Local Consistencies in SAT

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

[1] LIRMM-CNRS, Université Montpellier II
`bessiere@lirmm.fr`
[2] Cork Constraint Computation Centre, University College Cork
`{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 closely related 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 the other side. Enforcing a local consistency is one of the most important aspect of systematic search algorithms. For CSPs, in particular, enforcing arc consistency is often the best tradeoff between the amount of pruning and the cost of pruning. The AC encoding [12] has the property that arc consistency in the original CSP is established by unit propagation in the encoding [10]. A complete backtracking algorithm with unit propagation, such as DP [6], therefore explores an equivalent search tree to a CSP algorithm that maintains arc consistency. Likewise, DP on the Direct encoding behaves as the *Forward Checking* algorithm which maintains a weaker form of Arc Consistency [17]. In this paper we show that there is a continuity between direct and support encodings, and following this line, many other consistencies can be simulated by unit propagation in the SAT encoding, for any constraint arity and all in optimal worst case complexity.

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* [8] to be established by unit propagation on the SAT encoding. They work with any constraint arity. Section 4 focuses on binary networks, and shows that these encodings can also be used to establish any *(i,j)-consistency* (another large family of consistencies [9]). 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 combine 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 encoding. 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 incompatible 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 on 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 on S, then I doesn't belong to any solution.

2.2 Direct Encoding

The direct encoding [17] 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 [12] 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.
Support clause : Let X, Y be two variables, $v \in D(X)$ a value of X and $\{w_1, \ldots, w_k\}$ the supports of $X = v$ on 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 Xv 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 on 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*, represent supports on subsets S of variables of any size, for an instantiation of another subsets T of any size. Since a literal stands for an assignment, an *instantiation* (or a support) of several variables corresponds to a *conjunction* of positive literals. A k-AC clause represents the implication between the instantiation and its supports: if the instantiation holds, one of the supports must hold. Let $[v_1, \ldots, v_p]$ be a support on $X_1, \ldots, X_p$ of a given instantiation on other variables. The conjunction that encodes 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 $(\neg X_1v_1 \vee \ldots \vee \neg X_pv_p \vee s)$. We call s *support-variable*. If the support is unit (say $Y = v$), then there is no need for an extra variable, and the support-variable is the corresponding boolean variable (Yv).
k-AC clause : Let C_S be a constraint, $T = \{X_1, \ldots X_k\} \subset S$ be a set of k variables, $I = [v_1 \in D(X_1), \ldots v_k \in D(X_k)]$ an instantiation of T and $\{s_1, \ldots, s_m\}$ the supports of I on $S - T$, then we add the *k-AC* clause : $\neg X_1v_1 \vee \ldots \vee \neg X_kv_k \vee s_1 \vee s_2 \ldots \vee s_m$.
This clause is equivalent to $I \rightarrow (s_1 \vee s_2 \vee \ldots \vee s_m)$ which means : as long as I holds then at least one of its support must hold. Therefore when all the supports

of I are falsified I is itself falsified i.e, the k-AC clause is reduced to the conflict clause of length k forbidding I.

In figure 3, 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.

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

$\Rightarrow_{encoding}$

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

Table 1. First table: a ternary constraint involving the variables X, Y, Z, the allowed tuples are given. Second table: four possible k-AC encodings of this constraint, T = True and F = False.

The k-AC clauses are a generalisation of support clauses in two different ways. First they capture a larger family of consistencies, *relational k-arc-consistency* (section 3) and (i,j)*-consistency* (section 4). Second they work for any constraint arity. 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.

Recall that in a CSP, a nogood is a forbidden set of assignments, $\neg(X_1 = v_1 \wedge \ldots \wedge X_i = v_i)$. And that a Boolean variable correspond to an assignment, the atom X_iv_i represents $X_i = v_i$. For the theorems and proofs below, the word variable will refer to a *CSP* variable, *assignment* to a Boolean variable of the encoding and *support* to a conjunction of assignments in the conclusion of a k-AC clause. An interpretation I is a function that associates a value in $\{0,1\}$ to the atoms of a set of clauses $\mathcal{B}$. I is a model ($I(\mathcal{B}) = True$), iff all the clauses in $\mathcal{B}$ are satisfied by I.

