Enhancements of Branch and Bound Methods for the Maximal Constraint Satisfaction Problem

Richard J. Wallace
Department of Computer Science
University of New Hampshire
Durham, NH 03824 USA
rjw@cs.unh.edu

Abstract

Two methods are described for enhancing performance of branch and bound methods for overconstrained CSPs. These methods improve either the upper or lower bound, respectively, during search, so the two can be combined. Upper bounds are improved by using heuristic repair methods before search to find a good solution quickly, whose cost is used as the initial upper bound. The method for improving lower bounds is an extension of directed arc consistency preprocessing, used in conjunction with forward checking. After computing directed arc consistency counts, inferred counts are computed for all values based on minimum counts for values of adjacent variables that are later in the search order. This inference process can be iterated, so that counts are cascaded from the end to the beginning of the search order, to augment the initial counts. Improvements in time and effort are demonstrated for both techniques using random problems.

Introduction

Constraint satisfaction problems (CSPs) involve assigning values to variables which satisfy a set of constraints. In some cases, problems may be overconstrained so there is no assignment that will satisfy all of the constraints. In these cases it may still be useful to have an assignment that satisfies the most important constraints or, if constraints have equal weight, one that satisfies as many constraints as possible.

Because they return *guaranteed* optimal solutions, complete methods have a special role in the area of partial constraint satisfaction that cannot be filled by other methods such as heuristic repair. However, partial constraint satisfaction problems can be very difficult to solve by complete methods. As Schiex et al. have noted (Schiex *et al.* 1995), this is because the weights of violated constraints must be added to determine the quality of an assignment, and addition is non-idempotent, in contrast to the Boolean AND operator used in ordinary CSPs. In the latter case, a single falsified constraint is sufficient to stop search along a given path, while in the former a sum based on the violated constraints must be greater or equal to some critical value. This explains why difficult problems are found for every combination of problems parameters, and the difficulty is greater when there are more inconsistencies in an optimal solution (Freuder & Wallace 1992) (Schiex *et al.* 1995).

These considerations underscore the importance of efficient complete algorithms for solving overconstrained problems. This paper describes new methods for enhancing the performance of such algorithms. These methods build on earlier work in which depth-first search was used with the branch and bound strategy (Freuder & Wallace 1992).

The first procedure to be described is a method for improving the upper bound at the beginning of search. The strategy is simple. A heuristic repair procedure is run prior to complete search, and the cost associated with the best solution found by hill-climbing is used as the initial upper bound. The major portion of the analysis of this method is the assessment of improvements in performance as a function of the quality of this initial upper bound. This analysis is intended to answer these questions: If the initial upper bound is equal to the optimal cost, what is the reduction in search effort? (This is the best case possible, in which search serves merely to certify optimality.) And over what range of costs is there significant improvement in performance, i.e., how close to optimality must this bound be to get significant reduction? Following this analysis, the quality of the upper bound that can be obtained by some well-known hill-climbing methods is assessed experimentally, and the tradeoff between quality of bound and hill-climbing effort is considered.

The second method is both independent and complementary to the first since it involves augmenting the lower bound calculation, especially early in search when more substantial improvement is possible. This method uses information obtained from arc consistency tests carried out before search. Its use is, therefore, restricted to problems with arc consistency violations. The information takes the form of *inconsistency counts*, i.e., tallies for each value a of variable v_i, of the number of domains v_j that do not have any values consistent with a in the constraint between v_i and v_j. In the past these counts have been used to compute tighter lower bounds and to order domain values so as to find solutions with fewer inconsistencies earlier in search (Freuder & Wallace 1992). The present method is based on a specialized type of inconsistency count, called *directed arc consistency counts* (DACCs), that refer to variables before or after the current value in a specified search order (Wallace 1995).

The new method uses DACCs for variables later in the search order to infer inconsistencies for the values of variables that will be instantiated earlier. For example, if value a in the domain of v_i is supported by values in the domain of v_j that have counts of one or more, then, if a is assigned to v_i, a count can be deduced for a. If such infererences are carried out systematically, DACCs can be carried forward in the search order, i.e., DACCs can be "cascaded". In this way, it may be possible to derive tighter lower bounds at the beginning of search.

