

Neighbourhood SAC for Preprocessing and Search

Richard J. Wallace

Insight Centre-4C
University College Cork, Cork, Ireland
email: r.wallace@4c.ucc.ie

Abstract. This paper reports extensions and further analysis of a new form of singleton arc consistency, called neighbourhood SAC (NSAC), where a subproblem adjacent to the variable with the reduced domain is made arc consistent. The first part of the paper presents two important extensions: (1) NSAC is generalized to k -NSAC, where k is the longest path between the focal variable and any variable in the subgraph. (2) NSAC can be extended in a relatively straightforward way to handle n -ary constraints. The second part presents some studies of hybrid search techniques based on NSAC and SAC, using a variety of problems. Comparisons are made using the domain/degree and domain/weighted degree variable ordering heuristics. In some cases, higher levels of consistency maintenance outperform MAC by several orders of magnitude, although with weighted degree the best tradeoff is obtained when SAC-based consistency is restricted to preprocessing.

1 Introduction

Arc consistency (AC) is the best-known form of local consistency processing within the field of constraint solving and is used both for preprocessing a problem and during solution search. This is because it manages the tradeoff between processing effort and search space reduction in a very effective fashion. Singleton arc consistency (SAC) is a more powerful form of consistency which is still based on AC. As such it is often more effective than AC as a preprocessing algorithm; however, it is much more expensive. Moreover, the added expense limits its overall effectiveness when used during search.

Neighbourhood SAC is a form of singleton arc consistency in which consistency with respect to single domain values is established only with respect to the neighbourhood of the variable with a singleton domain (the “focal variable”), i.e. the subgraph that includes the latter plus its neighbours in the constraint graph. Although weaker than SAC, it is still a stronger form of consistency than AC [7]. As would be expected, this form of consistency can be established much more efficiently than full singleton arc consistency; at the same time, it often deletes many more values than simple AC, sometimes almost as many as SAC [7]. Also, since it is based on AC rather than higher-order forms of consistency, it requires very little additional code on top of that required to produce AC. As a result, it has considerable potential both as a preprocessing step

Proceedings of the 21st RCRA workshop on *Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion* (RCRA 2014).

Vienna, Austria, July 17–18, 2014.

before search and as a component of hybrid backtrack-consistency algorithms such as MNSAC.

This paper describes two kinds of extension of the neighbourhood SAC algorithm. One is a simple extension from the binary to the non-binary case. This requires some discussion since there are some subtleties that have to be attended to. The other is a more far-reaching generalisation, from neighbourhood SAC to k -neighbourhood SAC, where k is the longest path from the focal variable to any variable in the subgraph. Within this framework, neighbourhood SAC becomes 1-neighbourhood SAC. Because this is in some ways the more interesting extension, it will be discussed first.

In addition, the present paper examines hybrid backtracking algorithms based on varying degrees of singleton arc consistency (SAC). Two such algorithms are considered, one based on full SAC, the other based on NSAC. (The latter was considered briefly in [7].) Like the maintained arc consistency algorithm (MAC), these algorithms establish a specific level of consistency in a constraint satisfaction problem, and then re-establish it after each decision made in the course of backtrack search. Since both forms of singleton arc consistency are stronger than simple arc consistency (AC), their use during search serves to reduce the search space to a greater extent than with arc consistency alone. In what follows, the algorithm that maintains full SAC will be referred to as MSAC, while the one that maintains NSAC will be referred to as MNSAC.

The next section gives general background concepts and definitions, including a discussion of NSAC and an algorithm for establishing it. Section 3 gives an account of k -NSAC. Section 4 describes extensions of these ideas to non-binary constraints. Section 5 describes the basic MNSAC and MSAC algorithms including methods for tuning it for better performance. Section 6 describes the problems used in experiments on search. Sections 7-8 gives the results of performance comparisons, using several classes of CSPs. Section 9 describes some variants of MNSAC that perform more restricted SAC testing. Section 10 gives conclusions. Because of space constraints, proofs of propositions are omitted.

2 Background Concepts

A constraint satisfaction problem (CSP) is defined in the usual way, as a tuple, (X, D, C) where X are variables, D are domains such that D_i is associated with X_i , and C are constraints. A *solution* to a CSP is an assignment or mapping from variables to values that includes all variables and does not violate any constraint in C .

