Solving Dynamic Constraint Satisfaction Problems: Relations between Problem Alteration and Search Performance *

Richard J. Wallace, Diarmuid Grimes and Eugene C. Freuder

Cork Constraint Computation Centre and Department of Computer Science
University College Cork, Cork, Ireland
email: {r.wallace,d.grimes,e.freuder}@4c.ucc.ie

Abstract

This paper presents a new analysis of dynamic constraint satisfaction problems (DCSPs) and a new approach to solving them. We first show that even very small changes in a CSP, in the form of addition of constraints or changes in constraint relations, can have profound effects on search performance. These effects are reflected in the amenability of the problem to different forms of heuristic action and in the promise and fail-firstness of variable ordering heuristics applied to the problem. This may account for the poor performance of classical DCSP methods. We then show that the same changes do *not* markedly affect the locations of the major sources of contention in the problem. A technique, called "random probing", that performs a careful assessment of this property and uses the information during subsequent search, performs well even when it only uses information based on the original problem in the DCSP sequence. The result is a new approach to solving DCSPs that is based on a robust strategy for ordering variables rather than on robust solutions.

1 Introduction

An important variant of the constraint satisfaction problem is the "dynamic constraint satisfaction problem", or DCSP. In this form of the problem, an initial CSP is subject to a sequence of alterations to its basic elements (typically, addition or deletion of values or constraints). As a result of these changes, assignments that were solutions to a previous CSP in the sequence may become invalid, which means that search must be repeated to find a solution to the new problem.

Strategies that have been devised to handle this situation fall into two main classes [VJ05]:

- Efficient methods for solving the new problem, using information about the affected parts of the assignment.
- Methods for finding "robust" solutions that are either more likely to remain solutions after change or are guaranteed to produce a valid solution to the altered problem with a fixed number of assignment changes.

*This work was supported by Science Foundation Ireland under Grant 00/PI.1/C075.

Proceedings of the 15th International RCRA workshop (RCRA 2008):
Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion
Udine, Italy, 12–13 December 2008

To date, methods that have been devised for solving DCSPs (first category above) have not taken into account specific characteristics of problems - before and after change. Some of this neglect is likely due to simple oversight, but it may also be because methods have not been available for assessing problem characteristics that change or do not change after problem alteration. Further advances in this field may be possible if alterations in search across a sequence of altered problems can be characterised. In particular, this kind of analysis may suggest new ways of carrying over information learned before alteration to the altered problem.

In this work, we show that relatively small changes in the constraints of a CSP can result in dramatic changes in the character of search. Thus, problems that are relatively easy for a given search procedure can be transformed into much harder problems, and vice versa. Then, using recently developed methods for evaluating the performance of variable ordering heuristics [BPW05] [Wal08], we show that these changes have effects on the amenability of problems to different forms of heuristic action (specifically, build up of contention versus simplification of the remaining, or future, part of the problem), as well as the quality of performance when search is on a solution path and when it is not.

Next, we examine the performance of a standard algorithm for solving DCSPs. This method attempts to conserve the original assignment, while still conducting a complete search. This turns out to be a poor strategy, since partial assignments for CSPs often 'unravel' when this procedure is used, leading to tremendous amounts of thrashing.

We then show that there are problem features that are not significantly affected by the changes we have investigated. In particular, the major places of contention within a problem are not greatly changed. This discovery suggests that a heuristic strategy that is based on evaluating these sources of contention can perform efficiently even after problem change. We show this to be the case. More specifically, we show that a heuristic procedure that uses failures during iterated sampling (that we call "random probing") continues to perform effectively after problem change, *using information obtained before such changes* and thus avoiding the cost of further sampling.

A discussion of terminology is given in the following section. Section 3 contains a description of experimental methods. Section 4 present results on the extent of change in search performance after relatively small changes in the problem. Section 5 presents results based on a classical method for solving DCSPs, called "local changes". Section 6 then presents an analysis of random probing when this procedure is applied to DCSPs. Section 7 gives conclusions.

