Learning to Identify Global Bottlenecks in Constraint Satisfaction Search *

Diarmuid Grimes and Richard J. Wallace
Cork Constraint Computation Centre and Department of Computer Science
University College Cork, Cork, Ireland
{d.grimes, r.wallace}@4c.ucc.ie

Abstract

Using information from failures to guide subsequent search is an important technique for solving combinatorial problems in domains such as boolean satisfiability (SAT) and constraint satisfaction problems (CSPs). The information learnt can take various forms such as fine-grained information in the form of no-goods and explanations in CSPs and clause learning in SAT, or coarse-grained information in the form of constraint weighting in CSPs and clause weighting in SAT.

In this paper we focus on CSPs, using constraint weighting with restarts in order to identify global bottlenecks in a problem. This information is then used by a "weighted-degree" heuristic to guide complete search, with the belief that instantiating these elements first will reduce the overall search effort required to either find a solution or prove the problem insoluble.

We introduce two restarting strategies. ln WTDI (WeighTeD Information gathering) the weighted-degree heuristic itself is used with restarts; in RNDI (RaNDom Information gathering) random variable selection is combined with constraint weighting and restarts. For problems with clearly defined sources of global contention both approaches were superior to complete search with the original weighted degree heuristic. However when these "globally" difficult elements became more subtle, RNDI outperformed WTDI.

Introduction

Adaptive algorithms that use information gathered during search represent a relatively new approach to solving hard combinatorial problems such as the constraint satisfaction problem (CSP), in which a set of variables must be assigned values that satisfy constraints among subsets of variables in the problem.

These adaptive methods usually gather information about failure, which occurs when a partial solution cannot be extended. The most straightforward method is to store this partial solution as a *no-good*; then if search encounters this set of assignments again it can backtrack without further testing. A subtler method is to record evidence about the context of failure, usually in the form of *constraint weights*; this information is used to refine subsequent heuristic decisions. Examples of the latter strategy are the weighted degree heuristic proposed by (Boussemart *et al.* 2004) and the breakout preprocessing strategy of (Eisenberg & Faltings 2003).

Constraint- or edge-weight learning follows the logic of the Fail-First Principle of (Haralick & Elliott 1980). This principle states that, "In order to succeed, try first where you are most likely to fail." That is, in solving CSPs it is generally best to deal with variables that cause difficulty as early as possible. This is because there is no point in making assignments to easy parts of the problem and then undoing them repeatedly when it proves impossible to find consistent assignments for the remaining, difficult variables. Although we know that variable ordering heuristics are affected by factors other than fail-firstness (Smith & Grant 1998) (Beck, Prosser, & Wallace 2003), it is clear that this has a major impact on search efficiency (Beck, Prosser, & Wallace 2005). Under this interpretation, weighting strategies work by flagging variables that may be part of difficult subproblems, as indicated by failures in these or neighboring variables.

The present work is based on the weighted degree heuristic (Boussemart *et al.* 2004). This heuristic uses a strategy of associating weights in the form of integers with each constraint. Weights are initially set to 1, and each time a constraint removes the final value(s) from a domain (causing a domain wipeout), its weight is incremented. Variables are then selected on the basis of the total weight associated with their constraints. This is, therefore, an adaptive heuristic that works to enhance any heuristic that takes into account a variable's degree. This strategy has been shown to outperform one of the most widely used heuristics (min-domain/degree) on a variety of difficult problems (Boussemart *et al.* 2004), (Hulubei & O'Sullivan 2006).

However, using weighted degree in this way has potential limitations. The most obvious is that the heuristic uses the least amount of data when making its most important choices, i.e. the first few variable selections. Improvements in search are likely to be restricted to lower levels of the search tree; in particular, search may never return to those first few variables.

In this paper, we show that this heuristic can be improved by gathering information from different parts of the search space prior to solving. Restarting search is a widely used

*This work was supported by Science Foundation Ireland under Grants 00/PI.1/C075 and 05/IN/I886.

technique for problem solving that has proven to be effective in a number of domains. In SAT, the solver Chaff (Moskewicz, Madigan, & Malik 2001) uses restarts with clause weighting and clause learning to provide a chance to change early decisions in view of the current problem state. Here we use two approaches with CSPs that combine restarting with edge-weight learning in order to direct search more intelligently. The first approach (WTDI) is similar to that of Chaff, while the second approach could be viewed more as "random probing" of the search space for information, similar to (Refalo 2004).

