Conflict Directed Variable Selection Strategies for Constraint Satisfaction Problems

Thanasis Balafoutis and Kostas Stergiou

Department of Information and Communication Systems Engineering
University of the Aegean, Samos, Greece
email: {abalafoutis,konsterg}@aegean.gr

Abstract. It is well known that the order in which variables are instantiated by a backtracking search algorithm can make an enormous difference to the search effort in solving CSPs. Among the plethora of heuristics that have been proposed in the literature to efficiently order variables during search, a significant recently proposed class uses the learning-from-failure approach. Prime examples of such heuristics are the wdeg and dom/wdeg heuristics of Boussemart et al. which store and exploit information about failures in the form of constraint weights. The efficiency of all the proposed conflict-directed heuristics is due to their ability to learn though conflicts encountered during search. As a result, they can guide search towards hard parts of the problem and identify contentious constraints. Such heuristics are now considered as the most efficient general purpose variable ordering heuristic for CSPs. In this paper we show how information about constraint weights can be used in order to create several new variants of the *wdeg* and *dom/wdeg* heuristics. The proposed conflict-driven variable ordering heuristics have been tested over a wide range of benchmarks. Experimental results show that they are quite competitive compared to existing ones and in some cases they can increase efficiency.

1 Introduction

Constraint satisfaction problems (CSPs) and propositional satisfiability (SAT) are two automated reasoning technologies that have a lot in common regarding the approaches and algorithms they use for solving combinatorial problems. Most complete algorithms from both paradigms use constraint propagation methods together with variable ordering heuristics to improve search efficiency. Learning from failure has become a key component in solving combinatorial problems in the SAT community, through literals learning and weighting, e.g. as implemented in the Chaff solver [7]. This approach is based on learning new literals through conflict analysis and assigning weights to literals based on the number of times they cause a failure during search. This information can be then exploited by the variable ordering heuristic to efficiently choose the variable to assign at each choice point.

In the CSP community, learning from failure has followed a similar direction in recent years, in particular with respect to novel variable ordering heuristics.

Boussemart et al. were the first to introduce SAT influenced heuristics that learn from conflicts encountered during search [3]. In their approach, constraint weights are used as a metric to guide the variable ordering heuristic towards hard parts of the problem. Constraint weights are continuously updated during search using information learned from failures. The advantage that these heuristics have is that they use previous search states as guidance, while most formerly proposed heuristics either use the initial or the current state. The heuristics of [3], called *wdeg* and *dom/wdeg*, are now probably considered as the most efficient general purpose variable ordering heuristic for CSPs. Sunsequently, a number of alternative heuristics based on learning during search were proposed [8, 4, 6].

As discussed by Grimes and Wallace, heuristics based on constraint weights can be conceived in terms of an overall strategy that except from the standard Fail-First Principle also obeys the Contention Principle, which states that variables directly related to conflicts are more likely to cause a failure if they are chosen instead of other variables [6].

In this paper we focus on conflict-driven variable ordering heuristics based on constraint weights. We concentrate on an investigation of new general purpose variants of conflict-driven heuristics. These variants differ from *wdeg* and *dom/wdeg* in the way they assign weights to constraints. First we propose three new variants of the *wdeg* and *dom/wdeg* heuristics that record the constraint that is responsible for any value deletion during search. These heuristics then exploit this information to update constraint weights upon detection of failure. We also examine a SAT influenced weight aging strategy that gives greater importance to recent conflicts. Finally, we propose a new heuristic that tries to better identify contentious constraints by detecting all the possible conflicts after a failure. Experimental results from various random, academic and real world problems show that some of the proposed heuristics are quite competitive compared to existing ones and in some cases they can increase efficiency.

The rest of the paper is organized as follows. Section 2 gives the necessary background material and an overview on the existing conflict-driven variable ordering heuristics. In Section 3 we propose several new general purpose variants of conflict-driven variable ordering heuristics. In Section 4 we experimentally compare the proposed heuristics to *dom/wdeg* on a variety of real, academic and random problems. Finally, conclusions are presented in Section 5.

