

# Propagation in CSP and SAT

Yannis Dimopoulos<sup>1</sup> and Kostas Stergiou<sup>2</sup>

<sup>1</sup> Department of Computer Science  
University of Cyprus, Nicosia, Cyprus  
yannis@cs.ucy.ac.cy

<sup>2</sup> Department of Information and Communication Systems Engineering  
University of the Aegean, Samos, Greece  
konsterg@aegean.gr

**Abstract.** Constraint Satisfaction Problems and Propositional Satisfiability, are frameworks widely used to represent and solve combinatorial problems. A concept of primary importance in both frameworks is that of constraint propagation. In this paper we study and compare the amount of propagation that can be achieved, using various methods, when translating a problem from one framework into another. Our results complement, extend, and tie together recent similar studies. We provide insight as to which translation is preferable, with respect to the strength of propagation in the original problem and the encodings.

## 1 Introduction

CSPs and SAT are closely related frameworks that are widely used to represent and solve combinatorial problems. It is well known that there exist several ways to translate a problem expressed in one framework into the other framework (for example [2, 14]).

One of the most important concepts in CSP and SAT is the concept of constraint propagation. Solvers in both frameworks utilize propagation algorithms both prior to and during search to prune the search space and save search effort. Recently there have been several studies exploring and comparing the amount of propagation that can be achieved in each framework using standard techniques, such as arc consistency (in CSPs) and unit propagation (in SAT), under various encodings from one framework to another [3, 4, 9, 14]. A general lesson learned from these studies is that the choice of encoding is very important when comparing propagation methods in different frameworks. For example, arc consistency in a binary CSP is equivalent to unit propagation in the *support encoding* of the CSP into SAT [9, 10]. On the other hand, arc consistency is stronger than unit propagation under the *direct encoding* [14].

Apart from the variety of ways to translate problems from one framework into another, a second source of complexity (and confusion) when comparing propagation techniques in different frameworks is the large number of such techniques that have been proposed. So far, the comparisons between propagation methods in CSPs and SAT have only considered standard techniques like arc consistency

and forward checking, on one hand, and unit propagation on the other hand. Although these techniques remain at the core of most CSP and SAT solvers respectively, other stronger propagation methods are also attracting considerable interest in both communities in recent years. For example, some SAT solvers (e.g. `kcnfs`, `march_dl`, `Dew_Satz`, `2CLS+EQ`) employ strong reasoning techniques such as failed literal detection and variants of binary resolution [1, 6, 11]. Also, strong consistencies such as singleton and inverse consistencies are beginning to emerge as possible alternatives to arc consistency in CSPs [5, 8].

In this paper we make a detailed theoretical investigation of the relationships between strong propagation techniques in CSPs and SAT. We consider various encodings of SAT into CSP and binary CSPs into SAT. In some cases we prove that, under certain encodings, there is a direct correspondence between a propagation method for CSPs and another one for SAT. For example, failed literal detection applied to a SAT problem achieves the same pruning as singleton arc consistency applied to the literal encoding of the problem into a CSP. In other cases, where no direct correspondence can be established, we identify conditions that must hold so that certain consistencies can achieve pruning in a SAT or CSP encoding, and/or place bounds in the pruning achieved by a propagation technique in one framework in terms of the other framework. For example, we show that failed literal detection applied to the direct encoding of a CSP is strictly stronger than restricted path consistency and strictly weaker than singleton arc consistency applied to the original CSP. Finally, we introduce new propagation techniques for SAT that capture the pruning achieved by certain CSP consistencies. For example, we introduce *subset resolution*; a form of resolution that captures the pruning achieved by arc consistency in the dual encoding of a SAT problem into a CSP.

Our results provide insight and better understanding of propagation in CSPs and SAT and complement recent similar studies. Also, we give indications as to when encoding a problem is beneficial and which encoding should be preferred, with respect to the strength of propagation in the original problem and the encodings. Note that, due to space restrictions, we only give some of the proofs (or sketches of proofs) for our theoretical results.

## 2 Preliminaries

A CSP,  $P$ , is defined as a triple  $(X, D, C)$ , where:  $X = \{x_1, \dots, x_n\}$  is a finite set of  $n$  variables,  $D = \{D(x_1), \dots, D(x_n)\}$  is a set of domains, and  $C$  is a set of constraints. Each constraint  $C \in C$  is defined over a set of variables  $\{x_{j_1}, \dots, x_{j_k}\}$  and specifies the allowed combinations of values for these variables.