The next section, 2, presents some background for this work. Section 3 analyses the effect of initial upper bounds at varying distances from the optimal and evaluates heuristic methods for finding initial upper bounds. Section 4 describes cascaded DAC procedures for generating better lower bounds and evaluates these methods. Section 5 presents results for these methods in combination. Section 6 gives conclusions.

Background: Basic Concepts

A constraint satisfaction problem (CSP) involves assigning values to *variables* that satisfy a set of *constraints* among subsets of these variables. The set of values that can be assigned to one variable is called the *domain* of that variable. In the present work all constraints are binary. A binary CSP is associated with a *constraint graph*, where nodes represent variables and arcs represent constraints.

CSPs can be characterized by four parameters: number of variables, domain size, number of constraints, and number of acceptable (or unacceptable) tuples in a constraint. The number of constraints can be stated in relative terms: as the proportion of possible constraints or problem *density*. (In this paper values for density are always in terms of number of possible arcs that can be added to a spanning tree.) Similarly, the size of a constraint can be described as the number of unacceptable tuples relative to the total possible, where the latter is the cardinality of the Cartesian product of the domain sizes of variables associated with that constraint. This fraction is the *tightness* of the constraint.

Branch and bound algorithms associate each path through a search tree with a cost that is non-decreasing in the length of the path. Search down a given path can stop when the cost of the partial assignment of values to variables is at least as great as the lowest cost yet found for a full assignment. The latter, therefore, sets an *upper bound* on the cost. In addition to calculating the cost at a particular node, projected cost can be calculated to produce a higher, more effective, *lower bound*. The present algorithms use the number of violated constraints incurred by partial assignments of values to variables as the cost, called the *distance* of a partial solution from a complete solution. Since a minimum distance is obtained when a maximal number of constraints is satisfied, these problems are called maximal constraint satisfaction problems or MAX-CSPs.

Branch and bound algorithms have been developed for MAX-CSPs that are analogues of CSP algorithms (Freuder & Wallace 1992). Retrospective algorithms compare successive assignments of values to variables with previous assignments to maintain a consistent partial solution. The most powerful branch and bound algorithm of this type is an analog of backmarking. Prospective algorithms compare each assigned value against domains of unassigned (or future) variables to determine which values in these domains are consistent with values already assigned. In this way inconsistencies can be recorded prospectively, so that degree of inconsistency for a given value is known at the time of assignment. The best of these is an analog of forward checking, in which only the future domains of neighboring variables are checked for inconsistency with the value considered for the current assignment.

The efficiency of branch and bound algorithms is enhanced by variable and value ordering. Two useful heuristics for variable ordering are, (i) a combination of maximum width at a node (i.e., maximal connectivity with prior variables in the search order) and smallest domain size, which is especially effective with retrospective algorithms, and (ii) decreasing degree of a node, which is very effective with prospective algorithms. These heuristics work best when problems have relatively sparse constraint graphs (Wallace & Freuder 1993) (Wallace 1995). Ordering values by increasing number of inconsistencies, determined either during preprocessing or in the course of search, also improves search efficiency.

Upper Bound Techniques Using Heuristic Repair

Analysis of Initial Upper Bound Effects

An important question in this context is the degree of improvement to be expected from an initial upper bound of a given quality. This can be assessed experimentally by comparing search effort when the initial upper bound is set to 'infinity' with search when the initial upper bound is a specified distance from the optimum. This evaluation is possible if the true optimal distance is known; in the present work this information was obtained from an earlier run of a complete algorithm. This evaluation gives some indication of, (i) the relative effects of different values of the initial upper bound, (ii) how close this bound must be to the optimal distance to effectively reduce search effort. With this information we can assess the usefulness of heuristic techniques in this context and make judgements about parameterization of the heuristic procedure, specifically, the length of time that the procedure should be run in view of the tradeoff between hill-climbing effort and subsequent search reduction.

