Local Consistencies in SAT

Christian Bessiere, Emmanuel Hébrard, Toby Walsh

Local Consistencies in SAT

Christian Bessière[1], Emmanuel Hebrard[2], and Toby Walsh[2]

[1] LIRMM-CNRS,
161 rue Ada 34392 Montpellier, France
bessiere@lirmm.fr
[2] Cork Constraint Computation Centre,
University College Cork, Ireland
{e.hebrard, tw}@4c.ucc.ie

Abstract. We introduce some new mappings of constraint satisfaction problems into propositional satisfiability. These encodings generalize most of the existing encodings. Unit propagation on those encodings is the same as establishing **relational k-arc consistency** on the original problem. They can also be used to establish **(i,j)-consistency** on binary constraints. Experiments show that these encodings are an effective method for enforcing such consistencies, that can lead to a reduction in runtimes at the phase transition in most cases. Compared to the more traditional (direct) encoding, the search tree can be greatly pruned.

1 Introduction

Propositional Satisfiability (SAT) and Constraint Satisfaction Problems (CSPs) are two very typical NP-complete combinatorial problems. There has been considerable research in developing algorithms for both problems. Translation from one problem to the other can therefore profit from the algorithmic improvements obtained on either side. Enforcing a local consistency is one of the most important aspect of systematic search algorithms. In particular, arc consistency is often the best tradeoff between the amount of pruning and the cost of pruning. The AC encoding [Kas90] has the property that arc consistency in the original CSP is established by unit propagation in the encoding [Gen02]. A complete backtracking algorithm with unit propagation, such as DP [DLL62], therefore explores an equivalent search tree to a CSP algorithm that maintains arc consistency.

The rest of the paper is organized as follows. In section 2 we present the basic concepts used in the rest of the paper. In section 3 we introduce a family of encodings called the *k-AC encodings* where k is a parameter. These encodings enable a large family of consistencies, the so called *relational k-arc-consistency* [DvB95] to be established by unit propagation on the SAT encoding. They work with any arity of constraints. Section 4 focuses on binary networks, and show that these encodings can also be used to establish any *(i,j)-consistency* (another large family of consistencies [Fre85]). We also show that unit propagation on the k-AC encodings can achieve the given level of consistency in optimal time complexity in all cases. Section 5 introduces mixed encodings that combines previous ones

to perform a high level of filtering only where it is really needed. And finally, in section 6, we present some experiments, that assess the improvement of these encodings in comparison with the direct reformulation. The results also show the ability of this approach to solve large and hard problems by comparing it with the best algorithms for CSPs.

2 Background

2.1 Constraint satisfaction problem (CSP)

A *CSP* $\mathcal{P} = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ is a set $\mathcal{X} = \{X_1, \ldots, X_n\}$ of n *variables*, each taking a value from a finite *domain* $D(X_1), \ldots, D(X_n)$ elements of $\mathcal{D}$, and a set $\mathcal{C}$ of e *constraints*, d is the size of the largest domain. A *constraint* C_S, where $S = \{X_{i_1}, \ldots, X_{i_a}\} \subset \mathcal{X}$, is a subset of the cartesian product of the domains of the variables in S, $C_S \subset D(X_1) \times D(X_2) \times \ldots \times D(X_a)$ that denotes the compatible values for the variables in S. The incompatibles tuples are called *nogoods*. We are calling S, the *scope* of C_S and $|S| = a$ its *arity*. An *instantiation* I of a set T of variables is an element of the cartesian product of the domains of the variables in T. We denote $I[A]$ for the projection of I onto the set of variables A, and $C_S[A]$ the projection of the constraint C_S onto A. An instantiation I is *consistent* if and only if it satisfies all the constraints, that is, $\forall C_S \in \mathcal{C}$ such that $S \subseteq T, I[S] \in C_S$. A *solution* is a consistent instantiation over $\mathcal{X}$.
Let T and S be two distinct sets of variables $T, S \subset \mathcal{X}$, and I an instantiation of T which is consistent. A *support* J of I for S is an instantiation J of S such that $I \cup J$ is consistent. For an instantiation I, if there exists a set S such that I has no support for S, then I doesn't belong to any solution.