Our approach assumes that elements representing sources of global difficulty exist. Using the terminology of (Joslin & Clements 1998), *globally* difficult elements are elements which are difficult across large parts of the search space while *locally* difficult elements are difficult only in the context of their current state of search. These authors point out that identification of difficult elements through static analysis of the problem is sometimes possible but interactions between constraints can be quite complex. Often it is only through search that these intricate relations come to the fore.

The next section provides background and definitions. The following section describes the basic algorithms used in this paper. The next two sections give the results of experiments with our approaches, the first section deals with problems with insoluble cores, the other deals with problems where the sources of contention are less clearly defined. The following section gives an analysis of weight changes using these approaches, to provide a more in-depth understanding of these methods, The last section discusses issues involved in this type of learning and outlines our conclusions.

Background

A constraint satisfaction problem (CSP) is defined in the usual way, as a tuple (V, D, C) where: $V = \{V_1, ..., V_n\}$ is a set of variables which must be assigned values; $D = \{D_1, ..., D_n\}$ is the set of domains for those variables consisting of possible values which may be assigned to the variables; and $C = \{C_1, ..., C_m\}$ is the set of constraints. Constraints express relations $Rel(C_j)$ among domain values that that can be assigned to the variables in the scope of the constraint, $(Vars(C_j))$. In this paper we are concerned with binary constraint problems, so a constraint has at most two variables in its scope. Two variables are said to be *neighbors* if there is a constraint between them, i.e. if $\exists\ C_j$ s.t. $(X_a, X_b) \in (Vars(C_j))$, then X_a and X_b are neighbors.

An assignment is a set of tuples $A = \{(V_1, a), (V_2, b), ..., (V_k, h)\}$, each tuple consisting of a different instantiated variable and the value that is assigned to it in search. A solution to a problem is an assignment $A = \{(V_1, a), (V_2, b), ..., (V_n, x)\}$ that includes all variables and that does not violate any constraint.

A binary constraint satisfaction problem has an associated constraint graph, where a variable is represented by a node in the graph and a constraint C_j is represented by an edge between the two nodes in $(Vars(C_j))$. CSPs are also characterized by certain basic parameters. Thus, the *domain size* of a variable is simply the number of values in its domain. The *degree* of a variable is the number of edges connected to it. A constraint satisfaction problem is said to be arc-consistent when every value in every domain has at least one support in the domain of every neighboring variable (all other values are removed), and every variable has at least one consistent value in its domain.

The work described in this paper is concerned with complete search. It is based on depth-first backtracking search, where a value is assigned to each variable until either a complete consistent assignment has been found or a dead-end (in the form of an inconsistent assignment) has occurred. All of our work employs a hybrid version of backtracking and propagation or look-ahead techniques, termed MAC (for maintaining arc-consistency), where the problem is made arc-consistent after every assignment, i.e. all values which are arc-inconsistent given that assignment are removed from the current domain of their variable. If during this process a domain wipeout 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 search backtracks.

Description of the Algorithms

The basic weighted-degree heuristic was described in the Introduction. It should be noted that when selecting a variable, the edge-weight-sum for each variable is based only on edges connecting it to *uninstantiated* variables. These sums are compared, and the variable with the largest sum is selected as the next variable for instantiation. In this work, we use a variant of the heuristic in which the weight information is incorporated into the minimum domain/forward-degree heuristic to produce a minimum domain/weighted-degree heuristic (dom/wdeg for short).

We propose two main methods for combining dom/wdeg with restart. In both methods, restarts occur after a fixed number of nodes have been explored in the search tree (fixed cutoff C). The reason that we do not use a restarting approach such as (Luby, Sinclair, & Zuckerman 1993) which varies the cutoff, is that we do not want to bias the information learnt in favour of runs where the cutoff was large. If one were to vary the cutoff then more weight would be assigned to the variables that were sources of contention in the larger runs, which would localize the information learnt. In addition, there is a fixed maximum number of runs, R, which gives a maximum number of restarts = $R - 1$. If a solution is found or the problem is proven insoluble before the Rth run, the process terminates at that point. Otherwise, it continues until the final run, at which point the cutoff is removed (increased to ∞). This makes the method complete.