A binary constraint  $C$  on variables  $\{x_i, x_j\}$  is *Arc Consistent* (AC) iff  $\forall a \in D(x_i) \exists b \in D(x_j)$  such that the assignments  $(x_i, a)$  and  $(x_j, b)$  are compatible. In this case we say that  $b$  is a *support* for  $a$  on constraint  $C$ . A non-binary constraint is *Generalized Arc Consistent* (GAC) iff for every variable in the constraint and each one of its values there exist compatible values in all other variables involved

in the constraint. A CSP is (G)AC iff all constraints are (G)AC and there are no empty domains.

Several consistencies have been defined for binary CSPs. Most can be described as specializations of  $(i, j)$ -consistency. A problem is  $(i, j)$ -consistent iff it has non-empty domains and any consistent instantiation of  $i$  variables can be extended to a consistent instantiation involving  $j$  additional variables [7]. Under this definition, a problem is AC iff it is  $(1, 1)$ -consistent. A problem is *path-consistent* (PC) iff it is  $(2, 1)$ -consistent. A problem is *path inverse consistent* (PIC) iff it is  $(1, 2)$ -consistent. In addition, a number of consistencies that cannot be described as  $(i, j)$ -consistencies have been defined. A problem  $P$  is *singleton arc consistent* (SAC) iff it has non-empty domains and for any instantiation  $(x, a)$  of a variable  $x$ , the resulting subproblem, denoted by  $P_{(x,a)}$ , can be made AC. A problem is *restricted path consistent* (RPC) iff any pair of instantiations  $(x, a)$ ,  $(y, b)$  of variables  $x$  and  $y$ , such that  $(y, b)$  is the only support of  $(x, a)$  on the constraint between  $x$  and  $y$ , can be consistently extended to any third variable.

Following [5], we call a consistency property  $A$  *stronger* than  $B$  iff in any problem in which  $A$  holds then  $B$  holds, and *strictly stronger* iff it is stronger and there is at least one problem in which  $B$  holds but  $A$  does not. We call a local consistency property  $A$  *incomparable* with  $B$  iff  $A$  is not stronger than  $B$  nor vice versa. Finally, we call a local consistency property  $A$  *equivalent* to  $B$  iff  $A$  is stronger than  $B$  and vice versa.

A propositional theory  $T$  is a set (conjunction) of CNF clauses of the form  $l_1 \vee l_2 \vee \dots \vee l_n$ , where each  $l_i$ ,  $1 \leq i \leq n$ , is a *literal*, ie. an atom or its negation. A clause can be alternatively denoted as  $\{l_1, l_2, \dots, l_n\}$ . Finally the notation  $x_1 x_2 \dots x_n L$  denotes the clause  $\{x_1\} \cup \{x_2\} \cup \dots \cup \{x_n\} \cup L$ . If  $c$  is a clause,  $at(c)$  denotes the set of atoms of  $c$ . We assume that the reader is familiar with the basics of propositional satisfiability.

The most common propagation method used in SAT algorithms is *Unit Propagation* (UP) that repeatedly applies *unit resolution* (UR) to the clauses of the input theory. Among stronger propagation methods, one of the earliest is the *Failed Literal Detection* rule [6] denoted as *FL rule* or simply *FL*. Given a literal  $l$  in  $T$ , s.t.  $\{\neg l\} \notin T$  and  $\{l\} \notin T$ , the FL rule assigns the value true (false) to  $l$  iff  $UP(T \cup \{\neg l\})$  ( $UP(T \cup \{l\})$ ) derives the empty clause. We call *FL-prop* the propagation scheme that repeatedly applies the FL rule until no more variable values can be inferred or the empty clause is derived.