2.2 Direct encoding

The direct encoding [Wal00] is the most commonly used encoding of CSPs into SAT. There is one Boolean variable X_v for each value v of each CSP variable X. $X_v = T$ means the value v is assigned to the variable X. Those variables appear in three sets of clauses :
At-least-one clause : There is one such clause for each variable, and their meaning is that a value from its domain must be given to this variable.
let X be variable and $D(X) = \{v_1, v_2, \ldots, v_n\}$, then we add the *at-least-one* clause : $Xv_1 \vee Xv_2 \vee \ldots \vee Xv_n$.
At-most-one clause : There is one such clause for each pair of values for each variable, and their meaning is that this variable cannot get more than one value.
Let $v_i, v_j \in D(X), i \neq j$, then we add the *at-most-one* clause : $\neg Xv_i \vee \neg Xv_j$.
Conflict clause : There is one such clause for each nogood of each constraint, and their meaning is that this tuple of values is forbidden.
Let C_{XYZ} be a constraint on the variables X, Y, Z and $[u, v, w] \in D(X) \times D(Y) \times D(Z)$, an instantiation forbidden by C_{XYZ} ($[u, v, w] \notin C_{XYZ}$), then we add the *conflict* clause : $\neg Xu \vee \neg Yv \vee \neg Zw$.

2.3 AC encoding

The AC encoding [Gen02] enables a SAT procedure to maintain *arc-consistency* during search through *unit propagation*. It encodes not only the structure of the network, but also a consistency algorithm used to solve it. It differs from the direct encoding only on the conflict clauses which are replaced by *support* clauses, the others clauses remain unchanged.
Let X, Y be two variables, $v \in D(X)$ a value of X and $\{w_1, \ldots, w_k\}$ the supports of $X = v$ for Y, then we add the *support* clause : $\neg Xv \vee Yw_1 \vee Yw_2 \vee \ldots \vee Yw_k$. This clause is equivalent to $Xv \rightarrow (Yw_1 \vee Yw_2 \vee \ldots \vee Yw_k)$ which means : as long as Xv holds (i.e $Xv \neq False$, that is "the value v remains in X's domain"), then at least one of its support must hold. Therefore when all the supports of $X = v$ are falsified then v is itself falsified.

3 Generalisation of the AC encoding

The AC encoding can only be applied to binary networks, because support clauses encode the supports of *a single* variable for another *single* variable. Our goal is to encode any kind of support that follows from the definition in section 2.1. The new encoding we introduce here, *k-AC encoding*, allows this under the following restriction. The set of "supported" and "support" variables must be subsets of the scope of a constraint. In fact, this is not a strong restriction, because the union of this two sets can always be viewed as the scope of a constraint, i.e., the constraint which is the join of all those involved in these sets. The supports are conjunctions of values, they correspond to a conjunction of positive literals. Let $[v_1, \ldots, v_p]$ be a support of an instantiation, respectively for the variables $X_1, \ldots, X_p$. The conjunction that encode this support is $(X_1v_1 \wedge \ldots \wedge X_pv_p)$. To keep the encoding in clausal form, we need then to add an extra variable, say s, for this support and the following equivalence, $s \leftrightarrow (X_1v_1 \wedge \ldots \wedge X_pv_p)$ which result in the following *equivalence clauses* : $(\neg s \vee X_1v_1), \ldots, (\neg s \vee X_pv_p)$ and $(\neg X_1v_1 \vee \ldots \vee \neg X_pv_p \vee s)$.