Theorem 1 (Correctness and completeness of the k-AC Encoding.) *I is a model of the set with the* ***at-least-one, at-most-one,*** *and* ***k-AC*** *clauses, iff the assignment such that a variable X take a value v iff $I(Xv) = T$ is a solution of the original constraint network.*

Proof: Suppose that all the assignments of a nogood N are satisfied.

$$N = \neg(X_1v_1 \wedge X_2v_2 \wedge \dots X_nv_n)$$

Let C be the k-AC clause which premiss P is a subset of this nogood

$$C = (X_1v_1 \wedge X_2v_2 \wedge \dots X_kv_k) \rightarrow (s_1 \vee s_2 \vee \dots s_m),\ s_j = (X_{k+1}j_{k+1} \wedge \dots X_nj_n)$$

and let S be the rest of this nogood, $S = N - P$. This premiss is satisfied and then the conclusion must be satisfied. Now recall that at-least and at-most clauses ensure that one and only one assignment per (CSP) variable is satisfied. All the supports in C refer to the same variables but are by definition different from S by at least one assignment, (say X_iv_i is the assignment in the nogood, and X_ij_i is the assignment in the support). Since, for this variable, X_iv_i is satisfied, therefore X_ij_i is not, and then the whole conclusion is not satisfied.

Let S be a solution of the original constraint network, and let I be the assignment in which $I(X_iv) = T$ iff, in S, the value v is given to the variable X_i. S gives one and only one value to each variable, the **at-most-one** and **at-least-one** clauses are thus satisfied. Without loss of generality, let C be a k-AC clause which premiss P is an assignment on a set R and conclusion are supports on a set T. If S is a solution then $S[R \cup T]$ is consistent and then $S[T]$ is a support of $S[R]$. Either $P \neq S[R]$ and then C is satisfied since the premiss is falsified, or $S = S[R]$ and then $S[T]$ is one of its support and belongs to C's conclusion. C is then satisfied since both premiss and conclusion are satisfied. $\square$

Unit propagation on the k-AC Clauses corresponds exactly to enforcing relational k-arc-consistency. Relational arc-consistency [8] 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 1 (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 $s_1 \vee s_2 \vee \dots \vee s_m$.

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, it is removed from the corresponding constraint, i.e, this tuple is now explicitly forbidden. 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, $\mathcal{H} = \neg I$. To prove the equivalence between unit propagation on those encodings, and relational k-arc consistency on the original problem we first recall some definitions given in [1] and slightly modified for our purpose.

A CSP is said to be *empty* if at least one of its variables has an empty domain or at least one of its constraints is empty, i.e, forbids all assignments.

We denote `sat2csp`$(\mathcal{P})$ the transformation of a SAT-encoded CSP into a CSP consisting of a variable X_i with a domain $D(X_i) = [v_1, \ldots, v_d]$ for each at-least-one clause $X_i v_1 \vee \ldots \vee X_i v_d$ in $\mathcal{P}$, and a constraint forbidding the nogood $N = (X_1 = v_1 \wedge \ldots \wedge X_k = v_k)$ for each conflict clause $(\neg X_1 v_1 \vee \ldots \vee \neg X_k v_k)$ (or support clause reduced to a conflict clause by unit propagation).

First we show that the relational k-arc consistent closure of a CSP $\mathcal{P}$, written `r-`k`-AC`$(\mathcal{P})$ is empty iff the k-AC encoding of $\mathcal{P}$ has an empty image under `sat2csp`, that is, `sat2csp`$(k$`-sat`$(\mathcal{P}))$ is empty. We ignore the isuue of discovering the emptyness. This is trivial, both in the original problem and in the encoding, when the empty constraint arity is 1, whereas it is not for other arities, though it remains polynomial. Usually, this will be quickly discovered, providing that the empty constraint is small and that the branching heuristic chooses first the variables of this constraint.

