implying an infinite ratio (MAC detects arc-inconsistency without any backtrack).

	Parameters	#constraint checks			cpu seconds			#backtracks
	$N, D, C, T/D*D$	FC-CBJ	MAC	ratio	FC-CBJ	MAC	ratio	ratio
#1	35,6,501,4/36	506,265	330,717	1.53	6.83	2.66	2.56	7.45
#2	35,9,178,27/81	248,414	156,131	1.59	3.26	1.00	3.25	14.29
#3	50,6,325,8/36	412,505	152,197	2.71	5.81	1.29	4.50	17.35
#4	50,20,95,300/400	565,330	273,537	2.07	7.11	1.62	4.39	37.02
#5	100,12,120,110/144	243,766	15,709	15.52	3.79	0.14	25.99	870.28
#6	125,3,929,1/9	271,557	44,862	6.05	4.51	1.52	2.96	12.08
#7	250,3,391,3/9	19,636	2,686	7.31	0.55	0.05	11.26	arc-inc
#8	350,3,524,3/9	820,368	3,558	230.53	31.04	0.07	476.31	arc-inc
#9	350,3,2292,1/9	426,713	51,176	8.34	9.40	4.35	2.16	9.68

A first set of experiments (in which parameters are taken from [11]) is given in Table 1. The columns "ratio" represent how much MAC-`dom+deg−mc` was better than FC-CBJ-`dom+deg−mc` with respect to the associated measure (mean number of constraint checks, mean cpu time, mean number of backtracks). On this first set of experiments we can stress that the ratio of the number of constraint checks is less advantageous for MAC than the cpu time ratio. An explanation is that, for any search algorithm that performs some look-ahead filtering, each backtrack point involves restoring the previous state, and running again the variable-value selection. In spite of being free of any constraint check, this process is time consuming. MAC-`dom+deg−mc` being better and better than FC-CBJ-`dom+deg−mc` in number of backtracks (see the last column of Table 1) saves a lot of time in addition to the time saved by constraint checks savings. Anyway, MAC-dom+deg−mc significantly overcomes FC-CBJ-dom+deg−mc on these problems.

We performed a second set of experiments on the now classical $\langle 50, 10, 0.1, p_2\rangle$ set of parameters of Smith and Grant [30], [31]. In our formalism, it consists of the set of parameters $\langle 50, 10, 123, T\rangle$. Figure 1 gives the results, which corroborate those obtained in Table 1. MAC is slightly worse than FC-CBJ on easy problems (under– and over–constrained) while being much better around the cross–over point.

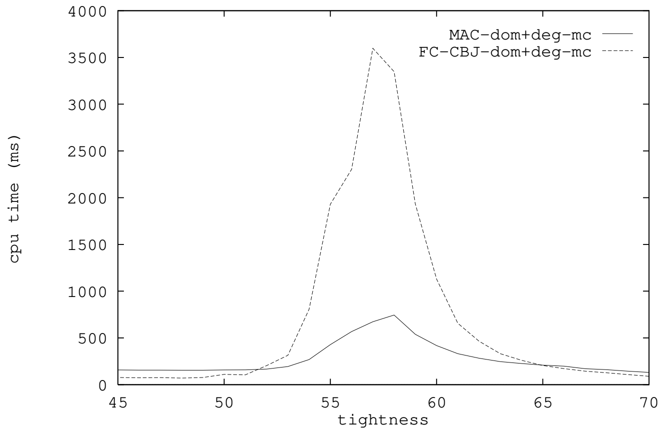

Fig. 1. FC-CBJ-dom+deg−mc and MAC-dom+deg−mc time performances on the $\langle 50, 10, 123, T\rangle$ experiment of Smith and Grant [30].

Frost–Dechter and Smith–Grant's parameters being limited to small domain sizes, we took the $\langle 50, 20, 95, 300\rangle$ set of parameters in Frost and Dechter's sample, and changed domain sizes while keeping N and C fixed at 50 and 95 respectively, T varying to stay at T_{co} (see Fig. 2-(left)). We note that the more D grows, the more MAC-dom+deg-mc outperforms FC-CBJ-dom+deg−mc, going from 3 times faster when D is smaller than 10 to 26 times faster when D reaches 40.

Finally, we wanted to see the behavior of MAC when the density of the constraint graph increases. Figure 2-(right) presents the FC-CBJ-dom+deg−mc to MAC-dom+deg-mc cpu time ratio when the number C of constraints increases in the $\langle 30, 10, C, T_{co}\rangle$ set of parameters. MAC efficiency increases till the constraint graph contains approximately a third of the possible number of constraints. Afterwards, FC-CBJ becomes less and less worse as the number of constraints grows till the complete graph[8]. This phenomenon was pointed out by Sabin and Freuder.