The first experiments were done with 15-variable problems in order to test the effect of the initial upper bound value on both weaker and stronger algorithms. (Note that, given the marked difficulty of partial CSPs, this problem size is within the usual range of those tested, cf. (Larrosa & Meseguer 1995) (Schiex *et al.* 1995). In addition, 30-variable problems were tested to confirm results with smaller problems and to assess the effect of problem size. In this case only the more efficient prospective algorithms were tested. The 15- and some of the 30-variable problems were of fixed density with varying domain size and constraint tightness. Variability was produced by randomly choosing a value between one and some maximum. For domain size, the maximum was nine; for constraints, the maximum number of acceptable tuples was one less than the product of the sizes of the relevant domains. (These will be called Type 1 problems.) A second type of 30-variable problem was also tested, in which for an initial set of values or constraint tuples, each element was included with a certain fixed probability. (These "Type 2" problems are more homogeneous in their quantitative features than Type 1 problems.) In all cases, a sample of 25 problems was generated. All tests were run on a DEC Alpha (DEC3000 M300LX), using compiled Lisp code. Program correctness was verified by automatically checking each solution to see that it had the number of violations reported and by cross-checking means and standard deviations for optimal distances for a set of problems in different runs.

Tables 1-2 show results for 15-variable Type 1 problems at the lowest density. Table 1 shows results for variants of backmarking, Table 2 for algorithms based on forward checking. When the upper bound is equal to the optimal distance, search effort is reduced by a factor of 1.5-6. The fall-off in efficiency as the value of the initial upper bound increases is nonlinear, with the greatest change near the optimal distance. Nonetheless, there is a range of values above the optimum that still afford appreciable improvement. This improvement must, of course, be compared with the hill-climbing effort required to produce these values.

In these tests improvement due to the initial upper bound did not cancel the effectiveness of techniques such as variable ordering and lower bound calculations. In contrast, value ordering based on inconsistency counts had no effect when the initial upper bound was optimal. However, when the upper bound had a higher value, such ordering was still effective.

Several variants of forward checking were also tested with problems of the same size having constraint graph densities of 0.33 and 0.66. Representative data are shown in Figure 1. This figure shows the average re-

Table 1: Effect of Initial Upper Bound on Retrospective Algorithms

init ub	BM lex	BM wd	BM lex ACv	BM wd ACv	RPO lex	RPO wd
infinite	6228	16.9	1355	14.9	1097	9.7
optim.	994	4.8	996	5.4	746	3.4
opt.+1	2570	8.4	1293	12.8	1040	8.2
opt.+2	4535	13.5	1343	13.5	1086	8.2
opt.+3	5865	14.4	1352	14.7	1094	9.5

Note. 15-variable problems, density = 0.16. Means for 25 problems. Constraint checks in thousands (preprocessing included). BM is backmark; lex, lexical variable ordering; wd, width/domain variable ordering; ACv, value ordering by increasing arc consistency counts; RPO, "retrospective-prospective-ordering" procedure of Freuder and Wallace, or BM/AC with AC counts also used in lower bound calculations.

duction in search effort in relation to the initial upper bound in terms of the difference from the optimum distance. Despite the variability of the data, there is an overall trend toward a lower reduction as density increases. In addition, in all but two cases (FCdeg33 and DAC33 in Figure 1) the reduction when the initial upper bound was more than 2 above the optimum was ≤ 0.05.

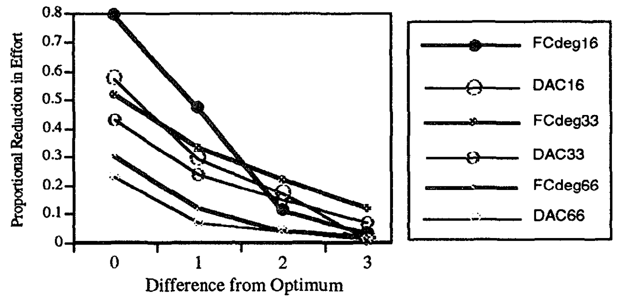

Figure 1: Improvement in performance in relation to quality of the initial upper bound for two different algorithms: forward checking (with degree ordering) and forward checking with directed arc consistency counts. 15-variable problems at densities of 0.16, 0.33 and 0.66.