2 Background

A *Constraint Satisfaction Problem* (CSP) is a tuple (*X, D, C*), where X is a set containing n variables $\{x_1, x_2, ..., x_n\}$; D is a set of domains $\{D(x_1), D(x_2),..., D(x_n)\}$ for those variables, with each $D(x_i)$ consisting of the possible values which x_i may take; and C is a set of constraints $\{c_1, c_2, ..., c_k\}$ between variables in subsets of X. Each $c_i \in C$ expresses a relation defining which variable assignment combinations are allowed for the variables *vars(c_i)* in the scope of the constraint. Two variables are said to be *neighbors* if they share a constraint. The *arity* of a constraint is the number of variables in the scope of the constraint.

The *degree* of a variable x_i, denoted by $\Gamma(x_i)$, is the number of constraints in which x_i participates. A binary constraint between variables x_i and x_j will be denoted by c_{ij}. In this paper we focus on binary CSPs. However, the proposed variable ordering heuristics are generic and can be applied on problems with constraints of any arity.

A partial assignment is a set of tuple pairs, each tuple consisting of an instantiated variable and the value that is assigned to it in the current search state. A full assignment is one containing all n variables. A solution to a CSP is a full assignment such that no constraint is violated.

An *arc* is a pair (c, x_i) where $x_i \in vars(c)$. Any arc (c_{ij}, x_i) will be alternatively denoted by the pair of variables (x_i,x_j), where $x_j \in vars(c_{ij})$. That is, x_j is the other variable involved in c_{ij}. An arc (x_i,x_j) is *arc consistent* (AC) iff for every value $a \in D(x_i)$ there exists at least one value $b \in D(x_j)$ such that the pair (a,b) satisfies c_{ij}. In this case we say that b is a *support* of a on arc (x_i,x_j). Accordingly, a is a support of b on arc (x_j,x_i). A problem is AC iff there are no empty domains and all arcs are AC. The application of AC on a problem results in the removal of all non-supported values from the domains of the variables. The definition of arc consistency for non-binary constraints, usually called *generalized arc consistency* (GAC), is a direct extension of the definition of AC. A *support check* (consistency check) is a test to find out if two values support each other. The *revision* of an arc (x_i,x_j) using AC verifies if all values in $D(x_i)$ have supports in $D(x_j)$. A domain wipeout (DWO) revision is one that causes a DWO. That is, it results in an empty domain.

In the following will use MAC (maintaining arc consistency) [9, 1] as our search algorithm. In MAC a problem is made arc consistent after every assignment, i.e. all values which are arc inconsistent given that assignment, are removed from the current domains of their variables. If during this process a DWO occurs, then the last value selected is removed from the current domain of its variable and a new value is assigned to the variable. If no new value exists then the algorithm backtracks.

2.1 Overview of existing conflict-driven variable ordering heuristics

The order in which variables are assigned by a backtracking search algorithm has been understood for a long time to be of primary importance. A variable ordering can be either static, where the ordering is fixed and determined prior to search, or dynamic, where the ordering is determined as the search progresses. Dynamic variable orderings are considerably more efficient and have thus received much attention in the literature. One common dynamic variable ordering strategy, known as "fail-first", is to select as the next variable the one likely to fail as quickly as possible. All other factors being equal, the variable with the smallest number of viable values in its (current) domain will have the fewest subtrees rooted at those values, and therefore, if none of these contain a solution, the search can quickly return to a path that leads to a solution.

Recent years have seen the emergence of numerous modern heuristics for choosing variables during CSP search. The so called conflict-driven heuristics

exploit information about failures gathered throughout search and recorded in the form of constraint weights.