Definition 1 (k-AC clause.). *Let $\mathcal{R} = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ be a constraint network, $C_S \in \mathcal{C}$ be a constraint on $S \subset \mathcal{X}$ such that $|S| = a$, $I = [v_{i_1}, \ldots, v_{i_k}] \in D(X_{i_1}) \times \ldots \times D(X_{i_k})$ an instantiation of k variables in S ($k \leq a$) and finally $s_1, \ldots, s_m$ its supports for the set of remaining variables in S : $\{X_{i_{(k+1)}}, \ldots, X_{i_a}\}$.*
I is represented by a conjunction : $X_{i_1}v_{i_1} \wedge \ldots \wedge X_{i_k}v_{i_k}$.
Therefore, $(X_{i_1}v_{i_1} \wedge \ldots \wedge X_{i_k}v_{i_k}) \rightarrow (s_1 \vee s_2 \ldots \vee s_m)$, (associated with the corresponding equivalence clauses) is the k-AC clause *representing the fact that if I is assigned to true, then at least one of its supports must also be true.*

In figure 1, we show the four possible k-AC encodings for a ternary constraint. Note that, in the particular cases where the set of support variables is a singleton or the empty set, in other words, $a-k = 1$ or $a-k = 0$, the conjunctions standing for the supports are unit and we do not need to add extra variables.
The k-AC clauses are a generalisation of support clauses in two ways:

X	Y	Z
a	a	b
a	b	b
b	a	a
b	a	b

$\Rightarrow_{k-AC}$

0-AC encoding	3-AC encoding
$T \rightarrow (S_1 \vee S_2 \vee S_3 \vee S_4) \wedge$ $(Xa \wedge Ya \wedge Zb) \leftrightarrow S_1 \wedge$ $(Xa \wedge Yb \wedge Zb) \leftrightarrow S_2 \wedge$ $(Xb \wedge Ya \wedge Za) \leftrightarrow S_3 \wedge$ $(Xb \wedge Ya \wedge Zb) \leftrightarrow S_4$	$((Xa \wedge Ya \wedge Za) \rightarrow F) \wedge$ $((Xa \wedge Yb \wedge Za) \rightarrow F) \wedge$ $((Xb \wedge Yb \wedge Za) \rightarrow F) \wedge$ $((Xb \wedge Yb \wedge Zb) \rightarrow F)$
2-AC encoding	**1-AC encoding**
$((Xa \wedge Ya) \rightarrow Zb) \wedge$ $((Xa \wedge Yb) \rightarrow Zb) \wedge$ $((Xb \wedge Ya) \rightarrow (Zb \vee Za)) \wedge$ $((Xb \wedge Yb) \rightarrow F) \wedge$ $((Xa \wedge Za) \rightarrow F) \wedge$ $((Xa \wedge Zb) \rightarrow (Ya \vee Yb)) \wedge$ $((Xb \wedge Za) \rightarrow Ya) \wedge$ $((Xb \wedge Zb) \rightarrow Ya) \wedge$ $((Ya \wedge Za) \rightarrow Xb) \wedge$ $((Ya \wedge Zb) \rightarrow (Xa \vee Xb)) \wedge$ $((Yb \wedge Za) \rightarrow F) \wedge$ $((Yb \wedge Zb) \rightarrow Xa)$	$(Xa \rightarrow (S_1 \vee S_2) \wedge$ $(Xb \rightarrow (S_3 \vee S_1) \wedge$ $(Ya \rightarrow (S_4 \vee S_5 \vee S_6)) \wedge$ $(Yb \rightarrow S_4) \wedge$ $(Za \rightarrow S_7) \wedge$ $(Zb \rightarrow (S_7 \vee S_8 \vee S_9)) \wedge$ $((Ya \wedge Zb) \leftrightarrow S_1) \wedge$ $((Yb \wedge Zb) \leftrightarrow S_2) \wedge$ $((Ya \wedge Za) \leftrightarrow S_3) \wedge$ $((Xa \wedge Zb) \leftrightarrow S_4) \wedge$ $((Xb \wedge Za) \leftrightarrow S_5) \wedge$ $((Xb \wedge Zb) \leftrightarrow S_6) \wedge$ $((Xb \wedge Ya) \leftrightarrow S_7) \wedge$ $((Xa \wedge Ya) \leftrightarrow S_8) \wedge$ $((Xa \wedge Yb) \leftrightarrow S_9)$

Table 1. A ternary constraint, first line : the variables involved in the constraint, the other lines give the allowed tuples. And four possible k-AC encoding of this constraint.

- They capture a larger family of consistencies, *relational k-arc-consistency* (section 3) and *(i, j)-consistency* (section 4).
- They work for any arity of constraints.