Figure 2 shows similar results for 30-variable problems, using forward checking with directed arc consistency, and variable and value ordering (DACdeg/ ACval). The same trend toward diminishing effect of the upper bound technique with increasing density is also seen, although the range of densities is more limited. However, significant reduction in search effort is found for a greater range of initial bound values (up to 3-4

Table 2: Effect of Initial Upper Bound on Prospective Algorithms

init ub	FC lex	FC lex FCv	FC deg	FC deg FCv	DAC deg ACv	FC dyd FCv
infinite	88	83	10.9	9.9	3.8	4.6
optim.	28	28	2.0	2.0	1.6	2.6
opt.+1	58	55	5.4	5.3	2.7	3.7
opt.+2	79	75	9.0	8.8	3.2	4.5
opt.+3	81	78	10.0	9.6	3.8	4.6

Note. Problems and measures as in Table 1. FC is forward checking; deg, variable ordering by decreasing degree; dyd, dynamic ordering by minimum domain size; FCv, value ordering by increasing forward checking counts; ACv, value ordering by increasing arc consistency counts; DAC, FC with directed arc consistency counts.

above the optimum in most cases) than for the smaller problems. These trends were confirmed on these problems using forward checking (FCdeg/FCval).

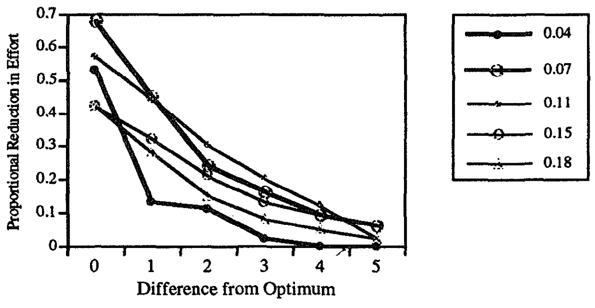

Figure 2: Improvement in performance in relation to quality of the initial upper bound. 30-variable problems with varying density. Algorithm is forward checking with degree ordering and directed arc consistency counts.

In the experiments described so far, increasing density was accompanied by an increase in magnitude of the mean optimal distance. Hence, it is not clear which factor is responsible for the changing effectiveness of the initial upper bound. These effects were separated by using Type 2 problems, where the average constraint tightness could be varied along with density to give problems at different densities having the same average optimal distance. The results in Table 3 indicate that when the optimal distance is held roughly constant, increasing density has no clear-cut effect on improvement due to the initial upper bound. On the other hand, when the distance is greater, the effectiveness of this procedure is diminished.

Table 3: Reduction in Search Effort in Relation to Initial Upper Bound

	density				
	low optimum			high optimum	
init ub	0.10	0.30	0.50	0.10	0.30
optim.	.69	.86	.75	.44	.56
opt.+1	.52	.68	.55	.20	.41
opt.+2	.36	.54	.32	.13	.27
opt.+3	.19	.37	.15	.08	.18
opt.+4	.11	.29	.08	.05	.12
opt.+5	.04	.16	.04	.02	.09

Note. 30-variable problems with average optimal distance of about 2 (low optimal distance) or 8.5 (higher optimal distance). Varying constraint graph densities. Table entries are reduction in effort as a proportion of the search effort starting with an upper bound of 'infinity'. Algorithm is FCdeg/FCval.

Upper Bounds From Heuristic Procedures

To get some idea of the performance of hill-climbing techniques in this domain, three heuristic repair procedures were tested. In some cases the procedures described in the literature were altered to enhance performance. The procedures were, (i) min-conflicts hill-climbing (Minton *et al.* 1990), with a random walk component, as described for GSAT (Selman & Kautz 1993); the probability of random walk was set to 0.10, since this had been found to be a good value in earlier work, (ii) the breakout procedure (Morris 1993), (iii) weak commitment search (Yokoo 1994), without the addition of nogoods during search; in earlier work it was found that the proliferation of these nogoods quickly slows down search, while the remainder of the procedure works quite well without this feature. In these tests, weak commitment search (weakcom) performed somewhat better than mincon-walk, which, in turn, did better than breakout.

Representative *anytime* curves are shown in Figure 3 for the three heuristic procedures for one set of 30-variable Type 1 problems. These curves are based on five runs of the procedure per problem. Solutions within one cost unit of the optimum were found for almost all problems after 1-2 seconds. For this set of problems, these procedures performed 30-40,000 constraint checks per second, so good solutions were found after an average of less than 80,00 constraint checks.