Second we prove that, assuming the same branching choices, this equivalence is maintained at each node of the search tree by unit propagation in the encoding. As a corollary, unit propagation on k-AC encoding prunes the search tree equivalently to relational k-rac consistency on the original problem.

Lemma 1 *($\mathcal{P}$) is empty after enforcing relational k-arc consistency iff* `sat2csp`*(k*`-sat`*($\mathcal{P}$)) is empty.*

Proof: The relational k-arc consistent closure of $\mathcal{P}$ contains all the nogoods of length k forbidding k-instantiations that don't have any support on the rest of the constraint they belong to. By definition, the k-AC clause for such an instantiation is the conflict clause of length k corresponding to the nogood. The later therefore belongs to `sat2csp`$(k$`-sat`$(\mathcal{P}))$. Moreover, all nogoods of length k are added to the relational k-arc consistent closure if and only if they are not supported. Therefore, for any nogood N of length k, $N \in$ `r-`k`-AC`$(\mathcal{P})$ iff $N \in$ `sat2csp`$(k$`-sat`$(\mathcal{P}))$

Beside, if $\mathcal{P}$ is emptied by relational k-arc consistency, then the empty constraint arity is always k, since only nogoods of size k are added during the process. $\square$

The proof of Lemmas 1 is based on the fact that the supports of an instantiation are equivalent in the encoding and in the orginal problem. Unit propagation ensures that this is the case as well during search. We consider a relationally k-arc consistent CSP $\mathcal{P}$, an assignment $X = v$ and the induced subproblem $assign(X = v, \mathcal{P})$. In the SAT encoding this corresponds to $assign(Xv = T, k\text{-}\mathtt{sat}(\mathcal{P}))$. We prove that an instantiation looses a support because of an assignment in the CSP if and only if the k-AC clause of this instantiation looses the same support in the encoding by unit propagation of the truth assignment.

Lemma 2 *If an instantiation J, support of another instantiation I in $\mathcal{P}$ is not a support anymore in $assign(X = v, \mathcal{P})$ for relational k-arc consistency, then the support-variable s_J of the corresponding k-AC clause is set to False after unit propagation.*

Proof:

Without loss of generallity, let I be an instantiation on a set T of k variables of a constraint C_S. Let J be a support of I for C_S in $\mathcal{P}$, such that J is not a support of I in $assign(X = v, \mathcal{P})$.

Implicitly, after the assignment $X = v$, all other values in $D(X)$ are removed. If J is not a support, it means that $\exists X \in T - S$, such that $J[X]$ has been removed from its domain ($J[X] \neq v$).

In the encoding, the assignment $Xv = T$, propagated to the at-most-one clauses yelds the assignments $Xw = F$ for all $w \neq v$. Let s_J be the proposition standing for the support J, then the equivalence clause $(\neg s_J \vee J[X])$ gives the unit clause $\neg s_J$, which is propagated to the k-AC clause. Consequently, the support-variable s_J is set to False (it is not a "support" in the encoding either). At any point of the resolution, a support-variable s_J belongs to the conclusion of a k-AC clause (is not assigned to False) iff its corresponding support J holds in the constraint network. □

Lemmas 1 establishes that if the supports are the same in the original and in the encoded problem, then the problem is empty iff the reformulation is empty. Lemma 2 shows that this is the case during search.

Theorem 2 *Performing full unit propagation on at-least-one, at-most-one and k-AC clauses during search is equivalent to maintain relational k-arc-consistency on the original problem.*

From this follows a strict equivalence between the search trees of an algorithm that maintains relational k-arc consistency in the original problem, and an algorithm that enforces unit propagation on the reformulation.

3.1 Complexity of k-AC Encoding

We assume that n is the number of variables, d is the size of the domains, e is the number of constraints and a denotes their arity. We can ignore the at-most and at-least clauses : there are n at-least clauses each containing d literals, and

nd^2 at-most clauses, which are binary. This $O(nd^2)$ space complexity is in all cases lower than the worst space complexity of the k-AC clauses. We therefore focus on the size of the k-AC clauses themselves.