The reasoning behind the two methods represent fundamentally different strategies for learning. In the first method, WTDI, the dom/wdeg heuristic is used to guide search throughout, and edge-weights are carried along from restart to restart. Since weights are constantly being updated the heuristic ensures that search is unlikely to visit the same part of the search space twice.

This algorithm was tested with the expectation that:

- for the *easier* problems in the set, a solution may be found without restarting or after a small number of restarts (un-

less the cutoff is too low).

- for *harder* problems, this approach, which learns from each search attempt, may begin to solve problems within the cutoff once it has enough information to make good variable selections early in search.
- for the *hardest* problems, it will most likely reach the final run where complete search is used. Here we expect that the weights learnt from all the previous runs should provide the heuristic with a better basis for making initial selections, and thus the problem will be solved more quickly.

In the second method, RNDI, a variable is selected randomly at each variable selection point during search for the first $R-1$ runs. Weights are incremented in the usual way (i.e. when a constraint causes a domain wipeout). On the final restart dom/wdeg is used in the normal way, but with weight information learnt from these random probes of the search space to enable it to make better early decisions. This method was tried with the expectation that random orderings would allow information to be gathered from more diverse areas of the search space, thus providing a better sample from which to identify global bottlenecks in the problem. The disadvantage of this method is that for many problem sets it is unlikely to solve even easy problem instances before the Rth run.

It should be noted that the use of restarts in the RNDI approach is mainly for information gathering purposes. Unlike WTDI we are not expecting to solve the problem before the cutoff. In this sense it differs from most other work which use restarts as a method to avoid thrashing (Gomes, Selman, & Kautz 1998), (Guddeti & Choueiry 2005).

Finding good parameters for C and R is an important aspect of our work. Currently our approach depends on trial and error; however automating the process is the subject of ongoing research. In (Grimes & Wallace 2006) we analyzed various values of C and R in combination and found that a good value for R was much more important than the value for C, provided C was large enough to ensure learning (i.e. C $>$ the number of variables). "Good" values for C and R can be different for the two approaches on a problem set. WTDI benefits more from a larger cutoff, as it increases the likelihood of solving the problem within the cutoff. RNDI benefits from larger R with smaller C, since this provides a more diverse sample of the search space.

In the following experiments values were chosen lexically on all problem sets reported except on the soluble 200 variable set for which a promise value ordering heuristic was used (based on (Geelen 1992)). This heuristic calculates, for each value in the domain of the current variable, the product of the supporting values in the domains of *neighboring* uninstantiated variables, and chooses as the next value the one with the largest product. This has been shown to be effective in boosting search when solving soluble problems.

Note: For all RNDI experiments reported in the following sections, each cutoff and restart pair were tested ten times and the averages of the ten experiments are reported. This was done to obtain adequate samples under randomization and to avoid spuriously good (or bad) effects due to random selections.

Problems with Insoluble Cores

Firstly we will provide results for some obvious cases where these two methods are appropriate, namely on problems with embedded insoluble subproblems. (It should be noted that (Hemery *et al.* 2006) have independently proposed using the weighted-degree heuristic with restarts as part of a process for extracting minimal unsatisfiable cores from constraint networks).

For these types of problems the smallest proof of insolubility occurs when the variables in the insoluble subproblem are selected first in search. However identifying an insoluble subproblem is NP-complete in itself. By restarting and collating information regarding failures in search we hope to identify these globally difficult elements to provide search with the best variable ordering from the outset.

The first two types of problems come from a CP solver competition [1] (queens-knights problems and random 3-Sat instances); the last problem set we generated ourselves. In the queens-knights problem one must place n queens and k knights on an n x n chessboard so that no two queens can attack each other, and the knights form a cycle (when considering knight moves). The problem is insoluble when the number of knights is odd. There are two different ways in which this problem can be formulated: in the first a queen and a knight can share a square on the board (qk-n-k-add), in the second they cannot share a square (qk-n-k-mul).