In problems with binary constraints, arc consistency (AC) refers to the property that for every value a in the domain of variable X_i and for every constraint C_{ij} with X_i in its scope, there is at least one value b in the domain of X_j such that (a,b) satisfies that constraint. For non-binary, or n -ary, constraints generalized arc consistency (GAC) refers to the property that for every value a in the domain of variable X_i and for every constraint C_j with X_i in its scope, there is a valid tuple that includes a .

Singleton arc consistency, or SAC, is a form of AC in which the just-mentioned value a , for example, is considered the sole representative of the domain of X_i . If AC can be established for the problem under this condition, then it may be possible to find a solution containing this value. On the other hand, if AC cannot be established then there

can be no such solution, since AC is a necessary condition for there to be a solution, and a can be discarded. If this condition can be established for all values in problem P , then the problem is singleton arc consistent. (Obviously, SAC implies AC, but not vice versa.)

Neighbourhood SAC establishes SAC with respect to the neighbourhood of the variable whose domain is a singleton.

Definition 1. The *neighbourhood* of a variable X_i is the set $X_N \subseteq X$ of all variables in all constraints whose scope includes X_i , excluding X_i itself. Variables belonging to X_N are called the neighbours of X_i .

Definition 2. A problem P is neighbourhood singleton arc consistent with respect to value v in the domain of X_i , if when D_i (the domain of X_i) is restricted to v , the problem $P_N = (X_N \cup \{X_i\}, C_N)$ is arc consistent, where X_N is the neighbourhood of X_i and C_N is the set of all constraints whose scope is a subset of $X_N \cup \{X_i\}$.

In this definition, note that C_N includes constraints among variables other than X_i , provided that these do not include variables outside the neighbourhood of X_i .

Definition 3. A problem P is neighbourhood singleton arc consistent (NSAC) if each value in each of its domains is neighbourhood singleton arc consistent.

In this paper we use NSACQ to ensure neighbourhood SAC; in previous work, this was found to be the most efficient NSAC algorithm across a variety of problem classes [7]. NSACQ uses an AC-3 style of processing at the top-level instead of the AC-1 style procedure that is often used with SAC algorithms. This means that there is a list (a queue) of variables, whose domains are considered in turn; in addition, if there is a SAC-based deletion of a value from the domain of X_i , then any neighbours of X_i that are not currently on the queue are put back on. Unlike most SAC (or NSAC) algorithms, there is no “AC phase” following a SAC-based value removal. The idea is that if a deletion from the domain of focal variable X_i has any effect, it must affect the neighbours of X_i , and any effects elsewhere in the problem can only occur through effects on these neighbours.

In practice, NSAC is preceded by a step in which arc consistency is established (shown in Fig. 1), although this is not required to establish neighbourhood arc consistency. This is done to rapidly rule out problems in which AC is sufficient to prove unsatisfiability. It also eliminates values which are easily detected using a less expensive arc consistency algorithm.

3 k -Neighbourhood SAC

The idea behind k -neighbourhood SAC is very simple: instead of restricting the variables in the subgraph to be made singleton consistent to neighbours of the focal variable, one extends the subgraph to include all those variables that are connected by a path of length k or less to the focal variable.

There are several reasons for investigating these forms of consistency:

- This form of extension continues to take advantage of the “neighbourhood character” of AC.

```

Procedure NSACQ
  Q  $\leftarrow$  X
  OK  $\leftarrow$  AC(P)
  While OK and not empty-Q
    Select and remove  $X_i$  from Q
    Changed  $\leftarrow$  false
    Foreach  $v_j \in \text{dom}(X_i)$ 
       $\text{dom}'(X_i) \leftarrow \{v_j\}$ 
      If AC( $X_i + \text{neighbours}(X_i)$ ) leads to wipeout
        Changed  $\leftarrow$  true
         $\text{dom}(X_i) \leftarrow \text{dom}(X_i) / v_j$ 
        If  $\text{dom}(X_i) == \emptyset$ 
          OK  $\leftarrow$  false
    If Changed == true
      Update Q to include all neighbours of  $X_i$ 

```

Fig. 1 Pseudocode for NSACQ.

- Evaluating k -neighbourhood SAC algorithms for different values of k may allow us to discover forms of singleton arc consistency that are best able to handle the tradeoff between effort and effectiveness for a given class of problems, where effort is the time taken to achieve the given level of consistency and effectiveness is the number of values discarded and the proportion of unsatisfiable problems proven unsatisfiable during preprocessing.
- Following from the last-mentioned point, with a series of consistencies of this sort, there may be opportunities to tune the level of consistency, i.e. to learn the best level of consistency for a given class of problems or for a given problem instance or for a given state of search.