The total number of k-AC clauses is bounded by $e\binom{a}{k}d^k$. We need to cover all the constraints (e). For each constraint, we consider all the subsets of k variables of that constraint ($\binom{a}{k}$). And for each subset, we consider all the instantiations (d^k). The total number of literals for each k-AC clause is bounded by $k+(3(a-k)+2)(d^{a-k}-1)$. The premiss contains k literals, and the conclusion at most $d^{a-k}-1$. Furthermore, if $a-k>1$, there are also $(d^{a-k}-1)(3(a-k)+1)$ additional literals from the equivalency clauses: Each one gives 1 clause of size $a-k+1$ and $a-k$ clauses of size 2. The space complexity is then $O(ed^k)$ clauses of $O(d^{(a-k)})$ literals, which is still $O(ed^a)$ for any arbitrary constraint and any k. Note that the space complexity of the reformulation and of the original problem are the same. Since unit propagation can be established in linear time, the time complexity is also in $O(ed^a)$, which is optimal worst case time complexity.

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 [9]) by adding the joins of certain constraints and performing the k-AC encoding on this augmented problem.

Definition 2 ((i, j)-Consistency.). *A binary CSP 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 other consistency, and moreover, combining this with improvements like (conflict directed) backjumping and learning requires a lot of work. With our approach, just by setting two parameters, (k and the size of the subsets to consider) and applying a SAT solver to the resulting encoding, we can solve the problem with the chosen consistency and all the other features of the SAT solver.

Definition 3 (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 3 *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.*

Proof: Let E_i be a set of i variables, If I, a consistent instantiation on E_i, is (i,j)-inconsistent, then there exists a set E_j of j variables such that $\forall IJ$ a consistent instantiation on $E_i \cup E_j, IJ[E_i] \neq I$. Let C be the constraint induced by the join of all the constraints involved in $E_i \cup E_j$. C is the set of all the allowed, i.e, consistent, instantiations on $E_i \cup E_j$, but I is consistent and $I \notin C[E_i]$, therefore C is relationally i-arc-inconsistent (see def 1). Conclusion : if I is (i,j)-inconsistent, then for any set E of $i+j$ variables containing the variables of I, the constraint obtained by joining all constraints which scopes are subsets of E is relationally i-arc-inconsistent. □

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. It can be encoded by d conflict clauses of size 2 with the direct encoding, whilst $2d$ clauses of size d are required in the AC-encoding even though AC propagation in *not_ equal* doesn't achieve much pruning. 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* [14] give us a way to choose among the different k.

Definition 4 (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 supports of I to X_i that satisfy R.*

Theorem 4 (van Beek and Dechter[14]) *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.*

Proof: See [14]. □

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. This suggests to use support clauses whenever the number of supports is lower than $\frac{d}{2}$ and conflict clauses otherwise. Moreover, for a given constraint arity a, the choice is now extented to any k-AC clause with k between 1 and a. To make a choice, we associate a treshold T_k on the number of supports above which we choose $(k+1)$-AC clauses rather than k-AC to encode a particular instantiation. To compute the mixed encoding of a given constraint we use the following algorithm:

First we consider all the instantiations of size 1 (all the values of all the variables), and for each of them we count the number of supports (of size $a-1$), if this number is less than T_1 then we add the corresponding 1-AC clause. In a second step, we consider all the instantiations of size 2 containing a non-yet-encoded instantiation of size 1, if the number of supports of this instantiation is less than T_2 we encode it with a 2-AC clause, and so on for a steps.

We propose $T_{a-1} = \frac{d}{2}$ whilst we don't have yet any sound value for T_k with k less than $a-1$.

Theorem 5 (Correctness and completeness of the mixed encoding.) *I is a model of the set containing the **at-least-one**, **at-most-one**, and any k-**AC** clauses, according to the rules above, iff I is a solution of the constraint network.*

Proof: By definition, all the nogoods have at least one k-AC clause which premiss is one of its subsets, then theorem 1 (correctness) can be applied.