2 Definitions and Notation

Following [DD88] and [Bes91], we define a dynamic constraint satisfaction problem (DCSP) as a sequence of static CSPs, where each successive CSP is the result of changes in the preceding one. In the original definition changes could be due ei-

ther to the addition or deletion of constraints. While adhering to this definition, we enlarge on it somewhat by defining restrictions on the type of change. In particular, we allow either additions alone, deletions alone, additions and deletions together, and finally a particular version of the latter where additions and deletions always pertain to the same sets of variables (same scopes). The latter case can, therefore, be described as changes in the tuples constituting particular relations. In addition, we consider DCSPs with specific sequence lengths, where "length" is the number of successively altered problems starting from the "base" problem. This allows us to sample from the set of DCSPs whose original CSP is the same.

We use the notation $P_{ij}(k)$ to indicate the kth member in the sequence for $DCSP_{ij}$, where i is the (arbitrary) number of the initial problem in a set of problems, and j denotes the jth DCSP generated from problem i. Since we are considering a set of DCSPs based on the same initial problem, this problem is sometimes referred to as the "original" or "base" problem for these sequences.

3 Experimental Methods

DCSPs were prepared using either of the following models of generation:

1. Addition and deletion of k constraints from a base CSP.

2. Replacement of k relations in a base CSP.

In these models, an initial CSP is generated (the base problem), and then a series of changes are made, *in each case starting with the same base problem*. In these cases, therefore, each alteration produces a new DCSP of length 1, i.e. a DCSP consisting of the base and a single altered problem. (Elsewhere, we discuss models that involve a sequence of CSPs derived from a single base problem; the issues raised are not materially different from those discussed here.) In all cases, care was taken to avoid deleting and adding constraints with the same scope in a single alteration, i.e. for each alteration the scopes for additions and deletions were mutually exclusive. In both models the number of constraints remains the same after each alteration.

The initial experiments were done with random problems generated in accordance with Model B [GMP+01]. The basic problems had 50 variables, domain size 10, graph density 0.184 and graph tightness 0.369. Problems with these parameters have 225 constraints in their constraint graphs. Although they are in a critical complexity region, these problems are small enough that they can be readily solved with the algorithms used.

DCSP sequences were formed starting with 25 independently generated initial problems. In most experiments, three DCSPs of length 1 were used, starting from the same base problem. Since the effects we observed are so strong, a sample of three was sufficient to show the effects of the particular changes we were interested in.

In the initial experiments (next section), two variable ordering heuristics were used: maximum forward degree (fd) and the FF2 heuristic of [SG98] ($ff2$). The latter chooses a variable that maximises the formula $(1-(1-p_2^m)^{d_i})^{m_i}$, where m_i is the current domain size of v_i, d_i the future degree of v_i, m is the original domain size, and p_2 is the original average tightness. These heuristics were chosen because they are most strongly associated with different basic heuristic actions on this kind of problem, as assessed by factor analytic studies of heuristic performance [Wal08]: (i) buildup of contention as search progresses, and (ii) simplification of the future part of the problem. Because of their associations, $ff2$ can be referred to as a contention heuristic, while fd is a simplification heuristic, although it should be borne in mind that the difference is one of degree. These heuristics were employed in connection with the maintained arc consistency algorithm using AC-3 (MAC-3). The performance measure was search nodes.

Most tests involved search for one solution. To avoid effects due to vagaries of value selection that might be expected if a single value ordering was used, most experiments involved repeated runs on individual problems, with values chosen randomly. The number of runs per problem was always 100. In these cases, the datum for each problem is mean search nodes over the set of 100 runs.

Later experiments were done with simplified scheduling problems, used in a recent CSP solver competition (http:/www.cril.univ-artois.fr/ lecoutre/benchmarks/ benchmarks.html). These were "os-taillard-4" problems, derived from the Taillard benchmarks [Tai93], with the time window set to the best-known value (os-taillard-4-100, soluble) or to 95% of the best-known value (os-taillard-4-95, insoluble). Each of these sets contained ten problems. For these problems, constraints prevent two operations that require the same resource from overlapping; specifically, they are disjunctive relations of the form, $X_i + k_1 \leq X_j \bigvee X_j + k_2 \leq X_i$. These problems had 16 variables, the domains were ranges of integers starting from 0, with 100-200 values in a domain, and all variables had the same degree. In this case, the original heuristics used were minimum domain/forward degree and Brélaz.