For the insoluble case (odd k), the min-domain/degree (dom/deg) heuristic performs badly because it assigns the n queens first (as their domains are of size n whereas the knights' domains are of size n^2), and then tries to assign the first knight at which point search backtracks. Normal backtracking search will then try all combinations of values for the n queens selected (backtracking when it reaches the knights' variables), without realizing that it is the knights that are the source of difficulty.

Table 1. Results For Unsatisfiable Embedded Problems
Average Search Nodes

	$\frac{dom}{fdeg}$	$\frac{dom}{wdeg}$	WTDI	RNDI
qk-25-5-add	$>$ 2M	116.9K	(2.625K) 625	(2.625K) 625
qk-25-5-mul	$>$ 2M	112.6K	(3.625K) 625	(2.625K) 625
ehi-85-297	(61 $>$ 100K) 63521	1160.3	(224.2) 14.2	(176.2) 21.1
ehi-90-315	(58 $>$ 100K) 58012	1008.6	(252.0) 30.0	(174.2) 23.5
composed	(100 $>$ 100K) -	13953	(425.96) 45.96	(2075.0) 75.0

Notes: In this and later tables, numbers in brackets for WTDI and RNDI represent total search nodes including preprocessing.

[1] http://cpai.ucc.ie/06/Competition.html

The dom/wdeg heuristic starts off identically to dom/deg until enough weight has been added to one of the knight's constraints to make up for its larger domain. This knight is then consistently selected after each backtrack (it can be thought of as incrementally moving up the ordering) until search has backtracked to the first variable. The problem is then proven insoluble when all values for the first (queen) variable have been tried with all values for the knight variable. Clearly if search had started by selecting one of the knights first it would have proven the problem insoluble in n^2 nodes (i.e. tried every value for that knight).

The second type of problem from the solver competition was originally proposed in CSP format in (Bacchus 2000). It is the dual encoding of 2 classes of easy random 3-Sat instances with an embedded unsatisfiable subproblem. The first class, *ehi-85*, has 100 problems, each containing 297 variables (85 in the original sat problems), the second class, *ehi-90*, also has 100 problems, each containing 315 variables (90 in the original sat problems).

The last problem set reported in Table 1 is a set of 100 "composed" random problems, consisting of a main underconstrained component in the form $< n, d, m, t >$ where n is the number of variables, d the uniform domain size, m the graph density of the component and t the constraint tightness, and k satellite components also in this form attached by links $< m, t >$. For our problems, the main component was <100,10,0.15,0.05>, there were 5 satellites, all <20,10,0.25,0.5> and links were <0.012,0.05>.

For WTDI, if the cutoff is so low that no wipeouts occur on the first run it will repeatedly explore the same part of the search space. Thus it needed a larger cutoff than RNDI. For example on the qk-25-5 instances, when WTDI had a cutoff of 1000, it solved the problem in 2 restarts on qk-add and 3 restarts on qk-mul. For the same problems RNDI had 20 restarts with a cutoff of 100, and identified the knights as the source of contention.

Further Experiments

The experiments in this section were carried out on random binary CSPs which take the form $< n, d, m, t >$ as defined before. The problem parameters are based on the problem sets used by (Boussemart *et al.* 2004) in their study of the weighted degree heuristic; the only difference is that each of the present problems consisted of a single connected component. This was done by first connecting the nodes of the constraint graph by a spanning tree and then adding edges at random until the required number had been chosen.

The parameters are <200,10,0.0153,0.45> and <80,10,0.1042,0.65>. In one case (the 200 variable problem set), soluble and insoluble problems were selected from the original set so that they could be tested separately.

The main results are shown in Table 2. Somewhat surprisingly, WTDI never improved on the original weighted degree heuristic, and in fact was sometimes appreciably worse. It only solved 14 problems in the Random-200 soluble set and 2 problems in the Random-80 set before the final run. RNDI, on the other hand, always led to improved performance although the differences were not large.