The effectiveness of these procedures is demonstrated in Figure 4. The branch and bound algorithm used in these tests was forward checking with directed arc consistency (DACdeg/ACval). Since these tests were assessments of the efficiency of heuristic repair methods in practice, each procedure was run only once with each cutoff value. For these problems, the mean performance of the basic algorithm (initial upper bound = 'infinity') was: for density = 0.07,

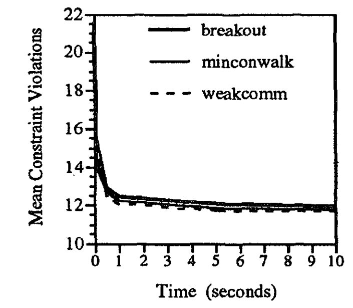

Figure 3: Anytime curves for three heuristic repair methods. Curves show mean number of violations (distances) in the best solution found after increasing durations. 30-variable problems with density = 0.11 and mean optimal distance of 11.7.

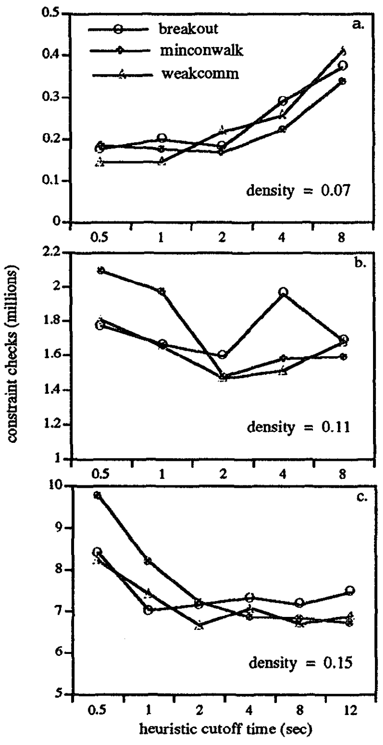

Figure 4: Total constraint checks (for heuristic repair *and* branch and bound) when best distance found by heuristic procedure before the cutoff time is used for the initial upper bound. 30-variable problems. Branch and bound algorithm is DAC with degree and ACval ordering. For each problem set, by the last cutoff, weak commitment and mincon-walk found optimal distances for all problems.

335,693 constraint checks, for density = 0.11, 3,137,590 constraint checks, for density = 0.15, 10,931,337 constraint checks. In each case there is a range of cutoff times that give markedly better results. Performance of branch and bound with optimal distances as initial upper bounds may be taken as the ideal reduction (impossible to attain in practice since some work must be performed to obtain the initial upper bound). For problems with 0.07 density, the ideal reduction was 0.68, while the maximum reduction for each procedure varied from 0.56 (weakcom) to 0.48 (breakout). For 0.11 density, the ideal reduction was 0.57, and the maximum reduction varied from 0.53 (weakcom) to 0.49 (breakout). For 0.15 density, the ideal reduction was 0.42, and the maximum reduction varied from 0.39 (weakcom) to 0.35 (breakout). It appears, therefore, that hill-climbing procedures can give results that are close to the best that one could possibly obtain through use of good initial upper bounds.

Directed Arc Consistency and Cascaded DACCs

Background: Directed Arc Consistency and Forward Checking for MAX-CSPs

The procedure for deriving directed arc consistency counts (DACCs) is shown in Figure 5. A variable search order is established at the outset. Arc consistency checking is then carried out in one pass through the ordered list of variables. Checking is done in one direction only; in the present work this is in the forward direction (i.e., against future variables in the search order) because forward DACCs can be used with forward checking. At each step, values in a domain are checked against values in the domain of each future variable that shares a constraint with the current variable. If a value in the current domain has no supporting values in a future domain, the DACC for that value is incremented by one.

```
Establish search order for variables
Set DAC-count for each domain value to 0

For each variable $v_i$
    for each value $a$ in domain $d_i$
        for each variable $v_j$ later than $v_i$ in the
        ordering such that $v_i$ and $v_j$ share
        a constraint
            if there is no value $b$ in domain $d_j$
            such that $(a, b)$ is in the constraint
            between $v_i$ and $v_j$
                increment the DAC-count for $a$
```

Figure 5: Directed arc consistency for MAX-CSPs. Here, checking is in forward direction, i. e., each value is tested for support in domains of future variables.

In forward checking for MAX-CSPs (Freuder & Wallace 1992) (Shapiro & Haralick 1981), a value a being considered for assignment to variable v_i is tested against values in the domain of each uninstantiated (future) variable v_j, that shares a constraint with v_i. If b, in the domain of v_j, is inconsistent with a, then an inconsistency count associated with b is incremented. This forward-checking count (FC count) is a kind of backward DACC, which is only valid for the current instantiation of the set of past variables. If the sum of this count and the current distance is as great as the current upper bound, b does not have to be considered for instantiation, given the current partial instantiation, including a. Lower bounds can be increased by adding the sum of the minimum FC counts for future domains. After directed arc consistency preprocessing, lower bounds can be augmented by two more factors: the DACC for b and the sum of the minimum forward DACCs.