In the present work we will only consider forms of k -neighbourhood SAC where $1 \leq k \leq 3$.

Proposition 1. For any value of k , $k+1$ -NSAC dominates k -NSAC, unless both subgraphs include all the variables in the problem. That is (except for the last-mentioned condition), if a value is removed by k -NSAC then it will also be removed by $k+1$ -NSAC, but the converse does not hold.

The algorithms used in this paper to establish each level of consistency are essentially the same as the basic NSACQ algorithm. The only difference is that the variables that make up $\text{neighbour}(X_i)$ in Fig. 1 include all those in the k -neighbourhood of X_i . Algorithms were implemented in Common Lisp, and experiments were run in the XLispstat environment with a Unix OS on a Dell Poweredge 4600 machine (1.8 GHz).

Initial tests were made with random problems having the same parameters as those in [5], with the additional constraint that the constraint graphs were connected. These problems have $\langle n, d, p_1, p_2 \rangle$ parameters $\langle 100, 20, 0.05, t \rangle$, where t (tightness) varies from 0.10 to 0.90 in steps of 0.05. There were 50 problems for each value of t .

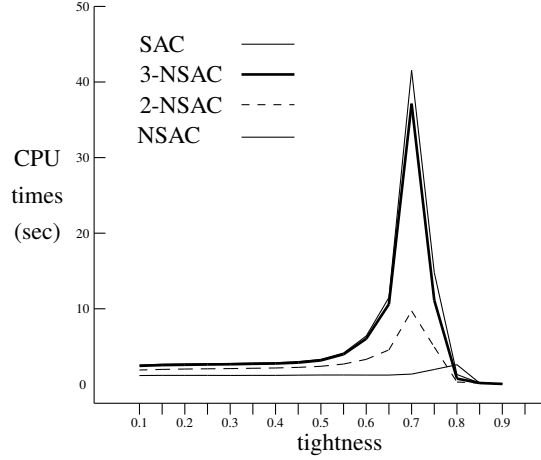


Fig. 2. Runtimes for full SAC and k -NSAC algorithms on random problems. (Highest curve is for SAC, lowest for NSAC.)

Fig. 2 shows runtimes across the spectrum of tightnesses; each curve is for a different algorithm producing a different level of consistency. In addition, results for a full SAC algorithm are shown, using SACQ, an efficient SAC algorithm that is similar to NSACQ except that instead of AC being applied to a k -neighbourhood, it extends to the entire problem. Table 1 shows the relative effectiveness of these algorithms across the part of the tightness spectrum where problems have no solutions. Here, the measure of effectiveness is the number of problems proven unsatisfiable out of a total of 50.

**Table 1. Random Problems Proved Unsatisfiable by
by Preprocessing Algorithms**

algorithm	problem tightness				
	0.70	0.75	0.80	0.85	0.90
SAC	2	50	50	50	50
3-NSAC	2	50	50	50	50
2-NSAC	0	50	50	50	50
NSAC	0	1	43	50	50
AC	0	0	0	50	50

Notes. Number unsatisfiable in each case is 50. (There were no unsatisfiable problems with tightness < 0.70 .)

These results show that for this kind of problem, 2-NSACQ handles the tradeoff between efficiency and effectiveness very well. Its runtimes are close to those of NSACQ, even for the hardest problems, while its effectiveness is almost as good as full SAC. In addition, 3-NSACQ is just as effective as the full SAC algorithm although it is somewhat faster.

In addition to random problems, these four algorithms were tested on various kinds of structured binary problems. These included two types of benchmark problem, ob-

tained from the site maintained at Université Artois ¹: the rlfap-graph problems with solutions and blackhole-4-7 problems. In addition, randomly generated problems with either intensional constraints based on relational operators (“relop” problems) or absolute distance constraints of the form $|X_i - X_j| > k$ (here called “distance” problems) were used. The former had 100 variables, domain size 20, graph density 0.26, with $\geq$ and $\neq$ constraints in equal proportions. The latter had 50 variables, domain size 20, graph density 0.23, while k had an expected value of 3 and deviations from this occurred with diminishing probability in either direction. (The graph density was chosen because this is within a critical complexity region.) All relop problems had solutions, while the distance problems included satisfiable and unsatisfiable instances.