[8] These cpu times ratios, despite showing the advantage of MAC, do not go higher than 3. The reason is that 30 variables is not enough to generate hard problems on which MAC would show its real efficiency.

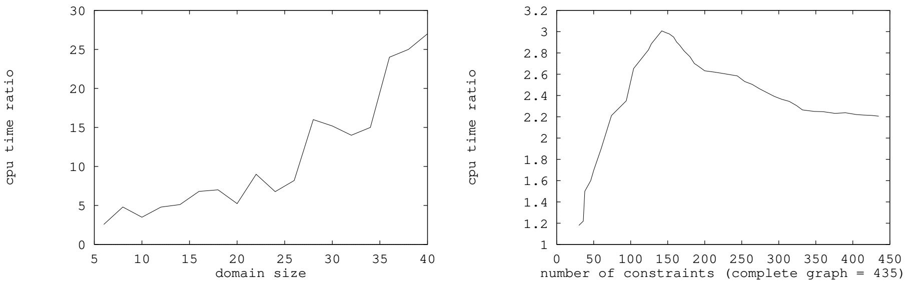

Fig. 2. FC-CBJ-dom+deg−mc to MAC-dom+deg−mc cpu time ratio on the $\langle 50, D, 95, T_{co}\rangle$ (left), D growing from 6 to 40; and on the $\langle 30, 10, C, T_{co}\rangle$ (right), where C grows from 29 to 435 (complete graph).

5 Combined DVOs: `dom/deg`

In Sect. 2 we presented different kinds of variable ordering heuristics and said that the `dom` DVO had been considered for a long time as the best one. However, when the constraint graph is sparse, many useful information is lost by this heuristic while it is caught by the SVOs based on the structure of the constraint graph.

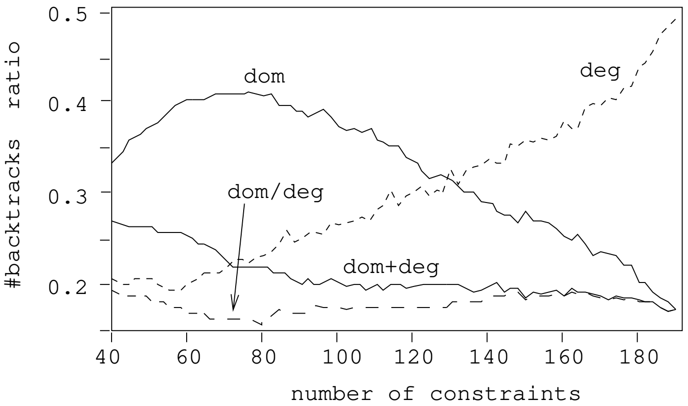

Fig. 3. Different variable ordering heuristics tested with MAC on the $\langle 20, 10, C, T_{co}\rangle$, where C grows from 40 to 190 (complete graph). Each graph represents the ratio of the mean number of backtracks of MAC with the given heuristic to the sum of the mean number of backtracks of the four algorithms tested (absolute results would have given unreadable graphs since the difficulty of the problems significantly grows when C grows).

In Fig. 3, where random problems with increasing density are solved by different versions of MAC (i.e. using different variable ordering heuristics[9]), it is shown that dom can be a very poor heuristic at low densities, while deg is very efficient on the same problems. Inversely, when the constraint graph becomes dense, deg goes blind while dom becomes clever. dom+deg, which breaks ties in dom by using the degree of the tying variables is shown in this Fig. 3 to improve dom on problems where it was bad. But, in dom+deg, the size of the domains clearly have the main influence on the ordering, the degree of variables being only used in cases where ties are found. To avoid this drawback, which prevents dom+deg from being as good as deg in sparse constraint networks, we propose to really combine dom and deg to obtain a new DVO in which deg is as influent as dom. This new DVO, dom/deg, selects as the next variable to be instantiated a variable that has the smallest ratio: size of the remaining domain to degree of the variable (i.e. a variable v minimizing $|D_v|/|\Gamma(v)|$). In Fig. 3 we have a first idea of its behavior: it has the behavior of dom+deg in networks where dom was good, and the one of deg in networks where deg was better. These first results being promising, we give in Table 2 and Fig. 4 a more complete set of experiments in which we compare MAC-dom+deg−mc and MAC-dom/deg−mc. Once again, the characteristics of the problems tested are taken from [11] and [30]. Results obtained in Table 2 show that with small domain sizes ($D < 10$) the two DVOs have similar behaviors, with a little advantage for dom/deg. The difference is slightly perceptible on the $\langle 35, 9, 178, 27\rangle$ and the $\langle 100, 12, 120, 110\rangle$ experiments. It is significant on the $\langle 50, 20, 95, 300\rangle$. This is explained by the fact that when D is very small, dom/deg and dom+deg are quite similar criteria, the variations of $|D_v|$ –for a given variable v– dominating those of $|\Gamma(v)|$ in dom/deg.