Note that support clauses are 1-AC clauses for binary constraints, and conflict clauses are a-AC clauses for constraints of arity a. For instance, let C_{XYZ} be a constraint on the variables X, Y and Z. If $I = \{X = u, Y = v, Z = w\}$ is an allowed tuple, then the corresponding 3-AC clause is $(Xu \wedge Yv \wedge Zw) \rightarrow True$ and is useless. If I is a nogood, then we have $(Xu \wedge Yv \wedge Zw) \rightarrow False$, which is a conflict clause $(\neg Xu \vee \neg Yv \vee \neg Zw)$. Direct and support encodings are then particular cases of k-AC encoding.

Unit propagation on the k-AC Clauses corresponds exactly to enforcing relational k-arc-consistency. Relational arc-consistency [DvB95] extends the concept of local consistency, which usually concerns variables, to constraints. A constraint is *relationally arc-consistent* if any instantiation which is allowed on a subset of its variables extends to a consistent instantiation on the whole. *Relational k-arc-consistency* is the restriction of the definition above to sets of variables of cardinality k.

Definition 2 (Relational k-arc-consistency.). *Let $\mathcal{R} = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ be a constraint network, C_S a constraint over the set of variables $S \subset \mathcal{X}$. C_S is* relationally k-arc-consistent *iff $\forall A \subset S$ such that $|A| = k$ and $\forall I$ a consistent instantiation on A, I can be extented to a consistent instantiation on S in relation to C_S. This means : if $C_S[A]$ is the projection of the relation C_S on A and I is consistent on A, therefore $I \in C_S[A]$.*
A constraint network is relationally k-arc-consistent iff all its constraints are relationally k-arc-consistent.

A k-AC clause is an implication which premiss is a conjunction that stands for the k-instantiation I, and conclusion is a disjunction of supports $s_1 \vee s_2 \vee \ldots \vee s_m$. The k-AC clause for I is $\mathcal{H} = I \rightarrow s_1 \vee s_2 \vee \ldots \vee s_m$. Relational k-arc-consistency ensures that each consistent instantiation of k variables of a constraint can be extented to all the variables of that constraint. In other words, if an instantiation doesn't satisfy this assertion, the constraint forbids that this tuple is added. In the case of the k-AC clauses, when all the supports (which are linked to the conjunction of assignments they represent by equivalence clauses), are falsified, then the premiss must be falsified and this is exactly the nogood corresponding to the k-instantiation, $\mathcal{H} = \neg I$.

Theorem 1 *Performing full unit propagation on at-least-one, at-most-one and k-AC clauses is equivalent to establish relational k-arc-consistency on the original problem.*

We assume that CSPs have e constraints, n variables, each with a domains size d. The total number of k-AC clauses is in $O(ed^k)$ and the total number of literals for each k-AC clause (and for the equivalency clauses) is in $O(d^{a-k})$. The space complexity is then still $O(ed^a)$ for any arbitrary constraint and any k.

4 (i, j)-Consistencies in SAT.

In addition to relational k-arc-consistency, k-AC clauses allow us to enforce another very common family of local consistencies (specifically, (i, j)-consistency [Fre85]) by adding the joins of certain constraints and performing the k-AC encoding on this augmented problem.

Definition 3 ((i, j)-Consistency.). *A binary constraint network is* (i,j)-consistent *iff $\forall E_i, E_j$ two sets of i and j distinct variables, any consistent assignment on E_i is a subset of a consistent assignment on $E_i \cup E_j$.*

This family includes many well known consistencies: arc consistency (AC) corresponds to (1,1)-consistency, path consistency (PC) corresponds to (2,1)-consistency, path inverse consistency (PIC) corresponds to (1,2)-consistency.
If on binary networks, arc consistency is often the best choice, higher level of filtering may sometimes be useful, for instance, path consistency is used in temporal reasoning. However, implementing algorithms to maintain any consistency, and moreover, combining this with improvements like (conflict directed) backjumping, requires a lot of work. With our approach, just by setting two parameters, (k and the size of the subsets to consider) and applying any SAT solver to the resulting encoding, you can solve the problem with the chosen consistency combined to all the features of the solver.