Table 2. Results for Different Forms of NSAC with Various Structured Problems

problems	#	AC		1-NSAC		2-NSAC		3-NSAC		SAC	
		t	del	t	del	t	del	t	del	t	del
rlfap-graph	4	1	279	422	778	966	988	2519	1021	4053	1038
relop-geneq	100	0.1	0	42	556	422	758	334	759	432	761
black hole	20	1	280	27	280	33	280	43	280	44	280
absol dist	100	0.2	8.2	1.4	13.9	8.3	17.8	8.4	18.0	8.7	18.0

Notes. # is number of problems in each set. t is mean time (sec). del is number of values deleted per problem. 3-NSAC time for relop problems has been verified.

These results show that in most cases any level of singleton arc consistency deletes considerably more values than AC, although the time required is always much greater. Secondly, as with random problems there is a diminishing rate of return for enforcing higher levels of singleton arc consistency. Usually, there is a clear increase in effectiveness on going from 1-NSAC to 2-NSAC. Beyond this, further improvements are smaller, either in absolute value or relative amount. However, the difference between 1-NSACQ and 2-NSACQ with respect to efficiency (mean runtime) is also usually marked. Thus, unless there is a striking payoff in reduced search effort, in these cases the best choice would be 1-NSAC. Moreover, in no case is there a clear benefit from using full SAC as opposed to some form of NSAC.

4 Extension to Non-Binary Constraints

Neighbourhood SAC can be extended to n -ary constraints in a way that is straightforward to deal with in practice. As far as the author is aware, this is not as easily done with other forms of higher-order consistency such as maxRPC [1].

However, there are subtleties in defining NSAC under these conditions that must be attended to. In particular, constraint reduction must be restricted to the domains of neighbouring variables, and this means that there can be non-binary constraints which contain non-neighbouring variables whose domains should *not* be reduced during a bout of neighbourhood consistency checking. In other words, for non-binary constraints we must revise Definition 2 slightly because $X_N \cup X_i$ no longer includes all the variables

¹ <http://www.cril.univ-artois.fr/lecoutre/benchmarks.html>

in the union of the scopes of C_N . This entails a single change in the original definition, which is indicated by italics.

Definition 2'. For problems with non-binary constraints, a problem P is neighbourhood singleton arc consistent with respect to value v in the domain of X_i , if when D_i (the domain of X_i) is restricted to v , the problem $P_N = (X_N \cup \{X_i\}, C_N)$ is arc consistent, where X_N is the neighbourhood of X_i and C_N is the set of all constraints whose scope includes at least two members of the set $X_N \cup \{X_i\}$.

It is important to note that under this definition, any non-binary constraint that includes X_i will also include *all* of the other variables in that constraint, since by definition all are in the neighbourhood of X_i . It is only when non-binary constraints include two or more neighbours of X_i that there is a difference from the binary case. This is shown in Fig. 3.

From this figure it should be clear that the present definition of the neighbourhood subgraph is the proper extension of the binary case. In the binary case, possible reductions in the domain of a non-neighbouring variable adjacent to one in the neighbourhood are ignored during neighbourhood SAC processing. Similarly, in the n -ary case we do not reduce domains of adjacent variables with respect to a non-binary constraint when these variables are not in the neighbourhood of the focal variable. The difference is that in the binary case we can ignore the constraint in question altogether, while in the non-binary case we must consider the projection of the constraint onto the set of neighbouring variables.

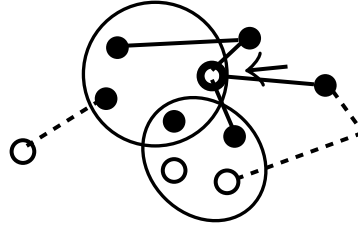


Fig. 3. A neighbourhood subgraph with binary and non-binary constraints. The former are shown as lines between the small circles, the latter as a large circle and ellipse. Arrow indicates focal variable. Filled circles are neighbouring variables; small outline circles are non-neighbours whose domains do not change during AC testing. Two constraints excluded from the neighbourhood subgraph are shown as dashed lines.

Proposition 2. For problems with non-binary constraints, a well-defined level of consistency in the form of domain reductions, i.e. a fixpoint can be achieved for any form of k -neighbourhood SAC.

The NSACQ algorithm can be easily extended to handle non-binary constraints. The only change is that a GAC algorithm must be used instead of a binary AC algorithm, and non-binary constraints must be projected onto neighbouring variables. In the code, which uses an STR-like procedure [6], this was accomplished by a single extra check. The next proposition shows that it achieves a well-defined level of consistency.