Table 2. MAC-dom+deg−mc versus MAC-dom/deg−mc. Only ratios are given (real values can be obtained from these ratios and Table 1). Values greater than 1 mean dom/deg is better, values smaller than 1 mean dom+deg is better.

	Parameters	ratios		
	$N, D, C, T/D*D$	#constraint checks	time	#backtracks
#1	35,6,501,4/36	1.00	1.01	1.35
#2	35,9,178,27/81	1.24	1.23	1.63
#3	50,6,325,8/36	1.11	1.12	1.53
#4	50,20,95,300/400	3.45	3.05	7.01
#5	100,12,120,110/144	1.11	1.10	3.20
#6	125,3,929,1/9	1.02	0.98	1.42
#7	250,3,391,3/9	1.00	1.00	arc-inc
#8	350,3,524,3/9	1.00	1.00	arc-inc
#9	350,3,2292,1/9	1.00	0.97	1.56

To be convinced that dom/deg is more advantageous when domains are larger, we tested the two heuristics on instances of problems with increasing domain size. In Fig. 5-(left), the domain sizes vary while N and C are fixed to 50 and 95 respectively. T changes so that problems are always on the cross-over point. The more the size of the domains increases, the more MAC-dom/deg−mc overcomes MAC-dom+deg−mc, going from once to 7 times faster when D grows from 6 to 40. Furthermore, Prosser (per-

[9] The LVO used is mc in all these versions. Without LVO, we remarked that the differences increase between good and bad algorithms.

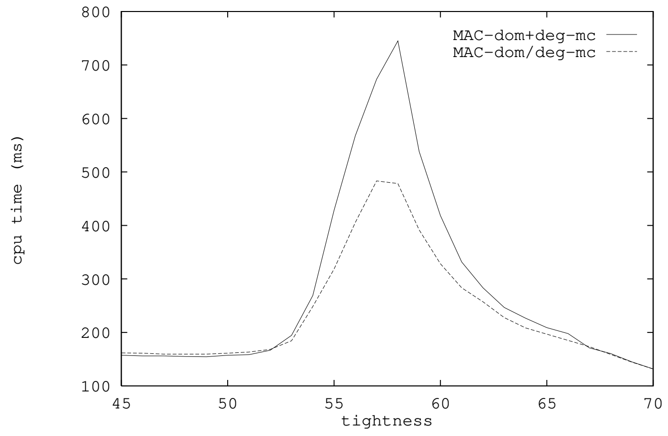

Fig. 4. MAC-dom+deg−mc and MAC-dom/deg−mc on the $\langle 50, 10, 123, T\rangle$.

sonal communication) has pointed out that when initial domain sizes are not all equal, dom (or dom+deg) can be fooled by these initial differences. We suppose that in these cases dom/deg would be even more interesting.

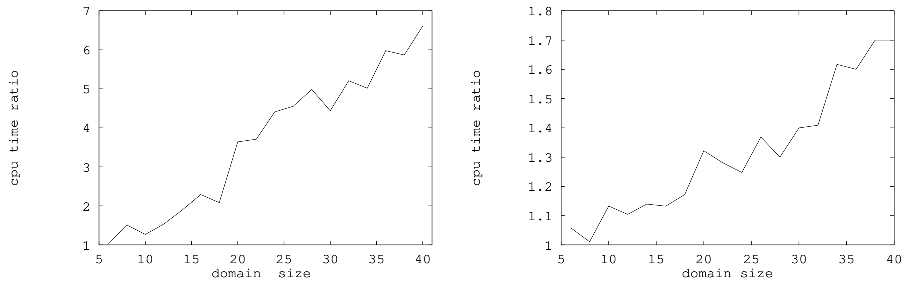

Fig. 5. MAC-dom+deg−mc versus MAC-dom/deg−mc on the $\langle 50, D, 95, T_{co}\rangle$ (left), and MAC-CBJ-dom/deg−mc versus MAC-dom/deg−mc on the $\langle 50, D, 95, T_{co}\rangle$ (right).