Another class of methods that are employed in state-of-the-art SAT solvers and preprocessing algorithms is *binary resolution*, in its general or various restricted forms. Binary resolution resolves two clauses of the form  $xy$  and  $\neg xZ$  and generates the clause  $yZ$ . A restricted form of binary resolution, called BinRes, has been introduced in [1], that requires that both resolvents are binary. The application of BinRes as a propagation method, denoted by BinRes-prop, consists of repeatedly adding to the theory all new binary and unit clauses produced by resolving pairs of binary clauses and performing UP on any unit clauses that appear until nothing new is produced (or a contradiction is achieved). Note

that BinRes-prop is a weaker propagation method than FL-prop [1]. Another restricted form of binary resolution is *Krom-subsumption resolution* (*KromS*) [13] that takes as input two clauses of the form  $xy$  and  $\neg xyZ$  and generates the clause  $yZ$ . Note that  $yZ$  subsumes  $\neg xyZ$ , therefore  $\neg xyZ$  can be deleted. *Generalized Subsumption resolution* (*GSubs*) takes two clauses  $xY$  and  $\neg xYZ$  and generates  $YZ$ . The propagation methods derived by repeatedly applying KromS or GSubs are denoted by KromS-prop and GSubs-prop respectively.

### 3 Encodings

*From CSP to SAT* We restrict our study to binary CSPs. Many ways to translate a binary CSP into SAT have been proposed [10, 14, 9, 12]. We focus on two of the most widely studied ones; the direct and the support encodings.

**Direct Encoding:** In the *direct encoding* a propositional variable  $x_{ia}$  is introduced for each value  $a$  of a CSP variable  $x_i$ . For each  $x_i \in X$ , there is an at-least-one clause  $x_{i1} \vee \dots \vee x_{id}$  to ensure that  $x_i$  takes a value. We optionally add at-most-one clauses that ensure that each CSP variable takes at most one value: for each  $i, a, b$  with  $a \neq b$ , we add  $\neg x_{ia} \vee \neg x_{ib}$ . Finally, for each constraint  $C$  on variables  $\{x_i, x_j\}$  and for each  $a, b$ , s.t. tuple  $\langle (x_i, a), (x_j, b) \rangle$  is not allowed, we add  $\neg x_{ia} \vee \neg x_{jb}$ .

**Support Encoding:** The *support encoding* also introduces a propositional variable  $x_{ia}$  for each value  $a$  of a CSP variable  $x_i$ . We also have all the at-least-one clauses and (optionally) the at-most-one clauses. To capture the constraints, there are clauses that express the supports that values have in the constraints. For each binary constraint  $C$  on variables  $\{x_i, x_j\}$  and for each  $a \in D(x_i)$ , we add  $x_{jb_1} \vee \dots \vee x_{jb_s} \vee \neg x_{ia}$ , where  $x_{jb_1}, \dots, x_{jb_s}$  are the propositional variables that correspond to the  $s$  supporting values that  $a$  has in  $D(x_j)$ .

*From SAT to CSP* The following three are standard ways to translate a SAT instance into a CSP.

**Literal Encoding:** In the *literal encoding* of a SAT problem  $T$  a variable  $v_i$  is introduced for each clause  $c_i$  in  $T$ .  $D(v_i)$  consists of those literals that satisfy  $c_i$ . A binary constraint is posted between two variables  $v_i$  and  $v_j$  iff clause  $c_i$  contains a literal  $l$  whose complement is contained in clause  $c_j$ . This constraint rules out incompatible assignments for the two variables (e.g.  $(v_i, l)$  and  $(v_j, \neg l)$ ).

**Dual Encoding:** In the *dual encoding* of a SAT problem  $T$  a dual variable  $v_i$  is introduced for each clause  $c_i$  in  $T$ .  $D(v_i)$  consists of those tuples of truth values that satisfy  $c_i$ . A binary constraint is posted between any two dual variables which correspond to clauses that share propositional variables. Such a constraint ensures that shared propositional variables take the same values in the tuples of both dual variables, if they appear with the same sign in the original clauses, and complementary values if they appear with opposite signs.

**Non-Binary Encoding:** In the *non-binary encoding* of a SAT problem  $T$  there is 0-1 variable for each propositional variable. A non-binary constraint is posted between variables that occur together in a clause. This constraint disallows the tuples that fail to satisfy the clause.

## 4 Encoding SAT as CSP

### 4.1 Literal Encoding

We denote by  $L(T)$  the translation of a propositional theory  $T$  under the literal encoding. From [3, 14] we know that there is a direct correspondence between UP and AC. We now study stronger consistency levels.

From [5] we know that PIC and RPC are stronger than AC for general CSP problems. Below we provide a characterization of the cases where these consistency methods lead to domain reductions in the literal encoding of a propositional theory.