Proposition 3. The NSACQ algorithm achieves the fixpoint specified in Proposition 2.

Preliminary tests were made with GAC-based versions of the SAC and k -NSAC algorithms. These were done with configuration problems used in previous work [4]. The constraints are table constraints with 20 or 40 variables and constraint arities ranging from two to five. In addition to the original problem, esvs-config, variations were constructed by adding constraints (esvs-config.h1 and .h2), tightening some constraints (esvs-cfghard) and doubling the problem whilst varying one binary constraint, which reduced the number of solutions drastically (esvs-cfgdouble).

Again, we see clear differences in effectiveness for the harder problems, especially between AC and 1-NSAC. 2-NSAC deletes a few more values than NSAC, but with very little difference in time, while higher consistencies offer no further benefit.

**Table 3. Results for Different Forms of NSAC
with Problems with N -ary Constraints**

problem	AC		1-NSAC		2-NSAC		3-NSAC		SAC	
	t	del	t	del	t	del	t	del	t	del
esvs-config	0.00	0	0.04	0	0.05	0	0.08	0	0.07	0
esvs-config.h1	0.00	0	0.05	4	0.06	4	0.09	4	0.09	4
esvs-config.h2	0.00	1	0.03	11	0.04	11	0.05	11	0.04	11
esvs-cfghard	0.01	3	0.05	60	0.07	72	0.08	72	0.08	72
esvs-cfgdouble	0.01	16	0.09	126	0.11	144	0.14	144	0.15	144

Notes. t is time (sec). del is number of values deleted per problem.

5 Maintained SAC Algorithms

5.1 Maintained neighbourhood SAC

Maintained neighbourhood SAC (MNSAC) uses the same basic strategy as the MAC algorithms. In an initial pre-search phase, maintained neighbourhood SAC is established. Then after each decision, i.e. after each assignment (and, in the case of binary branching, after each value exclusion) this level of consistency is reestablished for the entire problem.

Search with MNSAC (or MSAC) can be made more efficient thanks to the following property.

Proposition 4. Given previous processing that established neighbourhood SAC in the problem, if an AC step following a (single) new assignment does not delete any values, then neither will NSAC.

In addition to the basic MNSAC algorithm, two variants were tested. The first used a one-pass version of NSAC during search (full NSAC was done during the preprocessing phase). Because this version of NSAC may fail to filter values that are no longer supported, in this version we cannot immediately instantiate a variable with a singleton domain, as we can with MAC or MNSAC. In the second variant, a type of interleaving strategy is used in which a counter records the number of values deleted by AC processing. When the count reaches a certain number (set beforehand), NSAC is carried out at the same search node, and afterwards the counter is reset to zero. In this way the amount of NSAC processing during search is reduced.

5.2 Maintained SAC

The maintained SAC algorithm used in this work is based on SACQ, which corresponds to NSACQ except that after a SAC-based deletion all variables are put back on the queue (see [7] for further details). It is, therefore, similar to SAC-1 [3] in that it resets itself after domain reductions, but it does not go through the entire set of variables and domain values before doing so. In earlier work, SACQ was shown to be more efficient than SAC-1 and was often as efficient as more advanced SAC algorithms [7].

6 Methods for Testing Search Algorithms

Tests of the different maintained consistency algorithms were made with a variety of test problems including both random CSPs and problems with structural features of various types. To date, problems have been restricted to those with binary constraints. Implementation details were the same as described in previous sections.

6.1 CSPs without definite structure

These were either homogeneous random problems or geometric problems with variable support, as described in [7].

6.2 Structured CSPs

These CSPs had constraints that were related to the magnitudes of the domain values. Problems of various types have been tested, two of which are reported here. The first type were random relop problems, where constraints are relational operators, and the constraint graph is random. In all cases some of the constraints were inequality constraints, which gives intractable problems.

Problems of the second type were generated by taking benchmark RLFAPs and altered them by either decrementing or incrementing the values of k in the constraint specifications. Here, the intent was to start with benchmark problems, which as CSPs are easy to solve, and generate problems with the same structure but with increasing degrees of difficulty. For this purpose three scen06-sub problems (sub0, 2 and 4) were used. Since these problems are all unsatisfiable, k was decremented systematically until problems with solutions were produced. Only the $>$ constraints were altered in this fashion, and for these experiments all such constraints were altered.

In addition, a method was used which allowed the decrement for a given constraint to be calculated according to a pseudo-probability distribution: beginning with a fixed initial value, greater decrements could be obtained but with successively smaller probabilities. This was done by beginning with an initial value for the decrement and incrementing it by one as long as a pseudo-random generator generating values between 0 and 1 returned a value > 0.5 .

