Proceedings of the Twenty-Ninth International
Florida Artificial Intelligence Research Society Conference

Neighbourhood SAC for Constraint Satisfaction Problems with Non-Binary Constraints

Richard J. Wallace
Insight Centre for Data Analytics
Department of Computer Science, University College Cork, Cork, Ireland
richard.wallace@insight-centre.org

Abstract

Neighbourhood singleton arc consistency (NSAC) is a type of singleton arc consistency (SAC) in which the subproblem formed by variables adjacent to a variable with a singleton domain is made arc consistent. In this paper we consider how to apply this form of consistency reasoning to problems with n-ary constraints including global constraints. The chief problem encountered is that of neighbouring variables contained in a constraint that also includes non-neighbouring variables. In this case, a strict extension of NSAC involves projection of such constraints onto the neighbourhood variables, but for many global constraints this may be difficult to do in practice. Here, we consider a simple variant called restricted neighbourhood SAC, that avoids this problem. We compare the two approaches on random and structured problems and show that in all cases the restricted form of k-NSAC is nearly as effective as the unrestricted form.

Introduction

Arc consistency (AC) is the best-known form of local consistency processing within the field of constraint solving. 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, although it is still based on AC. As such it is often more effective than AC, either in removing values or proving unsatisfiability; however, it is much more expensive.

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 original constraint graph. Although weaker than SAC, it is still a stronger form of consistency than AC (Wallace 2015). 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 (Wallace 2015). As a result, it has considerable potential as a specialized form of SAC-based consistency. Recently, this approach has been extended to k-neighbourhoods, i.e. to subgraphs that include the focal variable and all variables within a distance k of it (Wallace 2016).

In this work we consider how to extend these ideas to problems with n-ary constraints. This involves the question of how to deal with neighbourhood variables that are partly constrained with respect to the neighbourhood subgraph. This work continues some of the work in (Wallace 2016) and provides a better resolution of a key issue.

Background

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 non-binary 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, and a can be discarded. Neighbourhood SAC establishes SAC with respect to the neighbourhood of the variable whose domain is a singleton.

Definition 1. 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 $X_N \cup \{X_i\}$.

For k-neighbourhood SAC, one extends the subgraph to include all variables connected by a path of length k or less to the focal variable. (Note that in this framework NSAC becomes 1-NSAC.) As shown elsewhere, for any value of k, k+1-NSAC dominates k-NSAC (Wallace 2016).

In this work we use NSAC algorithms based on the SACQ style of search. Earlier work has shown that NSAC algorithms based on the SACQ strategy outperform those based on SAC-1 as well as more advanced SAC algorithms such as SAC-SDS (Bessière and Debruyne 2005) and SAC-3 (Lecoutre and Cardon 2005). (N)SACQ uses an AC-3 style procedure at the top-level instead of the AC-1 style 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 (N)SAC-based deletion of a value from the domain of X_i, then all relevant variables not currently on the queue are put back on. Unlike other (N)SAC algorithms, there is no "AC phase" following a SAC-based value removal. Pseudocode for the basic (1-)NSAC algorithm is shown in Fig. 1.

```
   Function NSACQ
1      Q ← $X$
2      OK ← AC(P)
3      While OK and not empty-Q
4          Select and remove $X_i$ from Q
5          Changed ← *false*
6          Foreach $v_j \in dom(X_i)$
7              $dom'(X_i) \leftarrow \{v_j\}$
8              If AC($X_i$+neighbours($X_i$)) → wipeout
9                  Changed ← *true*
10                 $dom(X_i) \leftarrow dom(X_i)/v_j$
11                 If $dom(X_i) == \emptyset$
12                     OK ← *false*
13         If Changed == *true*
14             Update Q with all neighbours of $X_i$
15     Return OK
```

Figure 1: Pseudocode for NSACQ.

Extension to Non-Binary Constraints

Neighbourhood SAC can be extended to n-ary constraints in a way that, in some cases at least, 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 (Balafoutis et al. 2011).

However, there are some difficulties in defining NSAC under these conditions that results in a form commensurate with the binary case. This suggests that an extension like this can be done in various ways. Here, I define one method that can be considered as standard, then I define a modified method that avoids some implementation difficulties entailed by the former.