Table 2. Results For Random Binary Problems
Average Search Nodes Over 100 Problems

	$\frac{dom}{fdeg}$	$\frac{dom}{wdeg}$	WTDI	RNDI
Random-200 Soluble	107K	37.9K	(46.5K) 42.0K	(36.2K) 31.2K
Random-200 Insoluble	115.5K	41.2K	(47.1K) 42.1K	(39.3K) 34.3K
Random-80 (54/100 sol)	242.6K	186.9K	(229.3K) 221.3K	(183K) 175K

Notes: For Random-200 problem sets, R = 10 and C = 500. For Random-80 problem set, R = 40 and C = 200.

For purposes of analysis, we also ran a further set of experiments on the random binary problems where the weights were "frozen" after the $R - 1$th run, i.e. there was no learning on the final run, (cf. Table 3). This was done to obtain information about the effectiveness of weight increments during the initial runs without contamination from updates during the final run of search. As can be seen the weights learnt by WTDI made the heuristic perform worse; the opposite was true for RNDI since it performed equally well on the Random-200 problem sets and actually improved performance on the Random-80 set.

Table 3. Results with Frozen Weights
Average Search Nodes Over 100 Problems

	Random-200 Soluble	Random-200 Insoluble	Random-80 (54/100 insoluble)
Frozen WTDI	(74.0K) 69.4K	(59.1K) 54.1K	(276.3K) 268.4K
Frozen RNDI	(37.1K) 32.1K	(38.8K) 33.8K	(146.8K) 138.8K

Notes: For Random-200 problem sets R = 10 and C = 500. For Random-80 problem set R = 40 and C = 200.

This provides more compelling evidence concerning the quality of information learnt by both approaches. A puzzling aspect of these results is that learning during the final run can actually hinder search in the RNDI approach (results for Random-80 problems). This could be ascribed to the interaction between the weights learnt during preprocessing and the weights learnt on the final run. Since we are learning from a relatively small sample in our preprocessing (only 8000 nodes in this case), the weights learnt from it can be overcome by the weights learnt on the final run, thereby changing the order of the variables (except for the first variable selected).

It should be noted that, in these procedures, the extra work done before solving is simply heuristic search using either weighted-degree or random variable ordering. Furthermore, the total constraint checks for the RNDI approach averaged 100K to 100M better than dom/wdeg depending on the problem set.

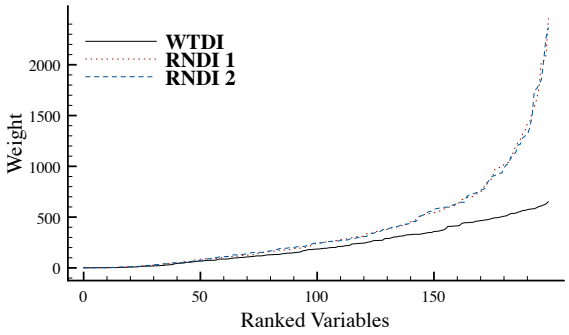

Figure 1: Variable weights with R = 100 and C = 500 using random variable ordering for weight learning.

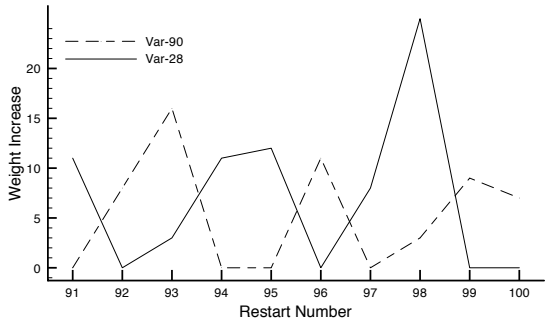

Figure 2: Weight Increase per restart for last 10 restarts, 500C.

Analysis of Weight Changes Produced by Each Strategy

In order to better understand the differences between WTDI and RNDI we ran both with 100 restarts and a cutoff of 500 nodes on selected problems of the 200-soluble problem set. The five problems chosen for these tests were sufficiently difficult that they would not be solved within the specified cutoff before the final run. Each variable's cumulated edge-weight was saved after every restart.

After 100 restarts the sum of weights across all variables was 50% to 100% greater for RNDI than for WTDI. This means that when using random variable ordering there were 50% to 100% more domain wipeouts than when using dom/wdeg for information gathering.

One explanation for the greater effectiveness of RNDI is that it provides better discrimination among variables, especially those with the largest weights. Figure 1 shows typical weight plots for a WTDI experiment and for two RNDI experiments on the same problem after 100 restarts with a 500 node cutoff. In all three cases the variables were ranked according to their weight after the 100th run.