7 Results with Random Problems

7.1 Homogeneous random problems

The results shown in Table 4, for the pure algorithms, are typical for this type of problem. While MNSAC and MSAC reduce the size of the search tree (in the latter case quite dramatically), their overall efficiency is less than that of MAC. Results are shown for two variable ordering heuristics, minimum domain over forward degree and minimum domain over (forward) weighted degree [2].

Table 4. Search Results for Homogeneous Random Problems

algorithm	dom/fwd		dom/wtdg	
	nodes	time	nodes	time
MAC	1621	5	1538	5
MAC/init NSAC	1621	5	1540	5
MAC/init SAC	1621	5	1540	6
MNSAC	751	38	775	37
MNSAC/init SAC	751	40	775	44
MSAC	83	51	83	56

Notes. $\langle 50, 10, 0.18, 0.369 \rangle$ problems. Sample size 100.

Mean search nodes and runtimes (sec).

When the weighted degree heuristic is used with MNSAC (or MSAC), there is a question of which failures should be included in the weights, since for this implementation at least, constraint failures could be due either to AC alone or to NSAC (or SAC) processing. For the results in the table, constraint weights were based on all failures. For comparison, if weights used during MNSAC are based on AC alone, the mean for search nodes is 785, while if they are based on NSAC alone, the mean is 773. (As one would expect, run times were similar in all cases.) Similar results were found with sparse 150-variable problems (here with both sources of information, the mean was 1254, with AC only it was 1282, with NSAC only it was 1344) and with 50-variable problems with the same parameters as those in Table 4 but with no solutions (here means were 398, 403 and 397, respectively). Similar findings were obtained with MSAC. Hence in this and subsequent tests with dom/wtdg, all failures were included in the counts.

7.2 Random problems with heterogeneous features

Results for geometric problems with varying support are shown in Table 5. These are for the dom/fwd heuristic only; with weighted degree these problems become very easy, so that extra processing does not improve performance. It is possible that there are patterns of support for which effects can be shown, but this line of inquiry has not yet been pursued.

**Table 5. Search Results for Random Geometric/
Varying-Support Problems**

algorithm	search nodes	time	# solved
MAC	>70,105	668	292
MAC/init NSAC	>30,610	167	295
MAC/init SAC	>10,801	83	297
MNSACQ	>10,078	?	297
MNSACQ/init SAC	72	78	300
MSAC	69	171	300

Notes. 120-variable problems. Other parameters in text. dom/fwd-degree heuristic. 1-million node cutoff. Sample size 300. Mean search nodes and runtimes (sec). “?” is due to three problems where cutoff could not be reached.

These results show, first, that there are problems for which the extra processing required by MNSAC or MSAC can pay off both in number of nodes searched and overall runtime, i.e. there are problems for which the search-tree-size/runtime tradeoff can be finessed. Moreover, it is of interest that such results are found when some form of structure or non-randomness is introduced into the problem. At the same time, the results with various forms of MNSAC show that it is sometimes difficult to judge beforehand how much processing is required for substantial and reliable improvement.

8 Results with Structured Problems

8.1 Relop problems

The first set of problems of this type had 100 variables, with domain size equal to 20; graph density was 0.25. Search results for the different algorithms are shown in Table 6. It can be seen that for the dom/fwd heuristic, these problems are quite difficult for MAC, and in this case using NSAC or SAC reduces the size of the search tree by three orders of magnitude. In this case runtimes also improve by an order of magnitude. It can also be seen that NSAC, used either as a preprocessing algorithm or as part of a maintained consistency algorithm, gives the best runtimes.

At the same time, with a more effective variable ordering heuristic, the performance of MAC is again improved to the extent that it becomes the most efficient algorithm.

**Table 6. Search Results for Random Relop
Problems**

algorithm	dom/fwd		dom/wtdg	
	nodes	time	nodes	time
MAC	676,442	5948	656	11
MAC/init NSAC	6637	103	262	50
MAC/init SAC	732	698	175	464
MNSAC	265	129	106	198
MNSAC/init SAC	143	731	116	512
MSAC	101	503	100	1041

Notes. 100-variable relop problems based on $\geq$ and $\neq$ constraints. Sample size 100. Mean search nodes and runtimes (sec).