Boussemart et al. [3] proposed the first conflict-directed variable ordering heuristics. In these heuristics, every time a constraint causes a failure (i.e. a domain wipeout) during search, its weight is incremented by one. Each variable has a *weighted degree*, which is the sum of the weights over all constraints in which this variable participates. The weighted degree heuristic (*wdeg*) selects the variable with the largest weighted degree. The current domain of the variable can also be incorporated to give the domain-over-weighted-degree heuristic (*dom/wdeg*) which selects the variable with minimum ratio between current domain size and weighted degree. Both of these heuristics (especially *dom/wdeg*) have been shown to be extremely effective on a wide range of problems.

Grimes and Wallace [6] proposed alternative conflict-driven heuristics that consider value deletions as the basic propagation events associated with constraint weights. That is, the weight of a constraint is incremented each time the constraint causes one or more value deletions. They also used a sampling technique called *random probing* with which they can uncover cases of *global contention*, i.e. contention that holds across the entire search space.

The heuristics of [6] work as follows:

- constraint weights are increased by the size of the domain reduction leading to a DWO.
- whenever a domain is reduced in size during constraint propagation, the weight of the constraint involved is incremented by 1.
- whenever a domain is reduced in size, the constraint weights are increased by the size of domain reduction (*allDel* heuristic).

3 Heuristics based on weighting constraints

As stated in the previous section, the *wdeg* and *dom/wdeg* heuristics associate a counter, called *weight*, with each constraint of a problem. These counters are updated during search whenever a DWO occurs. Although experimentally it has been shown that these heuristics are extremely effective on a wide range of problems, in theory it seems quite plausible that they may not always assign weights to constraints in an accurate way. To better illustrate our conjecture about the accuracy in assigning weights to constraints, we give the following example.

Example 1. Assume we are using MAC-3 (i.e. MAC with AC-3) to solve a CSP (X, D, C) where X includes, among others, the three variables $\{x_i, x_j, x_k\}$, all having the same domain $\{a, b, c, d, e\}$, and C includes, among others, the two binary constraints c_{ij}, c_{ik}. Also assume that a conflict-driven variable ordering heuristic (e.g. *dom/wdeg*) is used, and that at some point during search AC tries to revise variable x_i. That is, it tries to find supports for the values in $D(x_i)$ in the constraints where x_i participates. Suppose that when x_i is revised against c_{ij}, values $\{a, b, c, d\}$ are removed from $D(x_i)$ (i.e. they do not have a

support in $D(x_j)$). Also suppose that when x_i is revised against c_{ik}, value $\{e\}$ is removed from $D(x_i)$ and hence a DWO occurs. Then, the dom/wdeg heuristic will increase the weight of constraint c_{ik} by one but it will not change the weight of c_{ij}.

It is obvious from this example that although constraint c_{ij} removes more values from $D(x_i)$ than c_{ik}, its important indirect contribution to the DWO is ignored by the heuristic.

A second point regarding potential inefficiencies of *wdeg* and *dom/wdeg* has to do with the order in which revisions are made by the AC algorithm used. Coarse-grained AC algorithms, like AC-3, use a *revision list* to propagate the effects of variable assignments. It has been shown that the order in which the elements of the list are selected for revision affects the overall cost of search. Hence a number of revision ordering heuristics have been proposed [10, 2]. In general, revision ordering and variable ordering heuristics have different tasks to perform when used in a search algorithm like MAC. Before the appearance of conflict-driven heuristics there was no way to achieve an interaction with each other, i.e. the order in which the revision list was organized during the application of AC could not affect the decision of which variable to select next (and vice versa). The contribution of revision ordering heuristics to the solver's efficiency was limited to the reduction of list operations and constraint checks.

However, when a conflict-driven variable ordering heuristic like *dom/weg* is used, then there are cases where the decision of which arc (or variable) to revise first can affect the variable selection. To better illustrate this interaction we give the following example.

