Solving Dynamic Constraint Satisfaction Problems by Identifying Stable Features[*]

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) with finite domans 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 as well as overall quality of search. In addition, classical DCSP methods perform poorly on these problems because there are sometimes no solutions similar to the original one found. We then show that the same changes do *not* markedly affect the locations of the major sources of contention in the problem. A technique for iterated sampling 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 [Verfaillie and Jussien, 2005]:

- 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.

To the best of our knowledge, research in this field has not generally focussed on the nature of the change in search performance after a problem has been altered. Further advances in this field may be possible if changes 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 problem.

In this work, we show that relatively small alterations to 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 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 part of the problem).

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 for hard problems, 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 (i.e. bottlenecks that cause search to fail) are not greatly changed. This suggests that a heuristic strategy that is based on evaluating these sources of contention can perform efficiently even after problem change. We show that such a heuristic procedure (which identifies sources of contention through iterated sampling) 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 our sampling procedure applied to

[*]This work was supported by Science Foundation Ireland under Grant 05/IN/1886.

DCSPs. Section 7 gives conclusions.

2 Definitions and Notation

Following [Dechter and Dechter, 1988] and [Bessiére, 1991], 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 either to addition or deletion of constraints. For random CSPs, we consider two cases: additions and deletions together; and a variant of this 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. For CSPs with ordered domains and relational constraints, we consider changes in the maximum domain values (reductions or extensions), which also serve to tighten or loosen adjacent constraints.

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 based on random CSPs were prepared using either of the following models of generation:

1. Addition and deletion of c constraints from a base CSP.
2. Replacement of r 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. To make DCSPs of greater length, the same procedure is used but in each case the latest CSP in the sequence is used as the initial problem. In all cases, care was taken to avoid deleting and adding constraints with the same scope in a single alteration. In both models, therefore, the number of constraints remains the same after each alteration.

Experiments on random problems were generated in accordance with Model B [Gent *et al.*, 2001]. All 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 [Smith and Grant, 1998] ($ff2$). 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 [Wallace, 2008]: (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.

Experiments on problems with ordered domains involved 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 [Taillard, 1993], with the time window set to the best-known value (os-taillard-4-100, solvable) or to 95% of the best-known value (os-taillard-4-95, unsolvable). Each of these sets contained ten problems. For these problems, constraints prevent two operations that share the same job or require the same resource from overlapping; specifically, they are disjunctive relations of the form, $(X_i + dur_i \leq X_j) \bigvee (X_j + dur_j \leq X_i)$. 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 changing upper bounds of a random sample of domains. In the original problems, domains of the 4-100 problems are all ten units greater than the corresponding 4-95 problems. Perturbed problems were obtained by either increasing six of the domains of the 4-95 problems by ten units or decreasing four domains of the 4-100 problems by ten units. In both cases the altered problems had features intermediate between the os-taillard-4-95 and the os-taillard-4-100 problems. Perturbed problems were selected so that those generated from the 4-95 problems remained unsolvable, while those from the 4-100 problems remained solvable. Fifty runs were carried out on each of the 4-100 problems, and 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 (e.g. [Verfaillie and Schiex, 1994]), 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 nodes for fd and 3561 for $ff2$.)

A more adequate evaluation of variability following small changes was obtained by comparing results for ten sets of 25 random problems generated independently with ten sets of 25 problems where each set was generated by altering a common base problem, again by deleting and adding 5 constraints. In other words, each set consisted of problems $P_{ij}(1)$ where i is constant and $j = 1 \ldots 25$. To control for differences in the mean, the statistic used was the coefficient of variation, equal to the ratio of the standard deviation to the mean [Snedecor and Cochran, 1969]. Using fd as the heuristic, with the independently generated problems the range of values for this statistic over the ten samples was 0.61–1.08. For perturbed problems the range was 0.43–0.63. This is convincing evidence that the variability after small alterations is an appreciable fraction of that found with independently generated problems.

Table 2 presents further evidence on the magnitude of change following small alterations in the initial problem, based on the first ten sets of DCSPs. (Results are for fd; similar differences were found when the heuristic was $ff2$.) Percent change was calculated as the absolute difference between performance on the base problem and the altered problem, divided by the smaller of the two, times 100. The sign of the difference indicates whether search effort increased (positive change) or decreased (negative change). For example, if base performance was 1000 search nodes and performance on the altered problem was 2000, the % change is 100; if performance on the altered problem was 500 the % change is -100.

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 altered by adding and deleting 5 constraints. $P_{i-}(0)$ is base problem for each of three altered problems found on the same row. These are therefore separate DCSPs of length 1.