To maintain the idea of a k-neighbourhood in the form already defined, domain reduction via constraints must be restricted to the domains of neighbouring variables. This implies 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. To cover such cases, we can revise Definition 1 slightly because $X_N \cup X_i$ no longer includes all the variables in the union of the scopes of C_N.

Definition $1'$**.** 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* $X_N \cup \{X_i\}$.

Note that any non-binary constraint that includes the focal variable X_i will also include *all* other variables in the constraint. Only when non-binary constraints include two or more neighbours of X_i without including X_i itself is there a difference from the binary case. This is shown in Fig. 2.

From this figure it should be clear that the present definition of the neighbourhood subgraph is a reasonable extension of the binary case. In both cases only domains of neighbouring variables are affected when NSAC is established. The difference is that in the binary case we can ignore constraints involving non-neighbour variables altogether. But in the non-binary case even if there are non-neighbourhood variables whose domains should not be reduced, there still may be neighbourhood variables that constrain each other. One way to deal with this is to project the constraint onto the set of neighbourhood variables.

Proposition 1. The NSACQ algorithm in its strict form, with extensions to handle projections of neighbourhood variables onto non-binary constraints, achieves neighbourhood singleton arc consistency as defined in Definition $1'$.

Proof Sketch. NSACQ is carried out as shown in Fig. 1. For non-binary constraints with some variables outside the neighbourhood, the proper level of consistency is achieved if, for every value a in the domain of a neighbourhood variable that is in the scope of this constraint, there is a constraint tuple whose projection onto the neighbourhood variables that belong to this constraint includes a. As in the binary case, any value in any variable outside the neighbourhood will not have become neighbourhood inconsistent by virtue of removal of a singleton value, since the neighbourhood of that variable is unchanged. □

This adjustment to ordinary arc consistency, which I will refer to as the strict form of ($k-$) NSAC, is fairly straightforward to implement for simple table constraints. In the code used here, which employs an STR procedure (Ullmann 2007), this is accomplished with an additional check that a given domain is in the k-neighbourhood subgraph. However, it may not be straightforward to add this to solvers that handle global constraints using specialized filtering algorithms. So we need to consider other possible forms of NSAC for non-binary constraints that do not have this problem.

One strategy is to include any constraint involving two or more neighbourhood variables. However, this approach presents conceptual difficulties, since variables that are not in the k-neighbourhood will sometimes be treated as if they were. Even if this results in a valid kind of consistency, this is not really a form of neighbourhood SAC. So this option is not considered here.

Another strategy is to omit these problematical cases when checking for neighbourhood SAC. Since consistency maintenance is restricted to the same set of neighbouring variables in any instance, this certainly qualifies as a form of ($k-$) NSAC. In fact, this variant conforms to the original Definition 1. Since some forms of constrainedness are being

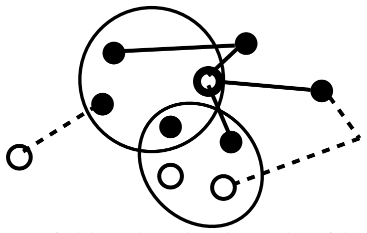

Figure 2: Neighbourhood subgraph with binary and non-binary constraints. Focal variable is circle with the thick outline, neighbouring variables are filled circles, non-neighbours outline circles. Lines between circles are binary constraints, large outline shapes non-binary. Dashed lines are constraints outside the neighbourhood subgraph.

ignored in this version of NSAC, it will be referred to it as restricted NSAC, or rNSAC for short.

It can be shown that the restricted form of NSACQ achieves NSAC as defined in the original Definition 1. In addition, the same dominance relations hold for the restricted versions of NSAC, i.e. any value removed by restricted k-NSAC will be removed by k+1-NSAC, but the converse does not necessarily hold.

Experimental tests

Tests were made with GAC-based versions of the SACQ and k-NSACQ algorithms, using homogeneous random and structured benchmark problems.