Example 2. Assume that we want to solve a CSP (X, D, C) using a conflict-driven variable ordering heuristic (e.g. dom/wdeg), and that at some point during search the following AC revision list is formed: Q=$\{(x_1), (x_3), (x_5)\}$. Suppose that revising x_1 against constraint c_{12} leads to the DWO of $D(x_1)$, i.e. the remaining values of x_1 have no support in $D(x_2)$. Suppose also that the revision of x_5 against constraint c_{56} leads to the DWO of $D(x_5)$, i.e. the remaining values of x_5 have no support in $D(x_6)$. Depending on the order in which revisions are performed, one or the other between the two possible DWOs will be detected. If a revision ordering heuristic R_1 selects x_1 first then the DWO of $D(x_1)$ will be detected and the weight of constraint c_{12} will increased by 1. If some other revision ordering heuristic R_2 selects x_5 first then the DWO of $D(x_5)$ will be detected, but this time the weight of a different constraint (c_{56}) will increased by 1. Although the revision list includes two variables (x_1, x_5) that can cause a DWO, and consequently two constraint weights can be increased (c_{12}, c_{56}), dom/wdeg will increase the weight of only one constraint depending on the choice of the revision heuristic. Since constraint weights affect the choices of the variable ordering heuristic, R_1 and R_2 can lead to different future decisions for variable instantiation. Thus, R_1 and R_2 may guide search to different parts of the search space.

From the above example it becomes clear that known heuristics based on constraint weights are quite sensitive to revision orderings and their performance can be affected by them.

In order to overcome the above described weaknesses that the weighted degree heuristics seem to have, we next describe a number of new variable ordering heuristics which can be seen as variants of *wdeg* and *dom/weg*.

3.1 Constraints responsible for value deletions

The first enhancement to *wdeg* and *dom/wdeg* tries to alleviate the problem illustrated in Example 1. To achieve this, we propose to record the constraint which is responsible for each value deletion from any variable in the problem. In this way, once a DWO occurs during search we know which constraints have, not only directly, but also indirectly contributed to the DWO. Based on this idea, when a DWO occurs in a variable x_i, constraint weights can be updated in the following three alternative ways:

- *Heuristic* ***H1***: for every constraint that is responsible for any value deletion from $D(x_i)$, we increase its weight by one.
- *Heuristic* ***H2***: for every constraint that is responsible for any value deletion from variable $D(x_i)$, we increase its weight by the number of value deletions.
- *Heuristic* ***H3***: for every constraint that is responsible for any value deletion from variable $D(x_i)$, we increase its weight by the normalized number of value deletions. That is, by the ratio between the number of value deletions and the size of $D(x_i)$.

The way in which the new heuristics update constraint weights is displayed in the following example.

Example 3. Assume that when solving a CSP (X, D, C), the domain of some variable e.g. x_1 is wiped out. Suppose that $D(x_1)$ initially was $\{a, b, c, d, e\}$ and each of the values was deleted because of constraints: $\{c_{12}, c_{12}, c_{13}, c_{12}, c_{13}\}$ respectively. The proposed heuristics will assign constraint weights as follows: H1($weight_{H1}[c_{12}] = weight_{H1}[c_{13}] = 1$), H2($weight_{H2}[c_{12}] = 3, weight_{H2}[c_{13}] = 2$) and H3($weight_{H3}[c_{12}] = 3/5, weight_{H3}[c_{13}] = 2/5$)

Heuristics $H1$, $H2$, $H3$ are closely related to the three heuristics proposed by Grimes and Wallace [6].

The last two heuristics in [6], record constraints responsible for value deletions and use this information to increase weights. However, the weights are increased during constraint propagation in each value deletion for all variables. Our proposed heuristics differ by increasing constraints weights only when a DWO occurs. As discussed in [6], DWOs seem to be particularly important events in helping identify hard parts of the problem. Hence we focus on information derived from DWOs and not just any value deletion.

