Multi-Level Dynamic Variable Ordering Heuristics for the Constraint Satisfaction Problem

Christian Bessière
LIRMM-CNRS
161, rue Ada
34392 MONTPELLIER Cedex 5
bessiere@lirmm.fr

Assef Chmeiss and **Lakhdar Saïs**
CRIL - Université d'Artois - IUT de Lens
Rue de l'université - SP 16
62307 LENS Cedex
{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 shows that the obtained heuristics improve significantly the best known until now.

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. Two famous algorithms maintaining different levels of consistency at each node are forward checking (FC), and maintaining arc consistency (MAC). Several papers have discussed their performances. It seems that they agree in considering MAC as a good choice for solving CSPs. Indeed, it shows good performances, and a better stability than FC [Sabin and Freuder, 1994; Bessière and Régin, 1996; Grant and Smith, 1996]. In the following, we will use MAC as our search algorithm.

Another improvement that has shown to be of major importance is the ordering of the variables (VO), 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 oder the variables were often quite simple, and concentrated on characteristics inherent to the variable to be ordered, and not too much on the influence its neighborhood could have. Those that used 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. In this paper, we want to use the influence of the neighborhood in the criterion of choice of a variable, while remaining free of any constraint check.

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 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.

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

2.2 Overview on the existing VOs

Ordering variables prior to or during search of solution in a CSP has been understood of prime importance for a long time. Many studies have concentrated on finding new heuristics that take more precisely into account the "constrainedness" of the variables, following the "fail first" principle [Haralick and Elliot, 1980]. 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 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 heursitics are `minwidth`, which chooses an ordering minimizing the width of the constraint network [Freuder, 1982], `maxdeg`, which orders the variables in decreasing size of their neighborhood [Dechter and Meiri, 1989], `bandwidth`, which minimizes the bandwidth of the constraint graph [Zabih, 1990], 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 Elliot in 1980 are called dynamic VOs (DVOs). The one introduced by Haralick and Elliot in [Haralick and Elliot, 1980] is `dom`, the heuristic choosing as next variable the one with the smallest remaining domain. We can point out that this heuristic forgets completely the structure of the network and the "fail first" principle to concentrate on minimizing the branching factor of the search tree.[2]

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 [Frost and Dechter, 1995]. Another heuristic, close to `dom+deg`, is the one derived from the Brélaz heuristic (proposed for graph coloring) [Brélaz, 1979]. We note it `dom+futdeg`. It breaks ties in `dom` by prefering the variable with highest future degree [Smith, 1999]. 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 ([Bessière and Régin, 1996]), 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 [Smith and Grant, 1998] under the name `DD`.

Finally, there is another kind of DVOs which it is worth mentionning: those taking into account the tightness of the constraints. The idea behind these heuristics is to use the consistency graph structure instead of the constraint graph structure. 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 [Nudel, 1983; Geelen, 1992; Gent *et al.*, 1996]. 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 on uniform random CSPs generated according to the model B [Prosser, 1996]. 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.

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 [Smith and Grant, 1998; Smith, 1999]. 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 at the value 55 since

[2] This is a bit peculiar that this heuristic has been introduced by the authors who defined the fail first principle.

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

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`. Those 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 $N > 150$.

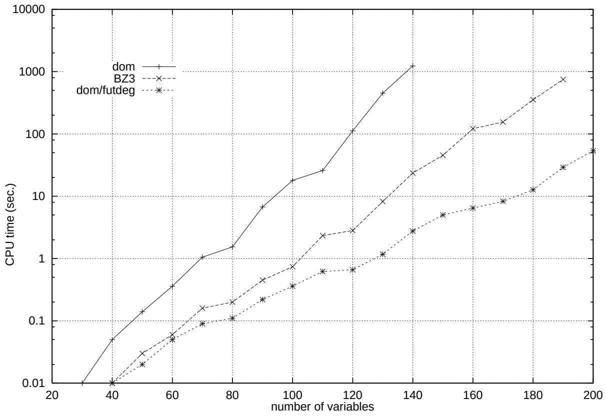

Figure 1: $< N, D = 10, C = 5 * N/2, T_{crit} >$: cpu time needed (log scale) for the search of the first solution with a MAC procedure.