Although, as expected, differences tend to be less pronounced when value ordering is randomised, they are, as noted, still substantial. They are also substantial in the all-solutions case (although here there is a greater effect of the number of solutions, especially since the present code was not designed to detect clusters of solutions).

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 an analysis like that presented in Table 2, the largest percent change in the group of DCSPs based on the same initial problem ranged from -177 to 284.

Table 2. Largest Performance Change for All Alterations under Different Conditions of Search ((5 constraints added and deleted; fd heuristic)

	single run		rptd single		all sol	
prob	orig	lg(%)	orig	lg(%)	orig	lg(%)
1	509	-261	600	117	79,567	651
2	2538	97	2136	95	192,528	117
3	1152	390	1682	21	356,687	-562
4	630	209	318	374	194,877	-147
5	5303	-421	2804	-121	102,801	254
6	2975	220	4954	-109	23,272	82
7	1148	-244	1065	63	1,417,139	-986
8	11,443	290	5085	-221	38,045	132
9	757	64	1859	289	202,777	-236
10	2465	265	1597	247	216,080	-111

Notes. <50,10,0.184,0.369> problems. Numbers under "orig" are search nodes. Single run is with lexical value order. Repeated single condition is 100 runs per problem with random value selection. (Here, nodes are means.) lg% is for largest change observed over the three DCSPs.

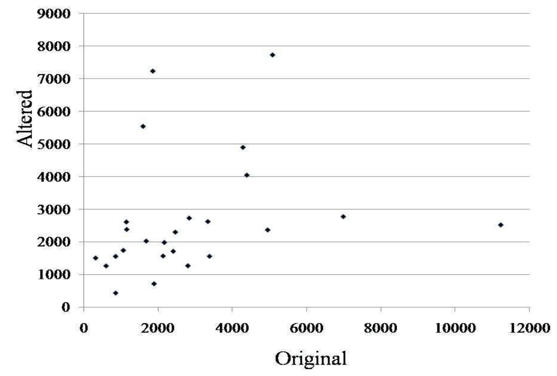

Figure 1: Scatter plot of search effort (mean nodes over 100 runs) with fd on original versus $P_{i3}(1)$ problems with five constraints added and deleted. (Overall correlation in performance between original and altered problems is 0.24.)

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.

The number of solutions for each problem tested in Table 1 is shown in Table 3. 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 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.

Table 3. 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

Table 4 presents summary data for the entire set of 25 original problems and the CSPs derived from those problems for a number of conditions. This takes the form of Pearson product-moment correlations between the set of original problems and each independent set of altered CSPs (P1, P2 and P3 in the table). It should be emphasized that this is a relatively crude measure of problem change, since large deviations from +1.0 will only occur if the relative values of an original problem ($P_{ij}(0)$ and the altered problem ($P_{ij}(1)$) with respect to other problems in the corresponding set of original or altered problems are markedly different for a sufficient number of DCSPs. That correlations appreciably lower than 1.0 are found in these experiments, indicates the frequency with which large changes in performance occur after even small changes in the original problems. Here, the lower correlations for a given type of change (i.e. within one row) are of greatest interest, since they give us an idea of how marked these effects can be.

Table 4. Correlations with Performance on Original Problems after Different Forms of Alteration

	fd			$ff2$		
condit	P1	P2	P3	P1	P2	P3
1c	.81	.83	.70	.81	.84	.67
5c	.49	.83	.24	.34	.54	.31
25c	.64	.34	.55	.46	.45	.23
1r	.92	.86	.92	.77	.71	.83
5r	.76	.84	.67	.51	.71	.56
25r	.71	.80	.85	.82	.13	.55

Notes. <50,10,0.184,0.369> problems. Single solution search with repeated runs on each problem. Condition "kc" is k deletions and additions; "kr" is k altered relations. "Pj" = $P_{-j}(1)$.

These results show that even after the addition of one constraint - or even a change in one relation - there are noticeable changes in search performance. In these cases, it was not uncommon to find correlations of .70; while this may seem high, it is associated with only a 30% reduction in the standard deviation of performance when the value of one variable is known [McNemar, 1969]. Correlations of this size were found with the max forward degree heuristic when 5 or 25 relations were altered; in this case not only does the constraint graph remain the same, but the variable ordering is also unchanged.