Cascaded Directed Arc Consistency

Here a method is described for enhancing the effect of DACCs, especially at the beginning of search where pruning is likely to be most effective. After the basic DACCs have been obtained, as in Figure 5, better estimates are derived for the minimal inconsistencies associated with each value, based on the DACCs of values later in the search order. These improved estimates are then used during backtrack search to enhance lower bounds. The procedure starts at the end of the search order and works backwards, so that inferences of eventual nonsupport in the form of DACCs can be carried forward. (For this reason these counts are referred to here as "cascaded" DACCs.)

The manner of inferring inconsistency from DACCs is indicated in Figure 6. In this figure variables v_i and v_j share a constraint, and v_j appears after v_i in the search order. In addition, value a in the domain of v_i is supported by values b and c in v_j's domain (i.e., (a, b) and (a, c) are acceptable tuples), but not by d and e. The original DACCs are shown beside each value. (Those for a are from other constraints.) The upper diagram depicts a simple inference, the lower diagram a cascaded inference. Pseudocode for the cascading procedure is also shown in Figure 7.

First, consider the upper diagram in Figure 6. In this case we can infer that, if a is assigned to v_i, a minimum cost of 1 can be deduced from the counts for the values of v_j, although the original DACC for this constraint is zero, since a is supported. This inference is possible because, (i) the minimum DACC for the supporting values is 1, (ii) the minimum DACC for the nonsupporting values is 0, but, if either d or e is assigned to v_j, an additional cost associated with this constraint will be incurred, which allows us to increment this minimum. As a result, we can increase the DACC for value a to 3.

When inferences are cascaded (lower diagram of Figure 6), the minimum counts must be adjusted, if the

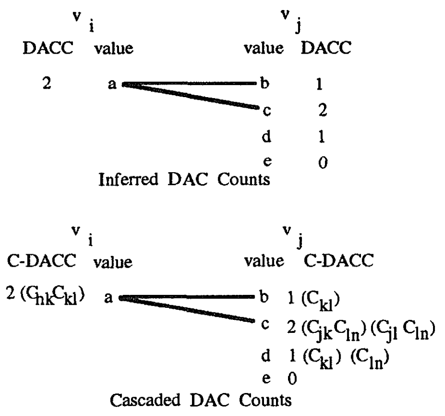

Figure 6: Scheme for inferred DACCs and cascaded (inferred) DACCs. a-e are values in domains of v_i and v_j, lines from a to b and c indicate that a is supported by these values. (C_{xy} ..) is a tag consisting of the constraints that contribute to a DACC. Further explanation in text.

(possibly cascaded) count for a is based on some of the same constraints as the cascaded DACCS for v_j's values. This is done with "tags" that denote the original violated constraints that each DACC is based on, and which are brought forward whenever a DACC is augmented and added to each set of tags for that value. In the present example, a, b, c and d each have one or more sets of tags, the size of each set being equal to the current DACC for that value. Since one of a's tags is the same as one of b's, the minimum inferred DACC for supporting values must be decremented by one, to zero. In this case we could not carry any inferred counts forward to augment the DACC for a. If, however, b's tag had been C_{ln}, a's DACC would be incremented, and this tag would be added to each of a's tag sets.

In the present implementation of cascaded directed arc consistency, the original DAC procedure is altered from an AC-3 style to an AC-4 style procedure, to collect sets of supporting and non-supporting values in each future adjacent domain. This obviates further constraint checking during cascading. Efficiency is also enhanced by using an integer representation for constraint-tags and by removing duplicate tags and performing subsumption tests on tag sets.

In the present work, variable ordering was by decreasing degree of the node in the constraint graph. Intuitively, this ordering should be well-suited for cascading DACCs to the front of the search order, since the initial variables should receive DACCs from a large number of adjacent variables. Cascaded DACCs can, of course, be used to order values in each domain.