With larger problems of this type search with different forms of singleton arc consistency can result in marked improvements in performance even when using the dom/wtdg heuristic. In this experiment, 150-variable problems were tested, again with a 50:50 mix of $\neq$ and $\geq$ constraints in which constraints and constraint-type were selected randomly. The domain size was always 20. Nine different densities were used, ranging from 0.1 to 0.5 in steps of 0.05. This spans a range that includes relatively easy problems with solutions through the critical complexity region and far enough into the unsatisfiable region that problems become easier to run to completion. The sample size at each density was 50.

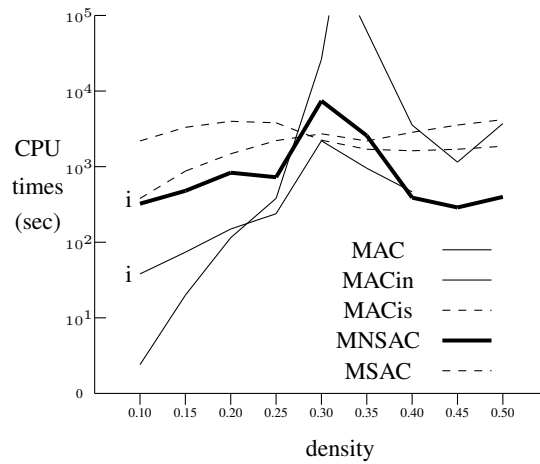


Fig. 4. Mean runtimes for 150-variable relop problems. Log scale on ordinate. MACix refers to MAC with initial NSAC or SAC; the curves are marked with an "i" on the left. Sample size 50.

The results are shown in Fig. 4. For densities 0.1-0.25 all problems had solutions, and here MAC was the most efficient. But for the next density (41/50 problems had solutions), MAC was appreciably less efficient than either of the other two, and for density = 0.35 (no problems had solutions) MAC could not be run to completion even after several days. For the very hardest problems (densities 0.3 and 0.35) MSAC was in fact the most efficient; however, outside this range it was much less efficient than the other two algorithms.

Table 7. Mean Search Nodes for 150-Variable Relop Problems

algorithm	density					
	0.10	0.20	0.30	0.35	0.40	0.50
MAC	236	3460	798293	$> 10^8$	138115	37813
MNSAC	159	217	3071	404	4	0
MSAC	150	150	134	2	0	0
MACiNSAC	179	1473	67388	13594	234	0
MACiSAC	163	505	5575	591	0	0

Notes. Mean search nodes for separate sets of 50 problems.

Table 7 shows that as problems of this type become more difficult, there are drastic differences in search tree size for MAC as opposed to SAC-based algorithms. In fact, at density 0.35 the differences are at least six or seven orders of magnitude (since the value 10^8 is a conservative lower bound). It also shows that most unsatisfiable problems could be proven unsatisfiable during preprocessing when either SAC or NSAC was used.

However, it was found that by using NSAC for preprocessing search could be reduced enough to make this combination competitive with MNSAC or MSAC. Preprocessing with SAC was also tested, but while it led to a greater reduction in search nodes (Table 7), overall runtime increased by a factor of two to ten depending on the problem class (Fig. 4).

8.2 Radio frequency problems

Here, we present results from problems based on the sub4 problem; results for problems based on sub0 and 2 were similar, although since these problems were easier to solve, the differences were less dramatic. Speaking very generally, the results followed a similar pattern as with other problem classes: when the dom/fwd heuristic was used, SAC-based maintained consistency led to orders-of-magnitude improvements; however, when dom/wtdg was used the results were either comparable or, as problems entered the phase transition, MAC was the most efficient.

The results just described refer to unsatisfiable problems, shown on the left side of Fig. 5. For problems with solutions, MAC was always much faster than the other algorithms, and in these cases this was true for either variable ordering heuristic. This was because in the critical complexity region (represented here by decrements in values of k beginning at 42) problems either had no solution or had hundreds of thousands or millions of solutions.

9 Approximations to Fully Maintained NSAC Algorithms

Tests were made with the one-pass and interleaved versions of MNSAC, using the 150-variable relop problems and densities between 0.20 and 0.35 inclusive. For the interleaved strategy, threshold values of 50 and 200 deletions were tested. Results are shown in Table 8. (Only ACct=200 is included since the other didn't finish in time for submission; however, it is clearly less efficient.)