Table 5. Examples of Performance after Perturbations (Scheduling Problems; dom/fd heuristic)

prob i	$P_{i-}(0)$	$P_{i1}(1)$	$P_{i2}(1)$	$P_{i3}(1)$
		$os-taillard-4-95$		
0	51	578	76	557
1	390,845	10,172	1,255,215	1,865,802
2	397	22,140	58,181	66,174
3	1755	22,745	38,060	41,111
4	167,719	51,148	320,280	569,975
		$os-taillard-4-100$		
0	17	18	31	41
1	4,745,944	2,635,513	463,465	6,934,799
2	52,377	81,762	53,060	4,812
3	507,254	233,397	103,630	169,570
4	558,176	4,569,500	9,429	3,481

Notes. Table shows first 5 base problems. Data for 4-100 series is mean search nodes for 50 runs with random value ordering. Note there are three *separate* DCSPs per row, cf. Sect. 2.

A even more impressive pattern of changes in search was found for perturbed scheduling problems. In this case, order-of-magnitude differences were sometimes observed (Table 5; Note that here altered problems should be more difficult on average than the base problems).

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 [Verfaillie and Schiex, 1994]. 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 data structures and style of control used in our basic MAC implementation. The algorithm was run with either lexical or min-conflicts value ordering. (In the latter case, values are chosen to minimize the number of conflicts with previous instantiations; this was used in the original paper of [Verfaillie and Schiex, 1994], possibly inspired by [Minton *et al.*, 1992]). For comparison with other data in this paper, the original solution was always found using lexical ordering.

With these random problems, local changes performs quite poorly, in spite of the fact that 5 additions and deletions forces only 1-3 variables to be unassigned initially. 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 235,321 and 6,602,715. (These latter means include 19 and 22 cases, respectively, in which the number of search nodes was < 10.)

These results can be explained by nearest-solution analyses, in which the perturbed problem is examined to find the minimum Hamming distance *of any solution* in comparison with the solution found for the the original problem (equal to n (here 50) - maximum number of matching values). This was done with a branch-and-bound search where number of mismatches was the quantity minimised. For solutions found by fd and ff2 using lexical ordering, the average minimum Hamming distance was 20 across all (75) problems; for 30% of the problems, this distance was 35 or greater. Similar results were found in more extensive tests with repeated runs, using domain/wdeg with random value ordering. Thus, part of the reason for the poor performance of local changes on these problems is that the minimum Hamming distance is sometimes much greater than the number of assignments discarded before search. (For cases noted above with < 10 nodes, minimum Hamming distances were in the range 0-7.)

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 [Boussemart *et al.*, 2004] [Grimes and Wallace, 2007]. 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.

6.1 The random probing procedure

The present work employed a recently developed method for assessing sources of contention prior to search, known as "random probing" [Grimes and Wallace, 2007]. It is based on the weighted degree heuristic ($wdeg$) of Boussemart et al. [Boussemart *et al.*, 2004]. 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. 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 selections by a weighted degree heuristic beginning with the first variable in the search order.

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 among variables in the problem.

6.2 Stability of points of contention

We first wished to determine the degree of correlation in 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 problem sets. 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 [Hays, 1973] (Table 6).

Table 6. 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 ranges 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. We also generated weight profiles for the insoluble perturbed scheduling problem set; the average correlation was 0.875. This shows that 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. For random problems, there were 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-orig*), after which these weights were used with the original and each of its altered problems (i.e. on each of the 100 runs performed with random value ordering). In the third case, all new constraints in an altered problem were given an initial weight of 1.

Table 7 presents results for each approach in terms of average nodes over the perturbed problems (75 for random, 50 for scheduling). Nodes explored during the probing phase are not included, since we were interested in examining quality of performance using the information gained by probing. These amounted to about 3100-3700 nodes per problem for the entire phase, depending on problem type. (With regard to overall efficiency, it should be noted that the work required for probing increases much more slowly than the improvement in performance as problem size increases [Grimes and Wallace, 2007] [Grimes, 2008].)

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-orig* for both random and scheduling problems. Differences in means were evaluated using paired comparison two-tailed t-tests [Hays, 1973]. For random problems, the difference for *rndi* and *rndi-orig* was not statistically significant ($t(74) = 1.46$, $p > 0.1$), while that for *dom/wdeg* and *rndi-orig* was significant ($t(74) = 6.58$, $p << 0.001$). Similar results were found for the taillard-4-95 scheduling problems; however there were no significant differences for the taillard-4-100 problems (due to variability across problems). These results show that weights learned by probing on the original problem are still viable on the altered problems.

Table 7. Search Results with Weighted Degree

problems	*dom/wdeg*	*rndi*	*rndi-orig*
random	1617	1170	1216
taill-4-95	16,745	4139	5198
taill-4-100	11,340	7972	5999

Notes. Mean search nodes across all altered problems. Random are <50,10,0.184,0.369> with 5 constraints added and deleted. Scheduling problems altered as described in text.

6.4 Probing with successive changes

A further important question regarding weights obtained on a base problem is, if there are repeated changes resulting in a sequence of CSPs, (DCSP >> length 1), over how many successive problems will these weights be effective in improving search performance?