Definition 4 (Join of Constraints.). *Let C_{S1}, C_{S2} be two constraints, the join $C_{S1} \bowtie C_{S2}$ is the relation on $S1 \cup S2$ containing all tuples t such that $t[S1] \in C_{S1}$ and $t[S2] \in C_{S2}$.*

Theorem 2 *Enforcing (i,j)-consistency is equivalent to enforcing relational i-arc-consistency on the join of all constraints* involved *in a set of $i+j$ variables, for each of them.*

The space complexity results of section 3 also apply here, but the number of constraints is equal to the number of subsets of $i+j$ vertices in the constraint graph, i.e. $O(n^{i+j})$, and $a = i+j$. Therefore the worst case space complexity is $O(n^{i+j}d^{i+j})$, and so is the worst case time complexity. This is again optimal.

5 Mixed encoding

There is a clear relation between the tightness of a constraint and the performance of DP on that constraint encoded with the direct or a k-AC encoding. Consider the binary *not_equal* constraint. You need only d clauses of size 2 to encode it in the direct encoding while you need $2d$ clauses of size d in the AC-encoding even though AC propagation in *not_equal* is pointless. On the other hand, consider the binary *equal* constraint. This is encoded with $(d-1)^2$ binary clauses in the direct encoding, while you need only $2d$ binary clauses in the AC encoding, and you can expect a lot of AC propagation. The space complexity and the level of propagation is thus linked to the tightness of the constraint. One strategy therefore is to adapt the encoding to the constraint's tightness, i.e. using the direct encoding when the constraint is loose and the AC encoding when it is tight. Moreover we can use, for each constraint, the k-AC clause with the best "adapted" k. The principal issue is to know *a priori* how to pick k. The notion of *m-looseness* [vBD97] give us a way to choose among the different k.

Definition 5 (m-looseness). *A constraint relation R of arity a is called* m-loose *if, for any variable X_i constrained by R and any instantiation I of the remaining $a-1$ variables constrained by R, there are at least m extensions*[3] *of I to X_i that satisfy R.*

Theorem 3 (van Beek and Dechter[vBD97]) *A constraint network with domains that are of size at most d and relations that are m-loose is relationally $(k,(\lceil \frac{d}{d-m} \rceil - 1))$-consistent for all k.*

We can restrict this to relational $(k,1)$-consistency (that is relational k-arc-consistency) and then we have the relation $\lceil \frac{d}{d-m} \rceil - 1 \geq 1$ which is reduced to : $m \geq \frac{d}{2}$. This means that, given a subset of variables, if all the relations that constrain these variables are $\frac{d}{2}$-loose or more (every instantiations of this subset minus one variable have at least $\frac{d}{2}$ supports on this variable) then these constraints are relationally k-arc-consistent for any k. Therefore enforcing relational k-arc-consistency will not give any pruning, at least initially. In addition, the direct encoding would be more compact for such constraints.

In the *mixed* encoding, k is adapted to the number of supports of any k-instantiation for k in any interval between a lower bound and the arity of the

[3] i.e., supports.

constraint. This interval has an arbitrary length M. For each $k, (a - M) \leq k < a$, let $T[k]$ the threshold associated to k. For each k-instantiation I, if I is not covered by a former clause, that is, there is not a former clause which premiss is subset of I[4], we count its number of supports. If I has less than $T[k]$ supports, then the corresponding k-AC clause is added, otherwise, we do the same operation with all the $(k+1)$-instantiations that contain I. A simple example is the mixed encoding with $M = 1$ on binary constraint, that is a mix of support and direct encodings. There is only one k to consider : 1, and $T[1] = \frac{d}{2}$ (because of the theorem 3). For each 1-instantiation of each constraint, that is for each value, the mixed(1) encoding contains the support clause for this value, iff it has less than $\frac{d}{2}$ supports, and all the conflict clauses with this value otherwise. The size of the k-AC clauses is then bounded by $T[k] + k$.

6 Experimental Results

We have performed a set of experiments to assess the respective charasteristics of the different encodings introduced. However, space limitation prevent us giving many details. Here are the main conclusions we can draw from those experiments.