Scheduling problems were perturbed by increasing the upper bound of six of the domains by ten units. As a result, the altered problems had features intermediate between the os-taillard-4-95 and the os-taillard-4-100 problems. (In the latter problems, the upper bounds of all domains are ten greater than their counterparts in the 4-95 set.) All perturbed problems were also insoluble. Single or repeated runs were carried out; in the latter case, value ordering was randomised by choosing either the highest or lowest remaining value in a domain at random.

4 Search Performance after Problem Alteration

Our first objective was to get some idea about the degree to which search performance is altered after small to moderate changes in a problem. To the best of our knowledge, data of this kind have not been reported previously in the literature.

The changes made in these experiments are well within the limits of previous experiments in the literature (e.g. [VS94b] [VS94a]), and in some cases are much smaller; in particular, we consider changes in existing relations as well as changes in the constraint graph.

4.1 Changes in performance in altered problems

Table 1 shows results for the first five sets of DCSPs in an experiment on problem alterations involving addition and deletion of constraints. (Similar patterns were observed in the remaining 20 sets of DCSPs.) We see that pronounced changes in performance can occur after a limited amount of alteration (deletion of 5 constraints out of 225 and addition of 5 new ones). In some cases where the original problem was more amenable to one heuristic than the other, this difference was reversed after problem alteration (e.g. $ff2$ was more efficient than fd for problem $P_{3\_}(0)$, but for $P_{32}(1)$, it was distinctly worse). This indicates that relative amenability to one or the other form of heuristic action can also change after small alterations in the constraints. Since the numbers in the table are means of 100 runs per problem in which values were chosen randomly from the remaining candidates, simple differences due to location of the first solution in a value ordering can be ruled out as contributing to these variations in performance. (For purposes of comparison with results in Section 6, we note that grand means over all 75 altered problems were 2601 for fd and 3561 for $ff2$.)

Table 1. Examples of Performance Change with Small Problem Perturbations

prob (i)	$P_{i-}(0)$	$P_{i1}(1)$	$P_{i2}(1)$	$P_{i3}(1)$
		fd		
1	600	1303	705	1266
2	2136	4160	2407	1569
3	1682	1794	1697	2027
4	318	755	586	1507
5	2804	4996	1425	1270
		$ff2$		
1	670	1280	1412	1004
2	3222	3990	2521	1582
3	924	1385	2385	968
4	713	1129	1027	941
5	3359	4549	2952	1780

Notes. <50,10,0.184,0.369> problems. Each datum is mean search nodes for 100 runs with random value ordering. Problems were altered by adding and deleting 5 constraints. $P_{i-}(0)$ is base problem for each of three altered problems found on the same row.

The number of solutions for each problem tested in Table 1 is shown in Table 2. In general, there is little correspondance between differences in number of solutions and differences in performance. (Note, for example, the change in solution count between $P_{1\_}(0)$ and $P_{12}(1)$, which is more than an order of magnitude, and

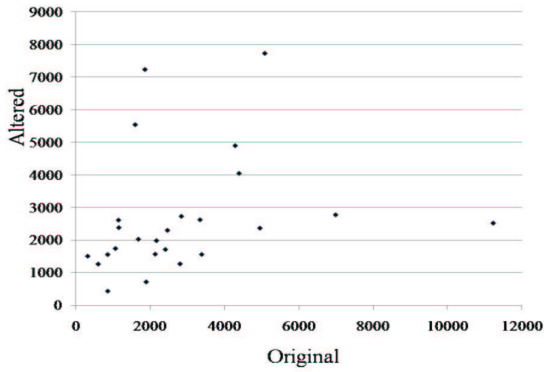

Figure 1: Scatter plot of search effort (mean nodes over 100 runs) with fd on original versus $P_{i3}(1)$ problems with five constraints added and deleted. (Overall correlation in performance between original an altered problems is 0.24.)

the small change in performance [which is not in the expected direction].) Across all 100 problems (including both original and perturbed), the correlation between search effort and number of solutions was -0.2 for each heuristic, for either single or repeated runs per problem. Although this is in the expected direction, the small magnitude shows that very little of the variation in performance is related to this factor.