Evidence on this question was obtained using DCSPs of length 20, based on random problems with the same parameters as before and perturbing problems by adding and deleting 5 constraints. Correlations between base problems and $P_{i1}(1)$ were .86, .61, and .66 for search effort, using fd, ff2, and dom/wdeg respectively. The correlations declined over the next few problems (e.g. they were .21, .36, .32 for the same heuristics on $P_{i1}(4)$), and were variable with low to moderate correlations thereafter.

Comparisons of search performance for *rndi-orig* and the other two strategies are shown in Table 8 and Figure 2, in each case averaged over 25 problems. Table 8 shows a comparison of probing with the original weights and ordinary *dom/wdeg* for the first eight CSPs (including the base problem). For this type of alteration, random probing remains effective over four problems after which the effect diminishes progressively. Figure 2 compares improvements due to probing across the entire sequence, for probing on each problem and probing only on the base problem

Table 8. Comparisons between Probing with Original Weights and Ordinary dom/wdeg after Successive Alterations

	P0	P1	P2	P3	P4	P5	P6	P7
dom/wdeg	1535	1523	1538	1338	1207	1187	1157	951
rndi-orig	1202	1175	1220	1102	927	1057	1034	935
df	334	348	318	235	280	130	124	16

Notes. <50,10,0.184,0.369> problems. Single solution search with 100 runs on each problem. Means of 25 problems. 5 deletions and additions at each step in sequence. "Pk" = $P_{-1}(k)$.

While, as expected, probing on each problem serves to improve performance in relation to dom/wdeg, using original weights also leads to improved performance over several perturbations. Differences in means were evaluated using a paired comparison one-tailed t-test [Hays, 1973]. The difference for *dom/wdeg* and *rndi-orig* was significant ($t(24) > 3$, $p < 0.005$, for the base and the first four perturbed problems in the sequence). For the next two problems $p < 0.1$; for 13 of the remaining 14 problems in the sequence, $p > 0.1$. (Differences between *dom/wdeg* and *rndi* were always statistically significant.) The fact that *rndi-orig* never became appreciably worse than *dom/wdeg* is probably due to the ability of weighted degree heuristics to adapt quickly to a given problem.

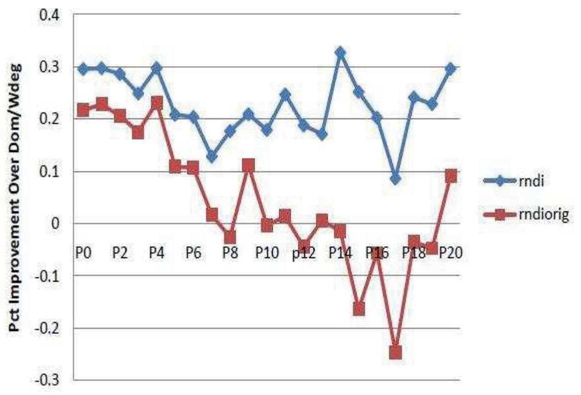

Figure 2: Percent improvement with rndi and rndi-orig in comparison with dom/wdeg on successive sets of perturbed problems. Problem set k+1 in sequence derived by adding and deleting 5 constraints in each problem in set k (Pk in figure).

7 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 [Wallace, 2008], to perform effectively across a DCSP sequence.

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 assesses these points of contention will perform effectively in this domain. Weighted degree heuristics have this characteristic. By using this general strategy together with a technique for sampling contention, it is possible to obtain information from the original problem in a DCSP sequence that can be used with subsequent problems. This means that the frequency of probing can be restricted, 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

[Bessiére, 1991] 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.

[Boussemart *et al.*, 2004] 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.

[Dechter and Dechter, 1988] 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.

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

[Grimes and Wallace, 2007] 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.

[Grimes, 2008] D. Grimes. A study of adaptive restarting strategies for solving constraint satisfaction problems. In *Proc. 19th Irish Conference on Artificial Intelligence and Cognitive Science-AICS'08.*, pages 33–42, 2008.

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

[McNemar, 1969] Q. McNemar. *Psychological Statistics*. John Wiley, New York, 4th edition, 1969.

[Minton *et al.*, 1992] S. Minton, M. D. Johnston, A. B. Philips, and P. Laird. Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems. *Artificial Intelligence*, 58:161–205, 1992.

[Smith and Grant, 1998] 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.

[Snedecor and Cochran, 1969] G. W. Snedecor and W. G. Cochran. *Statistical Methods*. Iowa State, Ames, 7th edition, 1969.

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

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

[Verfaillie and Schiex, 1994] 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.

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