- Encoding the supports tends to be more pruningful, and then more efficient than encoding conflicts. A DP algorithm on the best k-AC encoding is several times faster than on direct encoding for hard instances.
- The best choice usually is $(a-1)$-AC encoding, where a is the arity of the encoded constraint i.e. AC encoding [Gen02] for binary networks, 2-AC for ternary, etc. The reason is that other k-AC clauses need equivalence clauses and extra variables, increasing the number of unit propagations required for the same filtering.
- The performances of a DP solver on high filtering k-AC encoding (all but direct) are better on tight constraints than on loose. The main reason is that k-AC clauses encode supports, and they are by definition more numerous in loose constraints.
- For some structured problems (see [vBW01]), a DP algorithm on the mixed encoding is almost always faster than any other encoding. This seems to show that adapting the level of filtering depending on the constraint pay off for problem less homogeneous than random problems.
- For binary networks, a state of the art SAT solver, BerkMin [GN02], on k-AC encoding is between 4 and 15 times slower, depending on the tightness of the constraints, than a state of the art CSP solver [BR01].
- However, for non-binary networks, whereas implementing a good CSP algorithm is not easy, BerkMin on $(a-1)$-AC encoding can be faster than a state of the art CSP solver [BMFL02], on networks with tight constraints.

[4] we could skip this step, and consider an instantiation even if it is already covered, but these clauses are not required for correctness.

7 Conclusion

We presented a new family of mappings of constraint problems into satisfaction problems, and proved the optimality in space and time complexity of these encodings. We also proved that performing full unit propagation on k-AC encoding is the same as enforcing relational k-arc-consistency on the original problem, or used in a slightly different way, (i,j)-consistency. We showed how to mix the differents encodings to take advantage of their best individual features. And finally we demonstrated preliminary experimental results of the efficiency of the introduced encodings.

From a constraint programming perspective, these new encodings are a very easy way to implement and test algorithms for enforcing a wide range of filterings, all in optimal worst case time complexity. Such encodings also profit from the sophisticated branching heuristics and other algorithmic features of the SAT solver (like non-chronological backtracking and nogood learning). Given the recent rapid advances in SAT solvers, they offer an alternative way to solve hard problem instances. From the satisfiability perspective, these encodings are useful for modelling, since many real life problems are likely to have straightforward representations as CSPs whereas SAT models are often not as easy to make. Modelling is also far more understood for CSPs than for SAT. These encodings allow the SAT research community to take advantage therefore of CSP modelling results.

References

[BMFL02] C. Bessière, P. Meseguer, E.C. Freuder, and J. Larrosa. On forward checking for non-binary constraint satisfaction. *AI*, 141:205–224, 2002.

[BR01] C. Bessière and J.C. Régin. Refining the basic constraint propagation algorithm. In *Proceedings IJCAI'01*, pages 309–315, Seattle WA, 2001.

[DLL62] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem proving. *Communications of the ACM*, 5(7):394–397, July 1962.

[DvB95] R. Dechter and P. van Beek. Local and global relational consistency. *LNCS'99*, 976:240–257, 1995.

[Fre85] E.C. Freuder. A sufficient condition for backtrack-bounded search. *JACM*, 32:755–761, 1985.

[Gen02] I.P. Gent. Arc consistency in SAT. In *Proceedings ECAI'02*, 2002.

[GN02] E. Golberg and Y. Novikov. Berkmin: a fast and robust sat-solver. In *DATE2002*, pages 142–149, 2002.

[Kas90] S. Kasif. On the parallel complexity of discrete relaxation in constraint satisfaction networks. *Artificial Intelligence*, 45:275–286, 1990.

[vBD97] P. van Beek and R. Dechter. Constraint tightness and looseness versus local and global consistency. *J. of the ACM*, 44:549–566, 1997.

[vBW01] P. van Beek and K. Wilken. Fast optimal instruction scheduling for single-issue processors with arbitrary latencies. *LNCS'01*, 2239:625–??, 2001.

[Wal00] T. Walsh. SAT v CSP. In *Proceedings CP'00*, 2000.