Similar results were found for a corresponding set of problems without solutions. (Parameter values were identical except for density which was 0.19.) Differences of 2-3:1 in search effort were common, although very large differences were not as frequent as in solvable problems. In addition, noticeable effects were found for solvable problems with the same problem parameters when only one constraint was added and deleted, or 5 relations changed, or only 1 relation changed.

Table 2. Solution Counts for Problems in Table 1

prob	$P_{i-}(0)$	$P_{i1}(1)$	$P_{i2}(1)$	$P_{i3}(1)$
1	7,846	43,267	109,480	12,065
2	18,573	29,722	47,392	79,147
3	65,735	3,550	8,843	28,427
4	26,505	37,253	17,751	7,282
5	20,156	16,434	12,033	79,384

To give a more concrete idea of the extent of variation between the original and altered problems, a scatter plot is shown in Figure 1. This is for the $P_{i3}(1)$ problem set and includes all 25 problems.

4.2 Changes in promise and fail-firstness

Evidence concerning the basis of variation in performance for a given heuristic can be obtained by using measures of search quality based on the Policy Framework of [BPW04] [BPW05]. In this framework, quality is measured under two conditions: (i) when search is on a solution path, i.e. the present partial assignment can be extended to a solution, (ii) when a mistake has been made and search is in an insoluble subtree. In the first case, an optimal policy would maximise the likelihood of remaining on the solution path; in the second, an optimal policy would minimise the size of the refutation (insoluble subtree) needed to prove the incorrectness of the initial wrong assignment. These policies are referred to as the "promise" and "fail-first" policies, and measures of adherence to each policy have been developed, which are referred to by the same names.

The promise measure is basically a sum of probabilities across all complete search paths. Values can vary between 0 and 1, where a value of 1 means that any value in the domain will lead to a solution. The fail-first measure is the mean "mistake tree" size, where a mistake tree is an insoluble subtree rooted at the first non-viable assignment (i.e. the initial 'mistake'). A larger mean mistake tree size therefore indicates poorer fail-firstness. In this paper, the analysis is based on an all-solutions search. To avoid artifacts due to averaging over different frequencies of mistakes at different search levels, comparisons were restricted to a single level of search.

Representative results for each measure are given in Table 3, for the first five sets of DCSPs. (These are for fd; similar results were found for $ff2$.) Promise and fail-firstness both show differences across a set of perturbed problems, although the promise measure is more drastically affected. Adherence to different policies sometimes changes in concert (e.g. problem 4), while in other cases it changes in opposite ways (e.g. problem 5). (Comparison with the results in Table 1 shows the impact of these changes on overall search effort.)

5 Performance of an Algorithm Based on Solution Reuse

Local changes is a complete algorithm designed to find solutions to an altered problem while conserving as much of the original assignment as possible [VS94b]. It works by determining a minimal set of variables that must be reassigned, and undoing old assignments only when they are inconsistent with the new ones.

Our version of local changes updates the classical description by using MAC; it also makes use of the basic data structures and style of control used in our basic MAC implementation. The algorithm was run with either or lexical or min-conflicts value ordering. (In the latter case, values are chosen to minimize the number of conflicts with values in neighboring domains; this was used in the original paper of [VS94b]). For comparison with other data, the original solution was always found using lexical ordering.