Random problems Random CSPs were generated by a program written by the author in which arities, their proportions, and their satisfiabilities could be specified independently, along with the mean degree of a variable. Only one experiment is presented here. Problems had constraints of arities 2, 3, 4 and 5 in the following proportions: 0.5, 0.3, 0.15 and 0.05. The number of variables was 100; the domain size was always eight; the mean degree was 3. For each problem, the same tightness value was used across all arities. Tightness was varied across problem sets to produce a series. Specific tightness values were 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.725, 0.75, 0.775, 0.8 and 0.9. For each tightness value 50 problems were tested. Problems with the tightest constraints had no solutions; for tightnesses < 0.7 all problems had solutions.

k-neighbourhood sizes were calculated for a sample of ten problems of each type. For each problem, all neighbourhood sizes were calculated. For k=1, mean size per problem was 6; for k=2 it was 27; for k=3 it was 68 (so all were less than n).

Fig. 3 shows preprocessing times for algorithms that establish different levels of neighbourhood SAC consistency, as well as full SAC. With each increase in neighbourhood size, there is a definite increase in runtime. (Of course, all these algorithms are much slower than AC, which always gave a mean $\leq$ 0.83 sec.) In addition, each restricted form of neighbourhood SAC is a little faster than its unrestricted counterpart. The pronounced asymmetry in the curves is an

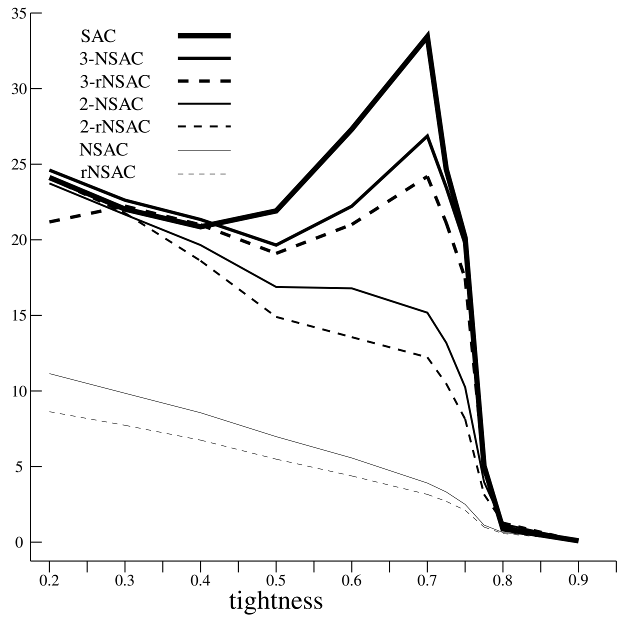

Figure 3: Runtimes (s) for full SACQ and k-NSACQ algorithms on random problems.

exaggeration of the asymmetry observed with binary problems and is due to the fact that here GAC is called repeatedly rather than AC. In addition, the looseness of the constraints means that STR is less efficient. [1]

Table 1 shows the number of problems proved unsatisfiable when each level of consistency was established. For these problems, there were cases where establishing a higher level of consistency proved more problems unsatisfiable up through 3-NSAC. In fact, for these problems 3-NSAC was sufficient to prove all unsatisfiable problems unsatisfiable. In addition, for each value of k, restricted NSAC was nearly as effective as the corresponding full NSAC algorithm.

At higher tightness levels, values deleted for problems with solutions also showed marked differences related to the level of consistency established. For example, for tightness 0.775, mean values deleted per problem for the 13 problems with solutions was 237 for AC, 251 for rNSAC, 253 for NSAC, 284 for 2-rNSAC, 330 for 2-NSAC, 358 for 3-rNSAC, 364 for 3-NSAC, and 369 for SAC. Individual problems showed marked differences across algorithms, ranging from little change between AC and SAC to almost doubling the number of deletions.

Structured problems The problems tested here were configuration problems used in previous work (Wallace 2016). Problems had 20 or 40 variables with constraints of arity 2-5. In addition to the original problem, esvs-config (n=20),

[1]These results differ in some ways from tests reported in (Wallace 2016) for the non-restricted forms, although the qualitative ordering is the same. In the present tests, in addition to performing all runs together in a block, care was taken to carry out only one run at a time, without using any other programs. The latter may be responsible for the differences found.

two variations were constructed by adding constraints and tightening some constraints (esvs-cfghard.1 and .2; n=20) and doubling the problem whilst varying one binary constraint, which reduced the number of solutions drastically (esvs-cfgdouble; n=40). For these problems, which have table constraints, the STR algorithm was used for filtering. Search was done with MAC-3 using the minimum domain variable ordering heuristic.