3.2 Constraint weight aging

Most of the state-of-the-art SAT solvers like BerkMin [5] and Chaff [7], use the strategy of weight "aging". In such solvers, each variable is assigned a counter that stores the number of clauses responsible for at least one conflict . The value of this counter is updated during search. As soon as a new clause responsible for the current conflict is derived, the counters of the variables, whose literals are in this clause, are incremented by one. The values of all counters are periodically divided by a small constant greater than 1. This constant is equal to 2 for Chaff and 4 for BerkMin. In this way, the influence of "aged" clauses is decreased and preference is given to recently deduced clauses.

Inspired from SAT solvers, we propose here the use of "aging" to periodically age constraint weights. As in SAT, constraint weights can be "aged" by periodically dividing their current value by a constant greater than 1. The period of divisions can be set according to a specified number of backtracks during search. With such a strategy we give greater importance to recently discovered conflicts. The following example illustrates the improvement that weight "aging" can contribute to the solver's performance.

Example 4. Assume that in a CSP (X, D, C) with D={0,1,2}, we have a ternary constraint $c_{123} \in C$ for variables x_1, x_2, x_3 with disallowed tuples {(0,0,0), (0,0,1), (0,1,1), (0,2,2)}. When variable x_1 is set to a value different from 0 during search, constraint c_{123} is not involved in a conflict and hence its weight will not increase. However, in a branch that includes assignment $x_1 = 0$, constraint c_{123} becomes highly "active" and a possible DWO in variable x_2 or x_3 should increase the importance of constraint c_{123} (more that a simple increment of its weight by one). We need a mechanism to quickly adopt changes in the problem caused by a value assignment. This can be done, by "aging" the weights of the other previously active constraints.

3.3 Fully assigned weights

When arc consistency is maintained during search using a coarse grained algorithm like AC-3, a revision list is created after each variable assignment. The variables that have been inserted into the list are removed and revised in turn. We observed that in the same revision list, different revision ordering heuristics can lead to the DWOs of different variables. To better illustrate this, we give the following example.

Example 5. Assume that we use two different revision ordering heuristic R_1, R_2 to solve a CSP (X, D, C), and that at some point during search the following AC revision list is formed for R_1 and R_2. R_1:{X_1,X_2}, R_2:{X_2,X_1}. We also assume the following: *a*) The revision of X_1 deletes some values from the domain of X_1 and it causes the addition of the variable X_3 in the revision list. *b*) The revision of X_2 deletes some values from the domain of X_2 and it causes the addition of the variable X_4 in the revision list. *c*) The revision of X_3 deletes some values from the domain of X_1. *d*) The revision of X_4 deletes some values from the

domain of X_2. e). A DWO occurs after a sequential revision of X_3 and X_1. f) A DWO occurs after a sequential revision of X_4 and X_2. Considering the R_1 list, the revision of X_1 is fruitful and adds X_3 in the list (R_1:{X_3,X_1}). The sequential revision of X_3 and X_1 leads to the DWO of X_1. Considering the R_2 list, the revision of X_2 is fruitful and adds X_4 in the list (R_2:{X_4,X_2}). The sequential revision of X_4 and X_2 leads to the DWO of X_2.

From the above example it is clear that although only one DWO is identified in a revision list, both X_1 and X_2 can be responsible for this. In R_1 where X_1 is the DWO variable, we can say that X_2 is also a "potential" DWO variable i.e. it would be a DWO variable, if the R_2 revision ordering was used. The question that arises here is: how can we identify the "potential" DWO variables that exists on a revision list? A first observation that can be helpful in answering this question is that "potential" DWO variables are among variables that participate in fruitful revisions.