With the present problems, local changes performs quite poorly, in spite of the fact that 5 additions and deletions forces only 1-3 variables to be unassigned. Basically, as the algorithm attempts to find new assignments, it progressively undoes the old assignment, and since this is done repeatedly, there is tremendous thrashing. As a result, the number of nodes in the search tree is sometimes orders of magnitude greater than when search with MAC is done from scratch. Thus, with lexical value ordering, the mean for the 75 perturbed problems was 854,544 with ff2 and 11,579,654 with fd. With min-conflicts value ordering the corresponding means were 156,032 and 6,463,863. (Each mean includes 12-13 cases in which the old solution was still valid, so the number of search nodes was 0.)

It should be noted that the experiments in the original report on this algorithm were based on problems with only 15 variables with domain sizes between 6 and 16. In addition, the versions of local changes in that paper were based on simple backtracking or backtracking with forward checking.

6 Results with an Algorithm that Samples Contention

The results described thus far all suggest that problems undergo marked changes after small alterations in their constraint graph topology or even in the patterns of support. However, it is still possible that certain fundamental features of problems do *not* change after such alterations. A possible feature of this type is the pattern of contention in a problem, especially the variables that are major sources of contention. Earlier work has shown that this feature can be assessed by tallying domain wipeouts during search [BHLS04] [GW07]. In this section, we show that this feature exhibits much less variability than do direct measures of performance. We also show that information related to this feature can be used to solve altered problems in a DCSP with considerable efficiency.

Table 3. Promise and Fail-Firstness Measures on Individual Problems (5 constraints added and deleted; fd heuristic)

	promise X 1000				mean mistake tree size at level 1			
prob i	orig	$P_{i1}(1)$	$P_{i2}(1)$	$P_{i3}(1)$	orig	$P_{i1}(1)$	$P_{i2}(1)$	$P_{i3}(1)$
1	.218	.528	.905	.303	1270	1030	608	1532
2	.439	.059	.153	1.261	903	1182	1254	801
3	.492	.916	.385	.647	965	702	1191	1613
4	2.380	.540	1.702	.230	366	499	355	1117
5	.259	1.411	.251	.253	1462	2189	1151	2382

Notes. <50,10,0.184,0.369> problems. Further details in text.

6.1 The random probing procedure

The present work employed a recently developed method for assessing sources of contention prior to search, that we refer to as "random probing" [GW07]. It is based on the weighted degree heuristic ($wdeg$) of Boussemart et al. [BHLS04].

In the weighted degree approach, each constraint is given an initial weight of 1. During search, a constraint's weight is incremented by 1 each time it causes a domain wipeout (i.e. removes all values from a variable's domain) during consistency checking. The weighted degree of a variable is the sum of the weights on constraints between the variable and its uninstantiated neighbors. The heuristic $wdeg$ chooses the variable with largest weighted degree; the variant $dom/wdeg$ chooses the variable with minimum ratio of current domain size to weighted degree.

Random probing attempts to boost the power of the weighted degree heuristic by gathering information prior to search with the heuristic. The method involves a number of short 'probes' of the search space where search is run to a fixed cutoff and variable selection is random. Constraint weights are updated in the normal way during probing, but the information is not used to guide search. After the probing phase, search runs to completion using the weights from probing to guide the selections of the weighted degree heuristic beginning with the first variable in the search order.

The weights learned during the probing phase are expected to boost the fail-firstness of the heuristic by enabling it to choose the most contentious variables from the beginning of search. Since each probe is an independent sample of the search space, the weight profile generated by the probes gives an overview of the spread of contention amongst the variables in the problem.

6.2 Stability of points of contention

We first wished to determine the degree of correlation in the weights produced by random probing before and after alteration. To investigate this, we obtained weight profiles (i.e. the weighted degree of each variable after random probing) for a random sample of the 25 DCSP problems. The probing regimen was 100 restarts with a 30-weight (30-failure) cutoff. Variables were then ranked by their weighted-degree. We compared variable ranks in the original problem with ranks in the altered problems using the Spearman rank correlation coefficient [Hay73] (Table 4).

Table 4. Correlations for Weight Profiles
(5 constraints added and deleted)

problem	P1	P2	P3
1	.957	.957	.947
2	.959	.950	.932
5	.921	.915	.924
18	.885	.867	.914
23	.963	.919	.924

Five DCSP sets chosen at random from original 25.
"Pj" = $P_{-j}(1)$.