3 Multi-levels DVOs

One of the key features for the efficiency in the backtrack search methods 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 [Hooker and Vinay, 1995] in the context of satisfiability problems (SAT): "a branching rule works better when it creates simpler subproblems". 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 *hardness* of such subproblems. Indeed, choosing the variable with minimum domain (resp. with maximum degree) reduces the number (resp. hardness) of such subproblems, whereas combining both, such as `dom/futdeg` does, tends to satisfy both objectives.

Let us mention, that most of the efficient satisfiability branching rules measure the score of a given variable as the sum of some given weights for the clauses in which it appears (e.g., [Jeroslow and Wang, 1990]).

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. As recalled in Section 2.2, this is not a new idea in CSPs, but what we propose below 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.

For the 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 $W(R_{ij}) = \alpha(X_i) + \alpha(X_j))$, where $\alpha(X_i)$ is instantiated to a simple syntactical property of the variable such as $|D(X_i)|$ or $\frac{|D(X_i)|}{|\Gamma(X_i)|}$.

We obtain the new formulation of (2):

(3) $H^{\alpha}(X_i) = \frac{1}{|\Gamma(X_i)|} \times (\alpha(X_i) + \frac{\sum_{X_j \in \Gamma(X_i)} \alpha(X_j)}{|\Gamma(X_i)|})$

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 hardness of the sub-problems, we propose a generalization of the DVO H^{α} such that variables at distance k from X_i are taken into account. This gives what we call a "multi-level DVO", H^{α}_k. To obtain this multi-level DVO, we simply replace $\alpha(X_j)$ in formula (3) by a recursive call to $H^{\alpha}_{k-1}(X_j)$. This means that to compute H^{α}_k on a given variable, we need to compute H^{α}_{k-1} on all its neighbors, and so on. The recursion terminates with H^{α}_0, equal to α. This is formally stated as follows:

(4) $H^{\alpha}_0(X_i) = \alpha(X_i)$

(5) $H^{\alpha}_k(X_i) = \frac{1}{|\Gamma(X_i)|} \times (\alpha(X_i) + \frac{\sum_{X_j \in \Gamma(X_i)} H^{\alpha}_{k-1}(X_j)}{|\Gamma(X_i)|})$

3.3 Some instantiations of the multi-level formula

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

- For $\alpha(X_i) = |D(X_i)|$ we obtain:

  (6) $H^{dom}_0(X_i) = |D(X_i)|$,

  (7) $H^{dom}_1(X_i) = \frac{|D(X_i)|}{|\Gamma(X_i)|} + \frac{\sum_{X_j \in \Gamma(X_i)} |D(X_j)|}{|\Gamma(X_i)|^2}$

- For $\alpha(X_i) = \frac{|D(X_i)|}{|\Gamma(X_i)|}$ we obtain :

  (8) $H^{dom/futdeg}_0(X_i) = \frac{|D(X_i)|}{|\Gamma(X_i)|}$,

  (9) $H^{dom/futdeg}_1(X_i) = \frac{|D(X_i)| + \sum_{X_j \in \Gamma(X_i)} \frac{|D(X_j)|}{|\Gamma(X_j)|}}{|\Gamma(X_i)|^2}$.

In the following, H_0^{dom} and $H_0^{dom/futdeg}$ are denoted by their classical name, `dom` and `dom/futdeg`, respectively. H_1^{dom} and $H_1^{dom/futdeg}$ are denoted by `H1_dom` and `H1_dom/futdeg` respectively.

4 Experiments

We will now compare experimentally the behavior of the new DVOs defined above on several classes of random CSPs. In the following, we limit our comparison to the first level DVOs (`H1_dom` and `H1_dom/futdeg`) and the most efficient DVO previously known, `dom/futdeg` (See Section 2.4).

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

1. the class of CSPs presented and described in Section 2.4 (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 $N = 100, D = 10$, and two different densities: $C = 250$ (Figure 3), and $C = 500$ (Figure 4).
3. CSPs with a greater number of variables and/or a greater domain size. The results are presented in Table 1. They are all near the crossover point.

In our tests we have used different kinds of measures of performances for the DVOs tested: cpu time in seconds,[4] number of constraint checks, and number of visited nodes. The different curves obtained using these different measures are very similar, so we just present those with cpu time. Each point represents the average performance for 50 generated problems.