Theorem 1 says that k-AC encoding is complete for any k, therefore, k-**AC** clauses are satisfied by I, and so is any combination of them. $\square$

5.1 Complexity

Theorem 6 *The mixed encoding ($k = [1, 2]$) requires less than $\frac{3}{2}d^2$ literals to encode a binary constraint, This limit can be asymptoticly reached.*

Proof: Let us consider the Boolean matrix of the constraint. Let r (resp c) be the number of *rows* (resp *colomns* encoded with support clauses, $0 \leq r, c \leq d$. These clauses have each less than $d/2$ literals, so we have $(r+c)\frac{d}{2}$ literals for the supports clauses. There are $(d-r)(d-c)$ elements of the matrix wich are not covered by the support clauses. Besides, there are $r(d-c)$ and $l(d-r)$ elements which are covered by only one support clause. On each row/colomn containing these elements, there are at most $\frac{d}{2}$ 0, so at most, half of them are 0. For each 0 we need a conflict clause of 2 literals, then, to encode these elements, we need $(r+c)\frac{d}{2} - rc + d^2$ literals. If $c > \frac{d}{2}$, then this number of literals increase when r decrease, and if $c < \frac{d}{2}$, then it increase with r. This number is maximized when r is max and l min or vice versa. The worst case is then $r = d$ and $c = 0$, in that case there are $\frac{3}{2}d^2$ literals.

Let C be a constraint on variables with odd domains and relation matrix as given in the margin (that is a checkerboard of 0 and 1, plus a full column of 0 and a full but one row of 1). The rows are all but one encoded with support clauses, half of this clauses are of size $\lfloor \frac{d}{2} \rfloor$ and the other half are of size $\lceil \frac{d}{2} \rceil$, the last row is encoded with a nogood. All the colomns but one are encoded with $\lfloor \frac{d}{2} \rfloor$ conflict clauses, the remaining one with a unary support clause. That is $3 + \lfloor \frac{d}{2} \rfloor^2 + \lfloor \frac{d}{2} \rfloor \times \lceil \frac{d}{2} \rceil + \lfloor \frac{d}{2} \rfloor \times 2 \times (d-1)$ literals, which is asymptoticly equal to $\frac{3}{2}d^2$ □.

1	0	1	. . .	0	0
0	1	0	. . .	1	0
1	0	1	. . .	0	0
.	.	.	. . .	.	0
.	.	.	. . .	.	0
0	1	0	. . .	1	0
1	1	1	. . .	1	0

6 Experimental Results

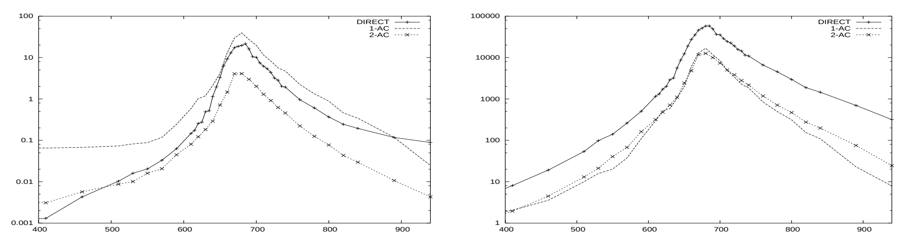
We have performed a set of experiments to compare the different encodings. Section 6 and 6 give a concrete idea of the improvement, in term of pruning and cpu time, in comparison with direct encoding. In section 6 we also show that mixed encodings are an even better way to encode heterogeneous or structured problems. And finally, in section 6 we compare this approach with the state of the art in CSP. For all the random instances, we used Bessiere and Frost's random generator. The CSPs are defined by 5 parameters, the number of variables, the size of the domains, the density (i.e, the number of constraints), their arity and their tightness (i.e, the number of nogoods per constraint). The four first parameters are fixed and the tightness is given on x axis, the y axis giving the cpu time or the number of backtracks. We generally focused on results at the phase transition, when the number of satisfiables instances is the closest to 50%. We used Berkmin SAT solver [11] on generated cnf files.