The correlation coefficient can range between -1 (where variables are ranked in the opposite order in the two cases) and 1 (where variables are ranked identically).

Here, the correlations range from .867 to .963, which shows that the rankings for the altered problems were very similar to those for the original problems. This, in turn, shows that the sources of contention remain more or less the same in spite of alterations.

6.3 DCSP search with weighted degree heuristics

In these experiments, search was for single solutions, again with 100 (experimental) runs with random value ordering. In each such test, weights were updated during the final (post-probing) run, which lasted until a solution had been found.

Problems were solved using three different forms of weighted degree: (i) $dom/wdeg$ with no restarting, (ii) independent random probing for each problem ($rndi$), (iii) a single phase of random probing on the original problems ($rndi\text{-}orig$), after which these weights were used with the original and each of its altered problems (on each of the 100 runs with random value ordering). In the third case, the new constraints in an altered problem were given an initial weight of 1.

Table 5 presents results for each approach in terms of average nodes over the 75 perturbed problems. Nodes explored during the probing phase are not included. These amounted to about 3700 nodes per problem for the entire phase. (It should be noted that the work required for probing increases much more slowly than the improvement in performance as problem size increases [GW07].)

As expected, probing resulted in a final performance that was better than that produced by the ordinary weighted degree heuristic. However, these results were only marginally better than with $rndi\text{-}orig$. The differences in means were evaluated using a paired comparison two-tailed t-test [Hay73]. The difference for $rndi$ and $rndi\text{-}orig$ was not statistically significant ($t(74) = 1.46$, $p > 0.1$), while that for $dom/wdeg$ and $rndi\text{-}orig$ was significant ($t(74) = 6.58$, $p << 0.001$). These results indicate that weights learned by probing on the original problem are still viable on the altered problems.

Table 5. Search Results with Weighted Degree (5 constraints added and deleted)

$dom/wdeg$	$rndi$	$rndi\text{-}orig$
1617	1170	1216

Notes. <50,10,0.184,0.369> problems. Mean search nodes across all altered problems.

7 Results with Scheduling Problems

Table 6 shows results with one heuristic for six of the ten os-taillard-4-95 problems; these data are averages of 50 runs per problem. As with random problems, even small perturbations greatly affect search performance.

Table 6. Examples of Performance Change after Perturbations (Scheduling Problems; dom/fd heuristic)

prob (i)	$P_{i-}(0)$	$P_{i1}(1)$	$P_{i2}(1)$	$P_{i4}(1)$
1	399,636	9837	1,240,418	2,811,257
2	395	20,913	56,146	49,867
3	1751	23,993	33,446	47,044
4	167,372	47,430	330,603	1,449,969
5	29	39	29	49
6	490	95,632	624	633

Notes. Os-taillard-4-95 problems. Each datum is mean search nodes for 50 runs with random value ordering. Problems were altered by increasing domain sizes for 6 randomly chosen variables by 10.

And as with random problems, heuristics based on information about patterns of contention outperform ordinary heuristics (Table 7). More importantly for present purposes, performance is diminished only slightly when weights derived from the original problems are used as the initial weights for the perturbed problems. (Probing was done with 100 restarts and a fixed, 30-failure cutoff.)

Table 7. Search Results with Weighted Degree on Perturbed Scheduling Problems

dom/wdeg	*rndi*	*rndi-orig*
16,745	4139	5198

Notes. Os-taillard-4-95 problems. Mean search nodes across all altered problems.

8 Conclusions

The present results show that for successive problems in a DCSP, not only must each new problem be re-solved, but if one uses ordinary solving methods, performance with a given algorithm and heuristic is highly unpredictable even after small changes in the previous problem. Problems, therefore, appear to change their intrinsic character in ways that alter a heuristic's effectiveness.