Based on this observation, we propose here a new conflict-driven variable ordering heuristic that takes into account the "potential" DWO variables. This heuristic increases the weights of constraints that are responsible for a DWO by one (as *wdeg* heuristic does) and also, only for revision lists that lead to a DWO, increases by one the weights of constraints that participates in fruitful revisions. Hence, to implement this heuristic we record all variables that delete at least one value during the application of AC. If a DWO is detected, we increase the weight of all these variables.

An interesting direction for future work can be a more selective identification of "potential" DWO variables.

4 Experiments and results

In this section we experimentally investigate the behavior of the new proposed variable ordering heuristics on several classes of real, academic and random problems. All benchmarks are taken from C. Lecoutre's web page[1], where the reader can find addition details about the description and the formulation of all the tested benchmarks. We compare the new proposed heuristics with *dom/wdeg* and *allDel* (whenever a domain is reduced in size, the constraint weights are increased by the size of domain reduction). Regarding the heuristics of Section 3.1, we only show results from $dom/wdeg_{H1}$, $dom/wdeg_{H2}$ and $dom/wdeg_{H3}$, denoted as $H1$, $H2$ and $H3$ for simplicity, which are more efficient than the corresponding versions that do not take the domain size into account. In our tests we have used the following measures of performance: cpu time in seconds (t) and number of visited nodes (n). The solver we used applies d-way branching and lexicographic value ordering. It also employs restarts. Concerning the restart policy, the initial number of allowed backtracks for the first run has been set to 10 and at each new run the number of allowed backtracks increases by a factor of 1.5. Regarding the aging heuristic, we have select to periodically decrease all

[1] http://www.cril.univ-artois.fr/∼lecoutre/research/benchmarks/

constraint weights by a factor of 2, with the period set to 20 backtracks. Our search algorithm is MGAC-3, denoting MAC with GAC-3.

Table 1. Averaged values for Cpu times (t), and nodes (n) from 6 different problem classes. Best cpu time is in bold.

Problem Class		*dom/wdeg*	*H1*	*H2*	*H3*	*aged dom/wdeg*	*fully* assigned	allDel
RLFAP scensMod	t	1,9	2	2,2	2,3	**1,7**	2,2	2,2
(13 instances)	n	734	768	824	873	646	738	809
RLFAP graphMod	t	9,1	**5,2**	6,1	5,5	12,9	13,4	11,1
(12 instances)	n	6168	3448	4111	3295	8478	11108	9346
Driver	t	22,4	7	7,8	11,6	**6,4**	18,8	20
(11 instances)	n	10866	2986	3604	5829	1654	4746	4568
Interval Series	t	34	19,4	23,4	13,3	**6,5**	66,4	17,4
(10 instances)	n	32091	18751	23644	13334	5860	74310	26127
Golomb Ruler	t	274,9	321,4	173,1	**143,4**	342,1	208,3	154,4
(6 instances)	n	7728	10337	4480	3782	7863	6815	3841
geo50-20-d4-75	t	62,8	174,1	72,1	95	69	**57,6**	76
(10 instances)	n	15087	36949	16970	23562	15031	12508	18094
frb30-15	t	37,3	35,1	45,8	57,2	42,3	32,9	**26,1**
(10 instances)	n	20176	18672	24326	30027	21759	17717	14608

Table 1 show results from six different problem classes. The first two classes are from the real world Radio Link Frequency Assignment Problem (RLFAP). For the *scensMod* class we have run 13 instances and in this table we present the averaged values for cpu time and nodes visited. Since these instances are quite easy to be solved all the heuristics have almost the same behavior. The *aged* version of the *dom/wdeg* heuristic has a slightly better performance. For the *graphMod* class we have run 12 instances. Here the heuristics *H1, H2, H3* that record the constraint which is responsible for each value deletion a better performance. The third problem class is from another real world problem, which is called *Driver*. In these 11 instances the *aged dom/wdeg* heuristic has on average the best behavior. The next 10 instances are from the non-binary academic problem "All Interval Series" (See prob007 at http://www.csplib.org) which have maximum arity of 3. We must notice here that the *aged dom/wdeg* heuristic, which has the best performance is five times faster compared with the *dom/wdeg*.