***k*-AC Encodings.** This experiment involves 1-AC, 2-AC and direct encoding on the following class of ternary networks, 30 variables, 10 values, 60 constraints. 1-AC and 2-AC encodings need both 5 times less backtracks than direct encoding[3] at the phase transition. But only 2-AC encoding translate this greater filtering into a cpu time reduction (again a factor 5). We can explain this by the amount of propagations needed to perform the same filtering in 1-AC encoding, because of the extra variables.

(i,j)-Encodings. This experiment involves PIC encoding, AC encoding and direct encoding on two classes of networks. A sparse class, 150 variables, 15 values, 350 binary constraints, and a dense one, 70 variables, 10 values, 310 binary constraints. According to the theory, PIC prunes even more the search tree than AC, (i.e, the backtracks are less numerous). However, on dense networks, where the gain in pruning is more evident, the amount of extra variables, as previously for 1-AC encoding, slow down the resolution.

Mixed Encoding. To emphasize the benefits of the mixed encoding on more structured problem, we used the Instruction Scheduling Problem, introduced in [15], The problem is to find a minimum length instruction schedule for a basic block of instructions (a straight-line sequence of code with a single entry

[3] experiments with Chaff showed an even greater difference, about a factor 10 for backtracks, and 15 for cpu time.



(a) cpu time (b) number of backtracks

Fig. 1. Cpu time and number of backtracks of BerkMin on GAC (1-AC), 2-AC and Direct (3-AC) encoding.

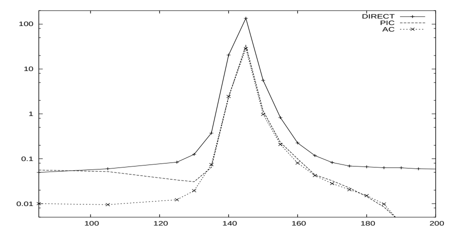
point) subject to precedence, latency, and resource constraints. Basic blocks are represented as DAG (Direct Acyclic Graph). To model this problem, van Beek used one variable for each instruction, its domain represents the possible positions in the total order we have to find. The constraints are: $instruction_i < instruction_j + k$ for each arc ij labelled with k in the DAG ($instruction_j$ must wait at least k cycles after $instruction_i$), and an AllDiff constraint on all the variables. The domains are initiallized with a lower bound on the number of cycles required, and the instance is solved, if no solution is found they are incremented and the instance is solved again. The first solution encountered is the optimal solution. Each point of the figure 6 represents the runtime Berkmin needed to find the optimal solution on the mixed encoding (x axis) and AC encoding (y axis). (all instances have the same parameters : 20 instruction, 40 constraints of latency and a latency between 1 and 3 inclusive). Figure 6 compares the mixed and direct encodings. The mixed encoding is almost always better in cpu-time, compared to the direct or the AC encoding. The number of backtracks is nearly the same as in AC encoding, while the space complexity is greatly reduced (mostly because of the alldiff constraint).

Comparison with the State of the Art in CSP. We also measured the efficiency of this approach in comparison with the state of the art for CSP solvers. We have done these comparisons on the following classes:

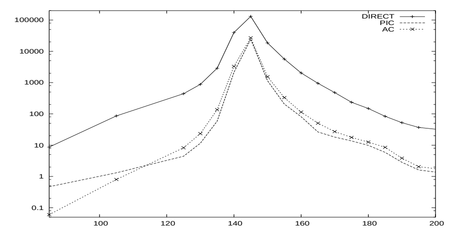
- (a) binary sparse : <180 variables, 15 values, 450 constraints, 147 nogoods>.
- (b) binary dense : <90 variables, 10 values, 400 constraints, 38 nogoods>.
- (c) ternary dense : <10 variables, 10 values, 100 constraints, 208 nogoods>.
- (d) ternary medium : <30 variables, 6 values, 75 constraints, 109 nogoods>.
- (e) ternary sparse : <50 variables, 10 values, 70 constraints, 790 nogoods>.