The slope of the line indicates the level of discrimination between successive variables. Note that for both random orderings the slope is very steep for the top 50 variables and maintains a better level of discrimination over the next 50 variables than WTDI. The slope for WTDI has a much more gradual inclination, which indicates that even after 100 restarts there are no variables with a clearly larger weight than their predecessor.

WTDI suffers from something that we call *variable convection*, where different variables get weighted on each run, changing the order each time. This effect occurs because failures don't normally occur until at least 4 or 5 variables have been assigned; thus the first few variables chosen rarely receive any weight in a given run. This is beneficial in the context of restarting, however it does affect the method's ability to identify globally difficult elements.

As a result, although WTDI can identify a subset of variables which are the sources of contention, it does not offer good discrimination between them. This is quite similar to "squeaky wheel optimization" (Joslin & Clements 1998) where difficult elements get handled earlier upon restart, and because they are handled earlier they cause less "trouble", eventually falling back down the ordering to the point where they are sources of difficulty again.

Figure 2 illustrates the notion of variable convection on two highly ranked variables. The weight increase per restart for two variables over the final 11 restarts out of the 100 is shown. The two variables were selected because they were chosen first by the heuristic on 8 out of the last 11 runs, var-90 was chosen first on restart 90, 93, 94, 96; var-28 was selected first on restart 95, 98, 99 and on the final run (which isn't included since there was no cutoff). Note that their weight increases were 0 on the runs when they were chosen first. Then when other variables achieved a weight larger than theirs they fell back down the ordering and their weight increased again.

These results show that RNDI results in more stable patterns of edge-weights among variables across restarts. This, in turn, implies that variables that are truly 'critical', i.e. those associated with global difficulties in the sense used above, can be found reliably in the sequence of repeated runs with this strategy of probing for failure.

Discussion

Domain/weighted-degree is a powerful heuristic that already demonstrates the benefits of learning from failure. Nonetheless, we have shown that further information can be extracted and utilised to improve search even more. Both of our approaches improved on the original heuristic when tested on problems with clearly defined sources of global difficulty, in some cases by two orders of magnitude.

When the sources of global difficulty were not as clear cut, e.g. on the random binary problems, RNDI consistently outperformed both dom/wdeg and WTDI, although the magnitude of the improvement was not appreciable on some problem sets if one considers the preprocessing.

WTDI performed poorly on these sets as it struggled to provide clear discrimination in weights between variables. In our analysis of weight change we have demonstrated why this is the case. However the WTDI approach might be better suited to a geometric restarting strategy which would still

guarantee completeness. Although this approach would bias towards weights learnt in the previous run, it is a strategy that we are currently testing.

As in most learning from search approaches (Sleeman, Langley, & Mitchell 1982), there are important blame assignment issues. When a wipeout is found during arc consistency maintenance, one doesn't know that a given constraint is solely to blame for a variable's domain going empty even if it removed its last value. It could be that the constraint of a neighboring variable had removed most of the values and thus it should be this which is weighted. In this context, a significant advantage of the weighted degree heuristic is that the weight it gives for each individual wipeout is small. It is the cumulative effect of a variable's constraints directly causing domain wipeouts on a number of occasions that determines whether that variable will be selected over another.

The means to be able to distinguish between local and global difficulty is an important aspect of this work. (Tompkins & Hoos 2004) presented experimental data in the SAT domain which suggested that clause weighting local search algorithms do *not* benefit from their weights on a global scale and that all learning is local. That is, they found that restarting with these weights performed no better than restarting with weights assigned randomly to clauses. We contend that the method for search used in generating weights highly influences the 'globality' of the weight. In particular, deterministic approaches such as our WTDI approach, or a min-conflicts local search approach biases the weights learnt. Therefore the 'warped landscape' generated by these weights may not be a true reflection of the difficulty of the problem as a whole.