Unfortunately, use of cascaded DACCs, as opposed to ordinary DACCs, involves an important tradeoff. In the former case, the sum of the minimum DACCs

```
Find DACCs (as shown in Figure 5), 'tagging' them with
the constraints whose violation produced these counts.

For variable $v_i = n - 2$ to 1 with respect to search order
  For each value $a$ in domain $d_i$
    For each variable $v_j$ later than $v_i$ in the
    ordering such that $v_i$ and $v_j$ share a constraint
      Let min-suppct = minimum DACC for values
        in $v_j$ that support $a$.
      Let min-nonsuppct = minimum DACC for values
        in $v_j$ that do not support $a$.
      If there are supporting values in $v_j$, increment
        min-nonsupport by one.
      Let min-suppct' = min-suppct adjusted for
        constraints common to DACCs associated with $a$
      Let min-nonsuppct' = min-nonsuppct adjusted for
        constraints common to DACCs associated with $a$
      Add min (min-suppct', min-nonsuppct') to the
        DACC for $a$ and update tag sets on which this
        DACC is based.
```

Figure 7: Generation of cascaded directed arc consistency counts (cascaded DACCs).

in each future domain (used for 'global' lower bound calculations) must be restricted to variables that are not connected to the current variable with respect to variables between them in the search order. Otherwise this sum could include counts that are redundant with the cascaded DACC for the current value. On the other hand, use of one lower bound estimate during search does not preclude use of the other, although there will be some extra overhead. For this reason, a version of search that used the maximum of the calculations based on either ordinary DAC or cascaded DAC counts was also tested.

Table 4 shows results of tests with the Type 1 30-variable problems. Times are given in addition to constraint checks because of differences in overhead involved in different procedures. Use of cascaded DACCs, particularly in combination with ordinary DACCs (having better 'global calculations'), resulted in consistent improvements in both measures, especially for the hardest problems.

Combining Upper and Lower Bound Methods

The techniques described in Sections 3 and 4 were tested together on the 30-variable Type 1 problems. In the first phase, a single run of weak commitment search was used to find an initial upper bound. Cutoff times for this phase were based on earlier tests reported above and were chosen conservatively to insure near-optimal bounds. This intention was met: for each problem set, all or all but one of the initial upper bounds were equal to the optimal distance, and in the remaining cases the bound was one greater than the optimum. Complete search procedures were then run with the initial bound for each problem equal to that found by the hill-climbing procedure.

Table 4: Experimental Results with Cascaded DACCS

algorithm		density				
		.04	.07	.11	.15	.18
FC	ck	286	1,499	12,274	38,418	
/FCv	s	17	74	571	1470	
DAC	ck	112	336	3,138	10,931	43,755
/ACv	s	6	13	110	350	1439
DACcasc	ck	97	327	3,105	10,507	35,960
/ACv	s	5	13	116	350	1209
DACcmb	ck	93	264	2,497	8,347	29,783
/ACv	s	5	13	100	312	1053
opt. dist.		4.0	6.4	11.7	14.7	19.8

Note. Constraint checks (ck) in thousands. Times (s, for seconds) include arc consistency preprocessing. Variable ordering by decreasing degree. FC is forward checking; FCv, value ordering by increasing forward checking counts; ACv, value ordering by arc consistency counts; DAC, FC with 'ordinary' directed arc consistency counts; casc, FC with cascaded DACCs; comb, FC using ordinary and cascaded DACCs in lower bound calculations.

In Table 5, results for the initial phase of heuristic repair are shown separately. The expected results were found: not only did the better initial upper bounds improve performance in all cases, but under these conditions the benefit due to cascaded DAC procedures was at least as great as when they were used alone.

Table 5: Results with Initial Upper Bounds Derived from Hill-Climbing and Cascaded DACCs

algorithm		density				
		.04	.07	.11	.15	.18
heuristic	ck	19	39	182	382	797
repair	s	3/4	1	4	8	16
FC	ck	123	390	4,175	23,324	
/FCv	s	8	17	146	864	
DAC	ck	53	108	1,342	6,322	26,055
/ACv	s	3	4	40	197	778
DACcasc	ck	39	116	1,358	6,016	22,283
/ACv	s	2	4	40	199	712
DACcomb	ck	37	82	864	4,453	17,758
/ACv	s	2	4	29	153	585

Notes for Table 4 apply. Compare data with that table.