Thus, we can conclude that combining different DVOs is a promising approach. We have tested other combined DVOs not presented in this paper. The one that can be named dom/card, in which the number of previously assigned neighbors of the variable replaces the total number of neighbors in the ratio seems to be quite worse than dom/deg (when card alone was considered as a better SVO than deg alone [5]). On the other hand, when the ratio involves the number of not yet assigned neighbors of the variable, the performances are roughly similar to those obtained with dom/deg, sometimes better, sometimes worse. dom/deg has been also implemented in FC-CBJ. We saw an improvement with respect to dom+deg, but smaller than the one observed on MAC.

6 CBJ Becomes Useless

We have shown that using MAC instead of FC as the filtering scheme was worthwhile on hard and large problems. If we follow the evolution of FC in FC-CBJ we should now use MAC-CBJ [25]. But, let us recall a sentence found in [16]: “Look ahead to the future in order not to worry about the past”. In fact, some authors remarked that if we use a good variable ordering heuristic “CBJ is unlikely to generate large backjumps, and its savings are likely to be minimal” because “variables that have conflicts with past assignments are likely to be instantiated sooner” [1]. In [30], Smith and Grant said that “for most problems, the ordering given by dom ensures that chronological backtracking usually results in backtracking to the real culprit for a failure, so that informed backtracking does not add very much”.

These statements, done in the case of FC-dom were probably too optimistic since a non negligible number of problems are easily solved by FC-CBJ-dom when FC-dom is thrashing [31]. But, as it is suggested by Haralick and Elliot’s sentence, the more we will perform look-ahead, the less we will have to worry about looking back. CBJ was a strong improvement on BT (simple backtracking), FC-CBJ can be an improvement on FC on hard problems, MAC-CBJ cannot simply be claimed to be an improvement on MAC. In [31], while a lot of problems were found on which FC-CBJ-dom outperformed FC-dom by at least one order of magnitude, only one instance was found on which MAC-CBJ-dom significantly outperformed MAC-dom. If we consider now the DVO dom/deg in place of dom, there are even more reasons to think that CBJ becomes useless (since dom/deg has been shown smarter than dom). Furthermore, the more the amount of filtering involved in a search procedure is high, the more the overhead caused by CBJ is heavy [25]. CBJ was cheap to incorporate in BT, it was not prohibitive in FC, but it palpably slows down the search in MAC. Hence, a significant number of constraint checks must be saved to outweigh this overhead.

Table 3. MAC-CBJ-dom/deg−mc versus MAC-dom/deg−mc.

	Parameters	ratios		
	$N, D, C, T/D * D$	#constraint checks	time	#backtracks
#1	35,6,501,4/36	0.99	1.17	0.99
#2	35,9,178,27/81	0.99	1.32	0.99
#3	50,6,325,8/36	0.99	1.21	0.99
#4	50,20,95,300/400	0.99	1.33	0.99
#5	100,12,120,110/144	0.98	0.99	0.96
#6	125,3,929,1/9	0.97	1.08	0.96
#7	250,3,391,3/9	arc-inc	arc-inc	arc-inc
#8	350,3,524,3/9	arc-inc	arc-inc	arc-inc
#9	350,3,2292,1/9	0.64	0.70	0.61

Table 3 gives the comparison of MAC-CBJ-dom/deg−mc and MAC-dom/deg-mc on the Frost and Dechter’s problems. On the problems #1 to #8 the result is easy to read: CBJ leads to a few constraint checks savings which are not sufficient to make good the loss of time. But, on the set of parameters #9, there is a significant gain for MAC-CBJ-dom/deg−mc. If we focus on the 100 instances which form this experiment we see that on 99 instances MAC-dom/deg−mc and MAC-CBJ-dom/deg−mc have almost the same behavior, solving the problem in less than 1 second with a number of backtracks smaller than 1000. But on one of the 100 instances MAC-dom/deg−mc needs 137 seconds and 41,639 backtracks to find a solution when MAC-CBJ-dom/deg−mc only needs 73 seconds and 20,069 backtracks. The mean performances are strongly influenced by this single instance which seems to match with the definition of "exceptionally hard problems" (*ehps*) [30]. Indeed, it occurs in the region where almost all problems are soluble (2547 constraints are necessary to be at the cross–over point in the $\langle 350, 3, C, 1\rangle$ set of parameters [11]). But, as opposed to the ehps found in [31], where FC-CBJ or MAC-CBJ were orders of magnitude faster than FC or MAC, MAC-CBJ-dom/deg−mc is only twice faster than MAC-dom/deg−mc on our ehp. Further experiments should probably be done to see whether ehps could be found on which MAC-CBJ-dom/deg−mc is really better than MAC-dom/deg−mc, though we did not find any in all the experiments we performed on smaller networks (50 variables).