RNDI learnt from more diverse parts of the search space and thus gave a better approximation to areas of global contention. Although this strategy is not suitable for solving easy problem instances (the cost of probing the search space is likely to outweigh the benefit of the information learnt) we have shown that on hard problem sets, information gathering prior to solving can be beneficial. Furthermore, the RNDI process could be augmented by using a heuristic such as dom/wdeg for the first restart, with a cutoff large enough to allow it to solve easy instances of the problem set. This would avoid biasing the overall information learnt for the harder problems.

References

Bacchus, F. 2000. Extending forward checking. In Dechter, R., ed., *Principles and Practice of Constraint Programming-CP'00. LNCS No. 1894*, 35–51.

Beck, J. C.; Prosser, P.; and Wallace, R. J. 2003. Toward understanding variable ordering heuristics for constraint satisfaction problems. In *Proc. Fourteenth Irish Artificial Intelligence and Cognitive Science Conference-AICS'03*, 11–16.

Beck, J. C.; Prosser, P.; and Wallace, R. J. 2005. Trying again to fail-first. In *Recent Advances in Constraints. Papers from the 2004 ERCIM/CologNet Workshop-CSCLP 2004. LNAI No. 3419*, 41–55.

Boussemart, F.; Hemery, F.; Lecoutre, C.; and Sais, L. 2004. Boosting systematic search by weighting constraints. In *Proc. Sixteenth European Conference on Artificial Intelligence-ECAI'04*, 146–150.

Eisenberg, C., and Faltings, B. 2003. Using the breakout algorithm to identify hard and unsolvable subproblems. In Rossi, F., ed., *Principles and Practice of Constraint Programming-CP'03. LNCS No. 2833*, 822–826.

Geelen, P. A. 1992. Dual viewpoint heuristics for binary constraint satisfaction problems. In *Proc. Tenth European Conference on Artificial Intelligence-ECAI'92*, 31–35.

Gomes, C. P.; Selman, B.; and Kautz, H. 1998. Boosting combinatorial search through randomization. In *Proc. Sixteenth National Conference on Artificial Intelligence-AAAI'98*, 431–437.

Grimes, D., and Wallace, R. J. 2006. Learning from failure in constraint satisfaction search. In Ruml, W., and Hutter, F., eds., *Learning for Search: Papers from the 2006 AAAI Workshop, 24-31. Tech. Rep. WS-06-11*.

Guddeti, V. P., and Choueiry, B. Y. 2005. Characterization of a new restart strategy for randomized backtrack search. In *Recent Advances in Constraints. Papers from the 2004 ERCIM/CologNet Workshop-CSCLP 2004. LNAI No. 3419*, 56–70.

Haralick, R. M., and Elliott, G. L. 1980. Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence* 14:263–314.

Hemery, F.; Lecoutre, C.; Sais, L.; and Boussemart, F. 2006. Extracting MUCs from constraint networks. In *Proc. Seventeenth European Conference on Artificial Intelligence-ECAI'06*, 113–117.

Hulubei, T., and O'Sullivan, B. 2006. The impact of search heuristics on heavy-tailed behaviour. *Constraints* 11:157–176.

Joslin, D., and Clements, D. 1998. Squeaky wheel optimization. In *Proc. Sixteenth National Conference on Artificial Intelligence-AAAI'98*, 340–346.

Luby, M.; Sinclair, A.; and Zuckerman, D. 1993. Optimal speedup of las vegas algorithms. In *Israel Symposium on Theory of Computing Systems*, 128–133.

Moskewicz, M.; Madigan, C.; and Malik, S. 2001. Chaff: Engineering an efficient SAT solver. In *Proc. Design Automation Conference*, 530–535.

Refalo, P. 2004. Impact-based search strategies for constraint programming. In Wallace, M., ed., *Principles and Practice of Constraint Programming-CP'04. LNCS No. 3258*, 557–571.

Sleeman, D.; Langley, P.; and Mitchell, T. M. 1982. Learning from solution paths: An approach to the credit assignment problem. *AI Magazine* 3:48–52.

Smith, B. M., and Grant, S. A. 1998. Trying harder to fail first. In *Proc. Thirteenth European Conference on Artificial Intelligence-ECAI'98*, 249–253.

Tompkins, D. A. D., and Hoos, H. H. 2004. Warped landscapes and random acts of SAT solving. In *Proc. Eight International Symposium on Artificial Intelligence and Mathematics-ISAIM04*.