Tests were also made with 40-variable problems, with densities of 0.03 (Table 6) and 0.08. In the former case, weak commitment search was run for 4 seconds to obtain initial upper bounds. Optimal solutions were found in all but three cases by the hill-climbing procedure; in the latter, the distance was one greater than the optimum. Here the combination of upper bound calculation and cascading DACCs (in combination with ordinary DACCs) gave improvements of more than an order of magnitude compared to forward checking with ordinary DAC counts. Moreover, in five cases (20%) with combined DACC strategies, no backtrack search was needed, since the initial lower bounds for all values of the first variable in the ordering were at least as large as the initial upper bound. (In comparison, there was only one such case when ordinary DACCs were used.)

Table 6: Results for 40-variable Problems

algorithm		no ub calc	ub calc	
			heur.	B&B
DACdeg	ck	1,356	126	173
/ACv	s	99	4	10
DACdegcasc	ck	1,221	126	144
/ACv	s	78	4	8
DACdegcomb	ck	1,174	126	100
/ACv	s	90	4	8

Note. Density = 0.03. Means of 25 problems. Constraint checks in thousands, time in seconds. Mean optimal distance = 4.6. Abbreviations as in Table 4.

For 40-variable Type 1 problems, problem difficulty increased enormously as density increased. (The mean distance increased in this case from 4.6 for density = 0.03 to 15.0 for density = 0.08.) The 0.08 density problems were solved by forward checking with ordinary DAC counts after 274 million constraint checks, on average. The combination of cascaded and ordinary DACCs gave a mean of 113 million constraint checks. When hill-climbing was run for five seconds (performing a mean of 147 thousand constraint checks), it found optimal solutions for 19 of the 25 problems, and distances one above the optimum for the rest. With these upper bounds, the means for ordinary and combined DACCS were 105 and 60 million, respectively, a reduction of almost 1/2 in each case. For individual problems the combined strategy sometimes reduced search effort by 1-2 orders of magnitude.

Conclusions

The improvements in techniques for upper and lower bounds described in this paper have made it appreciably easier to solve some PCSPs of moderate size with complete methods. There is some indication that these methods become more useful as problem size increases, since the results for 40-variable problems were generally more impressive than those for smaller problems. It should be emphasized that many of these problems became quite easy to solve when these methods were used, although they were not easily solved by simpler algorithms, so that few or even no constraint checks were required during the phase of exhaustive search.

Acknowledgments. This material is based on work supported by the National Science Foundation under Grant Nos. IRI-9207633 and IRI-9504316.

References

Freuder, E., and Wallace, R. 1992. Partial constraint satisfaction. *Artif. Intell.* 58:21–70.

Larrosa, J., and Meseguer, P. 1995. Optimization-based heuristics for maximal constraint satisfaction. In Montanari, U., and Rossi, F., eds., *Principles and Practice of Constraint Programming - CP '95*, LNCS No. 976. Heidelberg: Springer-Verlag. 103–120.

Minton, S.; Johnson, M. D.; Philips, A. B.; and Laird, P. 1990. Solving large-scale constraint satisfaction and scheduling problems using a heuristic repair method. In *Proceedings AAAI-90*, 17–24.

Morris, P. 1993. The breakout method for escaping from local minima. In *Proceedings AAAI-93*, 40–45.

Schiex, T.; ; Fargier, H.; and Verfaillie, G. 1995. Valued constraint satisfaction problems: Hard and easy problems. In *Proceedings IJCAI-95*, 631–637.

Selman, B., and Kautz, H. A. 1993. Domain-independent extensions to gsat: Solving large structured satisfiability problems. In *Proceedings IJCAI-93*, 290–295.

Shapiro, L., and Haralick, R. 1981. Structural descriptions and inexact matching. *IEEE Trans. Pattern Anal. Mach. Intell.* 3:504–519.

Wallace, R. J., and Freuder, E. C. 1993. Conjunctive width heuristics for maximal constraint satisfaction. In *Proceedings AAAI-93*, 762–768.

Wallace, R. J. 1995. Directed arc consistency preprocessing as a strategy for maximal constraint satisfaction. In Meyer, M., ed., *Constraint Processing*, LNCS No. 923. Heidelberg: Springer-Verlag. 121–138.

Yokoo, M. 1994. Weak-commitment search for solving constraint satisfaction problems. In *Proceedings AAAI-94*, 313-318.