We have shown experimentally that problems can change with respect to their relative amenability to different forms of heuristic action (contention and simplification). This makes it particularly difficult for ordinary heuristics, which generally favor one or the other action [Wal08], to perform effectively across a DCSP sequence. Problem alterations have effects on both promise and fail-firstness, although the effects are proportionally greater in the first case. On this basis, one would expect promise-based strategies to be less effective than fail-first strategies. This is one reason for the poor performance of local changes.

At the same time, we find that despite these manifold effects on the characteristics of search, points of maximum contention remain relatively constant when a small number of constraints are added or relations changed. Given this result, one would predict that a heuristic procedure that asseses these points of contention will

perform effectively in this domain. Random probing has this characteristic, and it performs well when information obtained from the original problem in a DCSP sequence is used with subsequent problems. This means that probing does not have to be done again, at least with alterations of the magnitude that we have studied here.

Used in this way, random probing constitutes a new approach to solving DCSPs, in which a robust strategy for ordering variables is derived from assessments of the major sources of contention in an original CSP. (Note that the variable ordering itself is not fixed.) It is then used together with adjustments following immediate failure in the course of search. In this way, it can be used to solve a sequence of altered problems effectively without repeating the initial sampling phase.

References

[Bes91] C. Bessiére. Arc-consistency in dynamic constraint satisfaction problems. In *Proc. Ninth National Conference on Artificial Intelligence-AAAI'91*, pages 221–226. AAAI Press, 1991.

[BHLS04] F. Boussemart, F. Hemery, C. Lecoutre, and L. Sais. Boosting systematic search by weighting constraints. In *Proc. Sixteenth European Conference on Artificial Intelligence-ECAI'04*, pages 146–150. IOS, 2004.

[BPW04] J. C. Beck, P. Prosser, and R. J. Wallace. Variable ordering heuristics show promise. In *Principles and Practice of Constraint Programming-CP'04. LNCS No. 3258*, pages 711–715. Springer, 2004.

[BPW05] J. C. Beck, P. Prosser, and R. J. Wallace. Trying again to fail-first. In B. Faltings, A. Petcu, F. Fages, and F. Rossi, editors, *Recent Advances in Constraints-CSCLP 2004. LNAI No. 3419*, pages 41–55. Springer, 2005.

[DD88] R. Dechter and A. Dechter. Belief maintenance in dynamic constraint networks. In *Proc. Seventh National Conference on Artificial Intelligence-AAAI'88*, pages 37–42. AAAI Press, 1988.

[GMP+01] I. P. Gent, E. MacIntyre, P. Prosser, B. M. Smith, and T. Walsh. Random constraint satisfaction: Flaws and structure. *Constraints*, 6:345–372, 2001.

[GW07] D. Grimes and R. J. Wallace. Learning to identify global bottlenecks in constraint satisfaction search. In *Twentieth International FLAIRS Conference*, pages 592–598. AAAI Press, 2007.

[Hay73] W. L. Hays. *Statistics for the Social Sciences*. Holt, Rinehart, Winston, 2nd edition, 1973.

[SG98] B. M. Smith and S. A. Grant. Trying harder to fail first. In *Proc. Thirteenth European Conference on Artificial Intelligence-ECAI'98*, pages 249–253. Wiley, 1998.

[Tai93] E. Taillard. Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64:278–285, 1993.

[VJ05] G. Verfaillie and N. Jussien. Constraint solving in uncertain and dynamic environments: A survey. *Constraints*, 10(3):253–281, 2005.

[VS94a] G. Verfaillie and T. Schiex. Dynamic backtracking for dynamic CSP. In *ECAI'94 Workshop on Constraint Satisfaction Issues Raised by Practical Applications*, pages 73–80, 1994.

[VS94b] G. Verfaillie and T. Schiex. Solution reuse in dynamic constraint satisfaction problems. In *Twelth National Conference on Artificial Intelligence- AAAI'94*, pages 307–312. AAAI Press, 1994.

[Wal08] R. J. Wallace. Determining the principles underlying performance variation in CSP heuristics. *International Journal on Artificial Intelligence Tools*, 17(5):857–880, 2008.