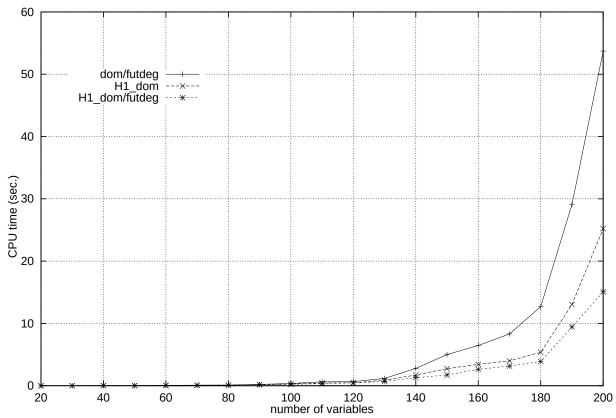

Figure 2: $< N, D = 10, C = 5 * N/2, T_{crit} >$: cpu time needed for the search of the first solution with a MAC procedure.

In Figure 2, N grows, while the domain size and the average degree remain constant. The tightness is always at the cross-over point. We can see that the two first-level DVOs, `H1_dom` and `H1_dom/futdeg`, outperform the best known DVO, `dom/futdeg`. `H1_dom/futdeg` has the best performances on this class of CSPs. It is interesting to note that the differences increase as the number of variables increase.

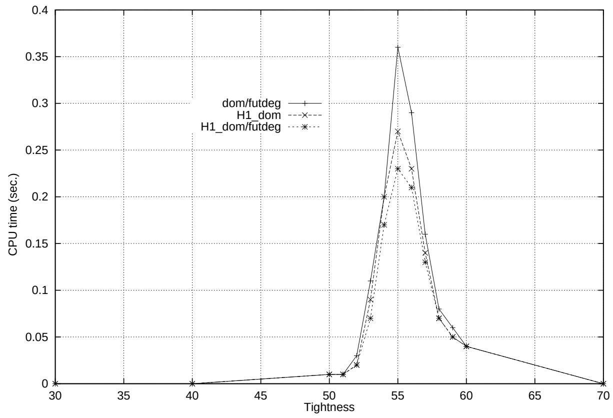

Figure 3: $< N = 100, D = 10, C = 250, T >$: cpu time needed for the search of the first solution with a MAC procedure.

Figures 3 and 4 confirm the improvements obtained using the two first levels DVOs. `H1_dom` and `H1_dom/futdeg` remain better than `dom/futdeg`. The obtained gain increases with the density of the graph. The behavior of `H1_dom` and `H1_dom/futdeg` become very similar when $C = 500$ (i.e., average degree of 10).

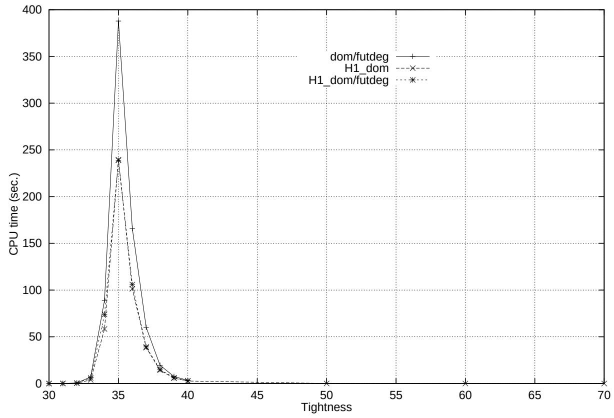

Figure 4: $< N = 100, D = 10, C = 500, T >$: cpu time needed for the search of the first solution with a MAC procedure.

In its three first lines, Table 1 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 two lines show that if we increase the number of variables even more ($< \mathbf{300}, 6, 1200, 12 >$), or if we increase the domain size ($< 150, \mathbf{15}, 375, 139 >$), the gain of the H_1 heuristics continue to grow compared to `dom/futdeg`. These are promising observations for the

[4]All experimentations have been run on a PC Pentium III 667 MHz under Linux.