For these problems all neighbourhood subgraph sizes were well below n, the number of variables. For example, for the original configuration problem, means were 4.3, 5.9 and 7.5 for 1-, 2- and 3-neighbourhood subgraphs, respectively, and for the cfg-double problems the corresponding means were 5.2, 11.1 and 18.6.

Table 1. Random n-ary Problems Proved Unsatisfiable by Preprocessing Algorithms

	problem tightness		
algorithm	0.75	0.775	0.80
SAC	3	37	48
3-NSAC	3	37	48
3-rNSAC	3	37	48
2-NSAC	3	33	47
2-rNSAC	3	32	46
NSAC	1	28	41
rNSAC	1	28	39
AC	1	26	37
total unsat	3	37	48

The results of these tests are shown in Table 2 [2]. For harder problems there were differences in the effectiveness of preprocessing (i.e. number of values deleted) that also had clear-cut effects on search effort. For (1-)NSAC, the unrestricted form deleted more values than the restricted form, with some reduction in search effort, although overall the differences were not marked. 2-NSAC deleted a few more values than NSAC with little difference in time, and in this case the restricted form performed as effectively as the unrestricted form. Preprocessing to this level of consistency also eliminated nearly all branching during search even for the harder problems. Higher levels of consistency gave no further benefit.

Conclusions

This work complements earlier work (Wallace 2016) in showing how neighbourhood SAC can be extended to n-ary constraints in a fairly straightforward fashion. This greatly extends the scope of application of algorithms for establishing this form of consistency. The results and arguments suggest that SAC-based forms of consistency may be particularly useful with n-ary constraints in that they are able to detect inconsistencies and even problem unsatisfiability in cases where this is not possible with GAC.

Moreover, given that we now have an entire series of local consistencies with an obvious dominance ordering with respect to values deleted, we can choose among different levels of consistency using the same basic strategy (and algorithms) to balance the conflicting goals of effectiveness and efficiency.

[2] AC deletions differ from corresponding results for non-restricted NSAC in (Wallace 2016). The present values are correct.

Table 2. AC and k- NSAC for Configuration Problems

prob	t	del	sn	t	del	sn
		AC			rNSAC	
cfg	.00	0	20	.03	0	20
hd1	.00	24	362	.05	46	278
hd2	.01	24	1158	.03	46	806
db	.01	74	41	.07	109	41
		NSAC			2-rNSAC	
cfg	.04	0	20	.05	0	20
hd1	.05	60	254	.07	72	21
hd2	.04	55	326	.07	72	20
db	.08	126	41	.15	144	40

t=time (s) del=vals deleted sn=nodes.

We also show how to resolve the difficulty that arises in connection with subsets of neighbouring variables that fall within the scope of a constraint that also contains variables outside the neighbourhood. In this case the user has a choice of projecting the constraint onto the neighbourhood subset or ignoring the constraint altogether. This means that for certain global constraints and specialized implementations where it would be difficult to perform the necessary projections, neighbourhood SAC can still be added in its restricted form.

From the limited experiments we have done so far, it appears that in some cases the restricted form of NSAC performs as well as the unrestricted form, while in other cases there are differences that serve to impact subsequent search. Such differences seem to be more pronounced with more limited forms of NSAC, so if one carries out NSAC on k-neighbourhoods with $k > 1$, then the decision to restrict NSAC in this way is less likely to affect performance.

References

Balafoutis, T.; Paparrizou, A.; Stergiou, K.; and Walsh, T. 2011. New algorithms for max restricted path consistency. *Constraints* 16:372–406.

Bessière, C., and Debruyne, R. 2005. Optimal and suboptimal singleton arc consistency algorithms. In *Proc. 19th Internat. Joint Conf. on Artif. Intell. – IJCAI'05*, 54–59.

Lecoutre, C., and Cardon, S. 2005. A greedy approach to establish singleton arc consistency. In *Proc. 19th Internat. Joint Conf. on Artif. Intell. – IJCAI'05*, 199–204. Professional Book Center.

Ullmann, J. R. 2007. Partition search for non-binary constraint satisfaction. *Information Sciences* 177:3639–3678.

Wallace, R. J. 2015. SAC and neighbourhood SAC. *AI Communications* 28(2):345–364.

Wallace, R. J. 2016. Neighbourhood SAC: Extensions and new algorithms. *AI Communications* 29:(to appear).