**Proposition 1.** *Value  $l$  in the domain of a variable in the literal encoding  $L(T)$  of a propositional theory  $T$  is not PIC iff  $T$  either contains the unit clause  $\neg l$  or the unit clauses  $m$  and  $\neg m$  or the clauses  $m$  and  $\neg l \vee \neg m$  or the clauses  $\neg l \vee m$  and  $\neg l \vee \neg m$ .*

*Proof.* The "if" part is straightforward. We will prove the "only if" part. Assume that the value  $l$  from the domain of variable  $v_i$ , corresponding to the clause  $c_i$  in  $T$ , is not PIC because it can not be extended to the variables  $v_j$  and  $v_k$ . If the value  $l$  has no support in any of the variables  $v_j$  or  $v_k$ ,  $T$  must contain the unit clause  $\{\neg l\}$ . Assume now that this is not the case. Suppose that the domain of variable  $v_j$  contains only the value  $m$  (ie. corresponds to a unit clause in  $T$ ). If the pair of values  $l$  and  $m$  can not be extended to variable  $v_k$ , it must be the case that the domain of  $v_k$  is either  $\{\neg m\}$ , or  $\{\neg l\}$ , or  $\{\neg l, \neg m\}$ . Consider the case now where the domain of  $v_j$  contains more than one value. Furthermore, assume that it does not contain the value  $\neg l$ . Then,  $l$  can form a consistent triple of values involving variables  $v_i$ ,  $v_j$  and  $v_k$ . Therefore, for  $l$  to be not PIC,  $v_j$  must contain the value  $\neg l$ . Moreover, if  $v_j$  has more than two values in its domain (including  $\neg l$ ) the value assignment of  $l$  to  $v_i$  can be extended to the other two variables. Therefore,  $v_j$  must have exactly two values in its domain, one of which is  $\neg l$ , ie., it is of the form  $\{\neg l, m\}$ . Since the values  $l$  and  $m$  cannot be extended to  $v_k$ , we conclude that the domain of  $v_k$  is of the form  $\{\neg l, \neg m\}$ .  $\square$

A similar result holds for RPC. The proof is similar to the above.

**Proposition 2.** *Value  $l$  in the domain of a variable in the literal encoding  $L(T)$  of a propositional theory  $T$  is not RPC iff  $T$  either contains the unit clause  $\neg l$  or the unit clauses  $m$  and  $\neg m$  or the clauses  $m$  and  $\neg l \vee \neg m$  or the clauses  $\neg l \vee m$  and  $\neg l \vee \neg m$ .*

From the above analysis we conclude that on the literal encoding of a SAT problem, PIC collapses down to RPC. Note that in general binary CSPs with more than 3 variables, PIC is strictly stronger than RPC [5]. These results also imply that BinRes-prop applied to  $T$  is at least as strong as enforcing PIC (and therefore RPC) on  $L(T)$ :

**Proposition 3.** *If enforcing PIC on the literal encoding  $L(T)$  of a propositional theory  $T$  deletes the value  $l$  from the domain of a variable of  $L(T)$  then BinRes-prop in  $T$  generates the unit clause  $\neg l$  or determines that  $T$  is unsatisfiable.*

*Proof.* Inductively on the values  $l_1, l_2, \dots, l_n$  deleted by PIC enforcement.

*Base case:* From proposition 1 follows that if  $l_1$  is not PIC, BinRes either derives  $\neg l_1$  or determines that  $T$  is unsatisfiable (in the case where  $T$  contains  $m$  and  $\neg m$ ).

*Inductive Hypothesis:* Assume that the proposition holds for all deletions  $l_i$ ,  $1 \leq i < k < n$ . That is, if PIC on  $L(T)$  deletes the values  $\{l_1, l_2, \dots, l_{k-1}\}$ , BinRes-prop on  $T$  derives the unit clauses  $\{\neg l_1, \neg l_2, \dots, \neg l_{k-1}\}$ .

*Inductive Step:* Assume that  $l_k$  is not PIC in  $L(T)$ . By analysis of the cases of proposition 1 we conclude that BinRes-prop either generates  $\neg l_k$ , or determines that  $T$  is unsatisfiable.

Assume that  $l_k$  is PIC in  $L(T)$ , but is not PIC in the problem resulting after the deletion of the values  $\{l_1, l_2, \dots, l_{k-1}\}$ . There are two cases.