	`dom/futdeg`		`H1_dom`		`H1_dom/futdeg`	
Problems $< N, D, C, T >$	#ccks (million)	time (sec.)	#ccks (million)	time (sec.)	#ccks (million)	time (sec.)
$< 100, 10, 250, 55 >$	2.64	0.36	1.86	0.27	1.45	0.23
$< 100, 10, 500, 35 >$	3713.87	387.90	2201.49	239.46	2148.35	238.94
$< 200, 10, 500, 55 >$	480.25	53.71	228.73	25.21	136.68	15.08
$< 300, 6, 1200, 12 >$	9158.42	2066.95	3989.01	940.57	4338.20	1060.94
$< 150, 15, 375, 139 >$	2267.35	208	929.43	88	516.72	50

Table 1: A sample of random CSPs at the cross-over point (results for the search of the first solution with a MAC procedure).

multi-level approach.

It also seems that when the average degree of the network is greater, `H1_dom` becomes as good as (or better than) `H1_dom/futdeg`. (See Figure 3 compared to Figure 4, and the line $< 300, 6, 1200, 12 >$ of Table 1, where the average degree is 8.)

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 "$\neq$".
- We have considered only the DVOs obtained by instantiating the first level of the general formulation. What about greater levels?
- Finally, we plan to investigate the effect of using other operators (e.g., the product) to compute the weight of the constraints.

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 advantage,

- the constrainedness of a given variable is computed without any constraint check, thanks to simple syntactical properties,
- it takes advantage from the neighborhood of the variable, with the notion of distance as a parameter,
- it can be instantiated with 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 with respect to random CSPs.

References

[Bessière and Régin, 1996] 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.

[Brélaz, 1979] D. Brélaz. New methods to color the vertices of a graph. *Communications of the ACM*, 22:251–256, 1979.

[Dechter and Meiri, 1989] R. Dechter and I. Meiri. Experimental evaluation of preprocessing techniques in constraint satisfaction problems. In *Proceedings IJCAI'89*, pages 271–277, Detroit MI, 1989.

[Freuder, 1982] E.C. Freuder. A sufficient condition for backtrack-free search. *Journal of the ACM*, 29(1):24–32, Jan. 1982.

[Frost and Dechter, 1995] D. Frost and R. Dechter. Look-ahead value ordering for constraint satisfaction problems. In *Proceedings IJCAI'95*, pages 572–578, Montréal, Canada, 1995.

[Geelen, 1992] P.A. Geelen. Dual viewpoint heuristics for binary constraint satisfaction problems. In *Proceedings ECAI'92*, pages 31–35, Vienna, Austria, 1992.

[Gent *et al.*, 1996] 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.

[Grant and Smith, 1996] S.A. Grant and B.M. Smith. The phase transition behavior of maintaining arc consistency. In *Proceedings ECAI'96*, pages 175–179, Budapest, Hungary, 1996.

[Haralick and Elliot, 1980] R.M. Haralick and G.L. Elliot. Increasing tree seach efficiency for constraint satisfaction problems. *Artificial Intelligence*, 14:263–313, 1980.

[Nudel, 1983] B. Nudel. Consistent-labelling problems and their algorithms: Expected-complexities and theory-based heuristics. *Artificial Intelligence*, 21:135–178, 1983.

[Prosser, 1996] P. Prosser. An empirical study of phase transition in binary constraint satisfaction problems. *Artificial Intelligence*, 81:81–109, 1996.

[Sabin and Freuder, 1994] D. Sabin and E.C. Freuder. Contradicting conventional wisdom in constraint satisfaction. In *Proceedings PPCP'94*, Seattle WA, 1994.

[Smith and Grant, 1998] B. Smith and S.A. Grant. Trying harder to fail first. In *Proceedings ECAI'98*, pages 249–253, Brighton, UK, 1998.

[Smith, 1999] B.M. Smith. The Bélaz heuristic and optimal static orderings. In *Proceedings CP'99*, pages 405–418, Alexandria VA, 1999.

[Zabih, 1990] R. Zabih. Some applications of graph bandwith to constraint satisfaction problems. In *Proceedings AAAI'90*, pages 46–51, Boston MA, 1990.

[Jeroslow and Wang, 1990] R. G. Jeroslow and J. Wang. Solving Propositional Satisfiability Problems. In *Annals of Mathematics and Artificial Intelligence*, 1:167–187, 1990.

[Hooker and Vinay, 1995] J. N. Hooker and V. Vinay. Branching rules for satisfiability. In *Journal of Automated Reasoning*, 15:359–383, 1995.