For binary classes, 100 instances were generated and solved by MAC[4] , (Maintain Arc Consistency) with AC2001 algorithm [5], and the dynamic variable ordering (dvo) H1_DD_x [2], which outperforms the well known dom/deg heuristic. The

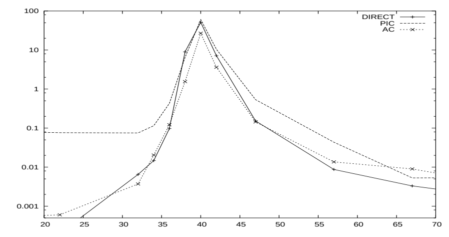
[4] For MAC the number of backtracks can be slightly overestimated, since the value given is in fact the number of visited nodes.



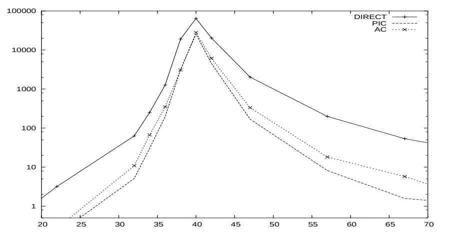
(a) cpu time, sparse networks



(b) backtracks, sparse networks



(c) cpu time, dense networks



(d) backtracks, dense networks

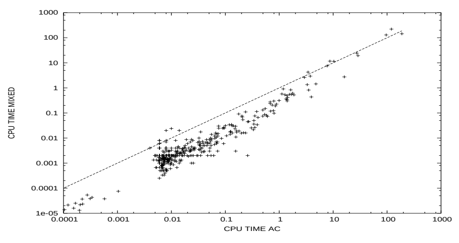
Fig. 2. Cpu time and number of backtracks of BerkMin on Direct, AC and PIC encoding for two classes of networks.

same instances were translated in SAT problems with AC and PIC encoding, and then solved by BerkMin561.

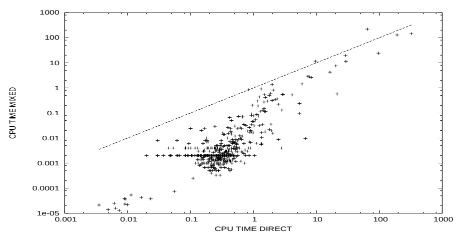
For ternary classes 100 instances were also generated and solved by NFCx [3], where x is 0 or 5, using GAC2001 [5], and dom/deg dvo [4] without singleton propagation [13]. Here again, the same instances were translated with 1-AC, 2-AC, 3-AC, mixed(1) and mixed(2) encodings, and solved by BerkMin561 [5]. The results of our approach take also into account the translation duration, which include the time spent on reading the csp file and writing the cnf. Note that this duration is insignificant when the problem is really hard, and can be dramaticly reduced by not creating a temporary file[6]. The first observation is that the performance of BerkMin on high filtering k-AC encodings (all but direct) is better on sparse than on dense networks. There are at least two reasons for that behaviour : firstly, for dense networks, at the cross-over point, the constraints are loose, and then there is not much propagation. Moreover, recall that k-AC clauses encode the *supports*, and they are more numerous when the constraints are loose. A 1-AC clause (and its equivalence clauses) for a ternary constraint can have between 1 and $3d^2$ literals, according to the number of supports, that can therefore make a great difference for the SAT solver. However, The results below show that this approach can really handle large and hard problems. The best

[5] all Christian Bessiere's algorithms ran on a 1.6 GHz pentium, whereas BerkMin ran on a 1.8 GHz one, BerkMin's results are then corrected by a factor 1.8/1.6.

[6] most of this time is spent on i/o



(a) mixed vs AC



(b) mixed vs direct

Fig. 3. 20 instructions, 40 latency constraints, max latency 3. cpu time for BerkMin on Mixed encoding (y axis) and AC encoding or Direct encoding (x axis).

class (a)	MAC2001	AC + BM	PIC + BM
#backtracks	55559	66749	62006
total time	**39.6**	165	178
translation	N/A	1.37	1.2

class (b)	MAC2001	AC + BM	PIC + BM
#backtracks	56718	136139	103173
total time	**17.0**	354	373
translation	N/A	0.5	0.3

(a) results of MAC2001 and BerkMin on AC and PIC encodings.