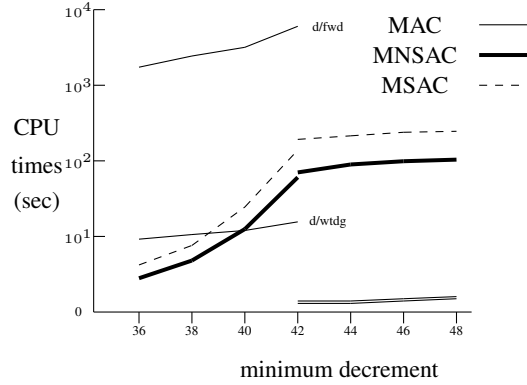


Fig. 5. Mean runtimes for perturbed RLFAP problems. Log scale on ordinate. Problems on right-hand side (decrements 42 to 48) had solutions; those on the left (36 to 42) did not; hence the break in each curve. Sample size 50.

The interleaving strategy used gave better results than the one-pass strategy, and times were always much lower than the full MNSAC. For three densities they were better than MAC alone; however, MAC with initial NSAC always gave better mean runtimes. In addition, for the hardest problem sets there was a definite fall-off in search space reduction.

Table 8. Results for Approximation Algorithms

density	MAC		MNSAC		MACiNSAC		MNSAC-1pass		MNSAC-AC200	
	t	nodes	t	nodes	t	nodes	t	nodes	t	nodes
0.20	115	3460	839	217	150	1473	351	195	199	291
0.25	378	10140	731	168	238	3021	348	167	257	188
0.30	26055	798293	7467	3071	2198	67388	4640	5243	3199	34859
0.35	—	—	2555	404	953	13594	1371	752	1248	7106

Notes. 150-variable relaps. t is time (sec). nodes is mean search nodes.

10 Conclusions

Together, these results show that while MAC is a very powerful general-purpose algorithm, there are cases in which it is insufficient. At the same time, these cases can sometimes be solved without inordinate expenditure when more stringent forms of consistency are employed either in the preprocessing stage and/or during search. Because SAC-based methods are based on arc consistency, they are relatively straightforward to implement (they ride on top of a basic AC solver) whilst being much more powerful with respect to values discarded. They are, therefore, potential competitors to other forms of enhanced consistency that are currently receiving considerable attention, such as max restricted path consistency (maxRPC) [1].

At present it is not possible to predict which type of consistency maintenance can best handle the basic tradeoff between search reduction and overall processing time. In

many cases, simply using SAC-based methods for preprocessing and then using MAC is the most efficient strategy.

A noteworthy finding was that the variable ordering heuristic dom/wtdg was often able to compensate for MAC's deficiencies in deleting values in comparison with the SAC-based methods. In many cases this occurred to such a degree that MAC, while being orders of magnitude less efficient than the other methods when used in combination with a simpler, non-adaptive heuristic, again became the most efficient method. This shows that in discussing the benefits of lesser or greater degrees of consistency maintenance one must take account of the varying effectiveness of consistency strategies in relation to the current state of the problem. Nonetheless, use of this heuristic did not always result in adequate performance (cf. hard relop problems), so that significant benefits still accrued when NSAC was used for preprocessing.

Because of the basic tradeoff between search space reduction and overall efficiency and because of the large number of possible strategies, SAC-based search is a promising area of application for adaptive algorithms, that is, algorithms that learn the best type or mix of procedures for a given problem class or even a given problem. Exploration of such possibilities will be an important part of future work in this area.

References

1. T. Balafoutis, A. Paparrizou, K. Stergiou, and T. Walsh. New algorithms for max restricted path consistency. *Constraints*, 16:372–406, 2011.
2. 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.
3. R. Debruyne and C. Bessière. Some practicable filtering techniques for the constraint satisfaction problem. In *Proc. 15th Internat. Joint Conf. on Artif. Intell. – IJCAI'97. Vol. 1*, pages 412–417. Morgan Kaufmann, 1997.
4. E. C. Freuder, R. Heffernan, N. Wilson, and R. J. Wallace. Lexicographically-ordered constraint satisfaction problems. *Constraints*, 15:1–28, 2010.
5. C. Lecoutre and S. Cardon. A greedy approach to establish singleton arc consistency. In *Proc. 19th Internat. Joint Conf. on Artif. Intell. – IJCAI'05*, pages 199–204. Professional Book Center, 2005.
6. J. R. Ullmann. Partition search for non-binary constraint satisfaction. *Information Sciences*, 177:3639–3678, 2007.
7. R. J. Wallace. SAC and neighbourhood SAC. *AI Communications*, 27:to appear, 2014.