First, there is a variable  $v_j$  such that all the supports of value  $l_k$  have been deleted from the domain of  $v_j$  by PIC. The domain of variable  $v_j$  is either  $\{\neg l_k, l'_1, l'_2, \dots, l'_n\}$  or  $\{l'_1, l'_2, \dots, l'_n\}$ , and  $\{l'_1, l'_2, \dots, l'_n\} \subseteq \{l_1, l_2, \dots, l_{k-1}\}$ . From the inductive hypothesis follows that BinRes-prop entails the unit clauses  $\neg l'_1, \neg l'_2, \dots, \neg l'_n$ . Then, BinRes-prop, either derives the unit clause  $\neg l_k$ , or determines that  $T$  is unsatisfiable.

Assume now that  $l_k$  has support in domain of  $v_j$  after the deletions  $\{l_1, l_2, \dots, l_{k-1}\}$  performed by PIC. This means that  $c_j$ , the clause of  $T$  that correspond to  $v_j$ , is of the form  $A \cup D \cup R$ , where  $A \subseteq \{\neg l_k\}$ ,  $D \subseteq \{l_1, l_2, \dots, l_{k-1}\}$ ,  $R = c_j - (A \cup D)$ , with  $|R| > 0$ . Assume that  $R = \{m\}$ . If  $m$  has no support in the initial domain of variable  $v_k$ , then this domain must be  $\{\neg m\}$ . Then, BinRes-prop, either derives the unit clause  $\neg l_k$ , if  $A = \{\neg l_k\}$ , or determines that  $T$  is unsatisfiable, if  $A = \emptyset$ .

Assume now that  $|R| > 1$ , and let  $R = \{m_1, m_2, \dots, m_n\}$ , with  $n \geq 2$ . Observe that all elements of  $R$  are supports for  $l_k$  in  $v_j$ . Suppose now that  $l_k$  is not PIC because none of the pairs of values  $(l_k, m_f)$ ,  $1 \leq f \leq n$ , can be extended to a consistent triple involving a value from some variable  $v_g$ . Let the domain of  $v_g$  be of the form  $B \cup S$  where  $B \subseteq \{\neg l\}$ . Note that any value of  $S$  is support for some value  $m_f$ ,  $1 \leq f \leq n$ . Since  $l_k$  is not PIC after the deletions of values  $\{l_1, l_2, \dots, l_{k-1}\}$ , we conclude that  $S \subseteq \{l_1, l_2, \dots, l_{k-1}\}$ . From the inductive hypothesis we know that BinRes-prop entails the unit clauses  $\neg l_1, \neg l_2, \dots, \neg l_{k-1}$ . Therefore, BinRes-prop will generate, using the clause that corresponds to variable  $v_g$ , either the unit clause  $\neg l_k$  or the empty clause.  $\square$

The following example shows that BinRes-prop is strictly stronger than PIC.

*Example 1.* Consider the propositional theory  $T = \{l_1 \vee l_2, \neg l_1 \vee l_3, \neg l_2 \vee l_3, \neg l_3 \vee l_4\}$ . BinRes derives the unit clause  $l_3$ , but enforcing PIC on  $L(T)$  does not lead to any domain reductions.

Not surprisingly, we can exploit the direct correspondence between AC and UP shown in [14] to prove that FL-prop on  $T$  is equivalent to enforcing SAC in  $L(T)$ . The proof proceeds by induction on the number of deletions performed by SAC and the number of assignments made by FL.

**Proposition 4.** *Enforcing SAC on the literal encoding  $L(T)$  of a propositional  $T$  deletes value  $l$  from the domains of all variables of  $L(T)$  iff FL-prop on  $T$  assigns false to  $l$ .*

## 4.2 Dual Encoding

We denote by  $D(T)$  the translation of a propositional theory  $T$  under the dual encoding. From [14] we know that AC applied to the dual encoding can achieve more propagation than UP in the original SAT instance.

As we will show, AC on the dual encoding can achieve a very strong level of consistency that cannot be captured by known propagation methods in the original SAT problem. For instance, the following example demonstrates that FL on  $T$  and AC on  $D(T)$  are incomparable.