This good performance that the *aged dom/wdeg* heuristic has, is not generic within different problem classes. This can be seen in the next academic problem class (the well known Golomb Ruler problem) where the *aged dom/wdeg* heuristic, have the worst performance. The last two classes are from the "*geo*"quasi-random instances (random problems which contain some structure) and from the "*frb*" pure random instances that are forced to be satisfied. Here, although on average the *fullyAssigned* and *allDel* heuristics have the best performance, within each class we observed a big variation in cpu time among all the tested heuristics. A possible explanation for this diversity is the lack of structure that random instances have.

Finally we must also comment that interestingly the *dom/wdeg* heuristic does not achieve any win, in all the tested experiments. As a general comment

we can say that experimentally, all the proposed heuristics are competitive with *dom/wdeg* and in many benchmarks a notable improvement is observed.

5 Conclusions

In this paper several new general purpose variable ordering heuristics are proposed. These heuristics follow the learning-from-failure approach, in which information regarding failures is stored in the form of constraint weights. By recording constraints that are responsible for any value deletion, we derive three new heuristics that use this information to spread constraint weights in a different way compared to the heuristics of Boussemart et al. We also explore a SAT inspired constraint aging strategy that gives greater importance to recent conflicts. Finally we proposed a new heuristic that tries to better identify contentious constraints by recording all the potential conflicts uppon detection of failure. The proposed conflict driven variable ordering heuristics have been tested over a wide range of benchmarks. Experimental results shows they are quite competitive to the existing ones and is some cases they can increase efficiency.

References

1. C. Bessière and J.C. Régin. MAC and combined heuristics: two reasons to forsake FC (and CBJ?). In *Proceedings of the 2nd Conference on Principles and Practice of Constraint Programming (CP-1996)*, pages 61–75, Cambridge MA, 1996.
2. F. Boussemart, F. Hemery, and C. Lecoutre. Revision ordering heuristics for the Constraint Satisfaction Problem. In *10th International Conference on Principles and Practice of Constraint Programming (CP-2004), Workshop on Constraint Propagation and Implementation*, Toronto, Canada, 2004.
3. F. Boussemart, F. Hemery, C. Lecoutre, and L. Sais. Boosting systematic search by weighting constraints. In *Proceedings of 16th European Conference on Artificial Intelligence (ECAI-2004)*, pages 146–150, Valencia, Spain, 2004.
4. H. Cambazard and N. Jussien. Identifying and Exploiting Problem Structures Using Explanation-based Constraint Programming. *Constraints*, 11:295–313, 2006.
5. E. Goldberg and Y. Novikov. BerkMin: a Fast and Robust Sat-Solver. In *Proceedings of DATE'02*, pages 142–149, 2002.
6. D. Grimes and R.J. Wallace. Sampling strategies and variable selection in weighted degree heuristics. In *Proceedings of the 13th Conference on Principles and Practice of Constraint Programming (CP-2007)*, pages 831–838, 2007.
7. M. Moskewicz, C. Madigan, and S. Malik. Chaff: Engineering an efficient sat solver. In *Proceedings of Design Automation Conference*, pages 530–535, 2001.
8. P. Refalo. Impact-based search strategies for constraint programming. In *Proceedings of the 10th Conference on Principles and Practice of Constraint Programming (CP-2004)*, pages 556–571, 2004.
9. D. Sabin and E.C. Freuder. Contradicting conventional wisdom in constraint satisfaction. In *Proceedings 2nd Workshop on Principles and Practice of Constraint Programming (CP-1994)*, pages 10–20, 1994.
10. R. Wallace and E. Freuder. Ordering heuristics for arc consistency algorithms. In *AI/GI/VI*, pages 163–169, Vancouver, British Columbia, Canada, 1992.