		NFC	1-AC	2-AC	3-AC	mix(1)	mix(2)
(c)	time	**0.1**	6.5	5.5	2.1	5.5	5.4
(c)	trans	N/A	0.87	1.8	1.8	1.9	2
(d)	time	**0.37**	3.26	0.84	1.14	0.84	0.86
(d)	trans	N/A	0.37	0.42	.72	0.45	0.46
(e)	time	18.40	59.8	15.5	85.8	11.4	**9.5**
(e)	trans	N/A	2.4	1.6	2.4	1.6	1.6

(b) Results of NFC and BerkMin on different encodings on 3 classes of ternary networks.

Fig. 4. total time is cpu time for MAC and BerkMin's cpu time + translation duration, all in seconds.

algorithm should probably always be to solve the original problem rather than its reformulation, but when good algorithms are hard to make, reformulation is a good alternative. For example NFC is certainly more distant from the "best possible algorithm" than MAC is, and then BerkMin on the right k-AC encoding is very close, and sometimes better, than NFC. In the same way, there are very few good PIC [7] or "Maintain Relational K-Arc Consistency " algorithms.

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 different 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.[7] Such encodings also profit from

[7] this goal was also pursued in [16], though the approach was completly different

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 of CSP modelling results.

Acknowledgements

We thank Ian P. Gent for helpful discussions about SAT encoding. We also thank Eugene Goldberg for his giving us a slightly modified version of Berkmin561. This work was supported by Science Foundation Ireland.

References

1. F. Bacchus, X. Chen, P. van Beek, and T. Walsh. Binary vs. non-binary constraints. *Artificial Intelligence*, 140(1-2):1–37, 2002.
2. C. Bessière, A. Chmeiss, and L. Saïs. Neighborhood-based variable ordering heuristics for the constraint satisfaction problem. In *Proceedings CP'01*, pages 565–569, 2001. Short paper.
3. C. Bessière, P. Meseguer, E.C. Freuder, and J. Larrosa. On forward checking for non-binary constraint satisfaction. *Artificial Intelligence*, 141:205–224, 2002.
4. C. Bessière and J.C. Régin. MAC and combined heuristics: two reasons to forsake FC (and CBJ?) on hard problems. In *Proceedings CP'96*, pages 61–75, 1996.
5. C. Bessière and J.C. Régin. Refining the basic constraint propagation algorithm. In *Proceedings IJCAI'01*, pages 309–315, 2001.
6. M. Davis, G. Logemann, and D. Loveland. A machine program for theorem proving. *Communications of the ACM*, 5:394–397, 1962.
7. Romuald Debruyne. A property of path inverse consistency leading to an optimal PIC algorithm. In *Proceedings ECAI'00*, pages 88–92, 2000.
8. R. Dechter and P. van Beek. Local and global relational consistency. *Theoretical Computer Science*, 173(1):283–308, 1997.
9. E.C. Freuder. A sufficient condition for backtrack-bounded search. *Journal of the ACM*, 32:755–761, 1985.
10. I.P. Gent. Arc consistency in SAT. In *Proceedings ECAI'02*, 2002.
11. E. Golberg and Y. Novikov. Berkmin: a fast and robust sat-solver. In *Proceeding DATE'02*, pages 142–149, 2002.
12. S. Kasif. On the parallel complexity of discrete relaxation in constraint satisfaction networks. *Artificial Intelligence*, 45:275–286, 1990.
13. B.A. Nadel. Constraint satisfaction algorithms. *Computational Intelligence*, 5:188–224, 1989.
14. P. van Beek and R. Dechter. Constraint tightness and looseness versus local and global consistency. *Journal of the ACM*, 44:549–566, 1997.
15. Peter van Beek and Kent Wilken. Fast optimal instruction scheduling for single-issue processors with arbitrary latencies. *Lecture Notes in Computer Science*, 2239:625–639, 2001.
16. G. Verfaillie, D. Martinez, and C. Bessière. A generic customizable framework for inverse local consistency. In *Proceeding AAAI'99*, pages 169–174, 1999.
17. T. Walsh. SAT v CSP. In *Proceedings CP'00*, pages 441–456, 2000.