Finally, we want to recall that the more domain sizes increase, the more the length of the jumps performed by CBJ decreases while CBJ time overhead increases (see the CBJ mechanism in [22]). This is confirmed in Fig. 5-(right) where MAC-CBJ-dom/deg−mc and MAC-dom/deg−mc are compared on the $\langle 50, D, 95, T_{co}\rangle$ experiment with increasing D.

Therefore, except on sparse networks with small domain sizes where more studies should be done, we think we can conclude that including CBJ in MAC-dom/deg−mc has more chances to slow down the search of at least 20% cpu time than to speed it up.

7 Conclusion

After a recall of the story of search procedures in constraint networks, this paper has shown how MAC can outperform FC and FC-CBJ on relatively hard and large randomly generated instances of constraint networks. Once the superiority of MAC has been proven, we have proposed a new kind of variable ordering heuristic, dom/deg, which really combines information on domain sizes and constraint graph structure. We proved its efficiency when compared with dom+deg, the most efficient previous heuristic. The total gain involved by these two techniques (MAC and dom/deg) is summarized in Table 4. The ratios of the mean performances of FC-CBJ-dom+deg−mc to the mean performances of MAC-dom/deg−mc are presented. The tested problems are again Frost and Dechter's problems. The benefit is always significant. Furthermore, we must have in mind that with larger domains the gain is greater and greater.

Therefore, we can conclude that on relatively hard and large instances of random problems, MAC and our new variable ordering heuristic are more efficient than FC-CBJ and classical dom or dom+deg DVOs.

Finally, we have shown in the last section that performing CBJ is almost always useless when combined with a procedure achieving as much look-ahead as MAC-dom/deg-mc. The time overhead is too heavy to be outweighed by the small number of constraint checks and backtracks saved.

Table 4. FC-CBJ-`dom+deg-mc` versus MAC-`dom/deg-mc`.

	Parameters	ratios		
	$N, D, C, T/D * D$	#constraint checks	time	#backtracks
#1	35,6,501,4/36	1.54	2.58	10.03
#2	35,9,178,27/81	1.97	3.98	23.33
#3	50,6,325,8/36	3.00	5.03	26.55
#4	50,20,95,300/400	7.13	13.38	259.64
#5	100,12,120,110/144	17.29	28.69	2785.20
#6	125,3,929,1/9	6.15	2.91	17.15
#7	250,3,391,3/9	7.31	11.26	arc-inc
#8	350,3,524,3/9	230.53	476.31	arc-inc
#9	350,3,2292,1/9	8.35	2.10	15.10

Acknowledgments. We want to thank Dan Frost and Stuart Grant for their help concerning instance generators, Olivier Dubois, who is at the origin of our comments on previous instance generators, Dan Sabin for the fruitful discussions we had on MAC, and Gene Freuder for his advice on ordering heuristics.

References

1. F. Bacchus and P. van Run. Dynamic variable ordering in csps. In *Proceedings CP'95*, pages 258–275, Cassis, France, 1995.
2. C. Bessière. Systèmes à contraintes évolutifs en intelligence artificielle. Phd thesis, LIRMM, University of Montpellier II, September 1992. (in French).
3. C. Bessière, E.C. Freuder, and J.C. Régin. Using inference to reduce arc consistency computation. In *Proceedings IJCAI'95*, pages 592–598, Montréal, Canada, 1995.
4. R. Dechter. Learning while searching in constraint satisfaction problems. In *Proceedings AAAI'86*, pages 178–183, Philadelphia PA, 1986.
5. R. Dechter and I. Meiri. Experimental evaluation of preprocessing algorithms for constraint satisfaction problems. *Artificial Intelligence*, 68:211–241, 1994.
6. R. Dechter and J. Pearl. Network-based heuristics for constraint-satisfaction problems. *Artificial Intelligence*, 34:1–38, 1988.
7. M.J. Dent and R.E. Mercer. Using local topology to model hard binary constraint satisfaction problems. In *Proceedings of the workshop –Studying and Solving Really Hard Problems–, CP'95*, pages 52–61, Cassis, France, 1995.
8. O. Dubois. Private communication, 1995.
9. E.C. Freuder. A sufficient condition for backtrack-free search. *Journal of the ACM*, 29(1):24–32, 1982.
10. D. Frost and R. Dechter. In search of the best constraint satisfaction search. In *Proceedings AAAI'94*, pages 301–306, Seattle WA, 1994.
11. D. Frost and R. Dechter. Look-ahead value ordering for constraint satisfaction problems. In *Proceedings IJCAI'95*, pages 572–578, Montréal, Canada, 1995.
12. J. Gaschnig. A general backtrack algorithm that eliminates most redundant tests. In *Proceedings IJCAI'77*, page 457, Cambridge MA, 1977.
13. J. Gaschnig. Performance measurement and analysis of certain search algorithms. Technical Report CMU-CS-79-124, Carnegie-Mellon University, Pittsburgh PA, 1979.
14. P.A. Geelen. Dual viewpoint heuristics for binary constraint satisfaction problems. In *Proceedings ECAI'92*, pages 31–35, Vienna, Austria, 1992.

15. M.L. Ginsberg, M. Frank, M.P. Halpin, and M.C. Torrance. Search lessons learned from crossword puzzles. In *Proceedings AAAI'90*, pages 210–215, Boston MA, 1990.
16. R.M. Haralick and G.L. Elliot. Increasing tree seach efficiency for constraint satisfaction problems. *Artificial Intelligence*, 14:263–313, 1980.
17. W.D. Harvey and M.L. Ginsberg. Limited discrepancy search. In *Proceedings IJCAI'95*, pages 607–613, Montréal, Canada, 1995.
18. P.D. Hubbe and E.C. Freuder. An efficient cross product representation of the constraint satisfaction problem search space. In *Proceedings AAAI'92*, pages 421–427, San José CA, 1992.
19. V. Kumar. Algorithms for constraint-satisfaction problems: A survey. *AI Magazine*, 13(1):32–44, Spring 1992.
20. B.A. Nadel. Tree search and arc consistency in constraint satisfaction algorithms. In L.Kanal and V.Kumar, editors, *Search in Artificial Intelligence*, pages 287–342. Springer-Verlag, 1988.
21. B.A. Nadel. Constraint satisfaction algorithms. *Computational Intelligence*, 5:188–224, 1989.
22. P. Prosser. Domain filtering can degrade intelligent backtracking search. In *Proceedings IJCAI'93*, pages 262–267, Chambéry, France, 1993.
23. P. Prosser. Hybrid algorithms for the constraint satisfaction problem. *Computational Intelligence*, 9(3):268–299, August 1993.
24. P. Prosser. Binary constraint satisfaction problems: some are harder than others. In *Proceedings ECAI'94*, pages 95–99, Amsterdam, The Netherlands, 1994.
25. P. Prosser. Mac-cbj: maintaining arc consistency with conflict-directed backjumping. Technical Report 95-177, Department of Computer Science, University of Starthclyde, 1995.
26. J.F. Puget. Ilog solver. In J. Gensel, editor, *Journées Contraintes et Objets*, Grenoble, France, November 1992. (in French).
27. P.W. Purdom. Search rearrangement backtracking and polynomial average time. *Artificial Intelligence*, 21:117–133, 1983.
28. D. Sabin and E. Freuder. Contradicting conventional wisdom in constraint satisfaction. In Alan Borning, editor, *PPCP'94: Second Workshop on Principles and Practice of Constraint Programming*, Seattle WA, May 1994.
29. B. Smith. The phase transition in constraint satisfaction problems: A closer look at the mushy region. In *Proceedings ECAI'94*, pages 100–104, Amsterdam, The Netherlands, 1994.
30. B. Smith and S.A. Grant. Sparse constraint graphs and exceptionally hard problems. In *Proceedings IJCAI'95*, pages 646–651, Montréal, Canada, 1995.
31. B. Smith and S.A. Grant. Where the exceptionally hard problems are. In *Proceedings of the workshop –Studying and Solving Really Hard Problems–, CP'95*, pages 172–182, Cassis, France, 1995.
32. Pascal Van Hentenryck. *Constraint Satisfaction in Logic Programming*. MIT Press, Cambridge, MA, 1989.
33. R.J. Wallace and E.C. Freuder. Conjunctive width heuristics for maximal constraint satisfaction. In *Proceedings AAAI'93*, pages 762–768, Washington D.C., 1993.