*Example 2.* Consider the theory  $T = \{l_1 \vee l_4, \neg l_1 \vee l_2, \neg l_1 \vee l_3, \neg l_2 \vee \neg l_3\}$ . Note that FL will assign the value  $F$  to  $l_1$ . The problem  $D(T)$  is AC, therefore no domain reduction is performed by AC.

Consider now the theory  $T$  that contains all possible clauses in three variables, ie.,  $l_1 \vee l_2 \vee l_3, l_1 \vee l_2 \vee \neg l_3, \dots, \neg l_1 \vee \neg l_2 \vee \neg l_3$ . AC on  $D(T)$  will lead to a domain wipeout, whereas FL does not lead to any simplifications.

It is not difficult to show that FL on  $T$  is weaker than SAC on  $D(T)$ .

**Proposition 5.** *If FL assigns true to literal  $l$  of a propositional theory  $T$ , then all variable values of  $D(T)$  that correspond to valuations that assign false to  $l$  are not SAC.*

To precisely identify the propagation achieved by AC on the dual encoding, we first provide two characterizations of propositional theories that are not AC under the dual encoding. The first is a general characterization based on the form of the propositional theory  $T$ , whereas the second describes the form of the values that are not AC.

**Proposition 6.** *A value in the domain of a variable  $x_i$  of  $D(T)$  that corresponds to the clause  $c_i$  in  $T$  is not AC iff  $T$  contains a clause  $c_j$  such that  $at(c_j) \subseteq at(c_i)$  and there exists  $l$  such that  $l \in c_j$  and  $\neg l \in c_i$ .*

Let  $x_i$  be a variable in the dual encoding  $D(T)$  of a propositional theory  $T$ , that corresponds to a clause  $c_i$  of  $T$  defined on the set of atoms  $at(c_i) = \{a_1, a_2, \dots, a_n\}$ . We assume an order on the set of elements of  $at(c_i)$  which, if not otherwise stated, corresponds to the order of appearance of the atoms in  $c_i$ . We denote this by  $at(c_i) = (a_1, a_2, \dots, a_n)$ . We use this order to refer to the values of variable  $x_i$  of  $D(T)$  as follows. A value in the domain of  $x_i$  is a tuple  $v = (v_1, v_2, \dots, v_n)$ , where  $v_i \in \{T, F\}$  denotes the value of atom  $a_i$ , with  $1 \leq i \leq n$ . Alternatively a value for variable  $x_i$  is a tuple  $v = (l_1, l_2, \dots, l_n)$ , where  $l_i = a_i$  if  $a_i$  is assigned true in  $v$  and  $l_i = \neg a_i$  if  $a_i$  is assigned false in  $v$ .

**Proposition 7.** Value  $v = (l_1, l_2, \dots, l_n)$  in the domain of variable  $x_i$  in the dual encoding  $D(T)$  of a propositional theory  $T$  is not AC iff  $T$  contains a clause  $c_j = l'_1 \vee l'_2 \vee \dots \vee l'_m$  such that for each  $l'_k$ ,  $1 \leq k \leq m$  it holds that  $l'_k = \neg l_p$  for some  $1 \leq p \leq n$ .

Note that the above results are valid for a CSP  $D(T)$  that is the dual encoding of a propositional theory  $T$ . Enforcing AC on  $D(T)$  may lead to a sequence of domain reductions, leading to a CSP that does not necessarily correspond to the dual encoding of an associated simplification of theory  $T$ .

We now introduce *subset resolution*, a form of resolution that is stronger than GSubs, and is intended to capture the domain reductions performed when enforcing AC.

**Definition 1.** Subset resolution *resolves* two clauses  $c_i$  and  $c_j$  of a theory  $T$  iff  $T$  contains a clause  $c$  such that  $at(c_i) \subseteq at(c)$  and  $at(c_j) \subseteq at(c)$ .

We denote by *SubRes-prop* the propagation algorithm obtained by repeatedly applying subset resolution. The following result shows that SubRes-prop is at least as strong as enforcing AC.

**Proposition 8.** Let  $x_i$  be a variable in the dual encoding  $D(T)$  of a propositional theory  $T$  that corresponds to clause  $c_i$  of  $T$  such that  $at(c_i) = (a_1, a_2, \dots, a_n)$ . If enforcing AC deletes the value  $v = (l_1, l_2, \dots, l_n)$  from the domain of variable  $x_i$ , then either  $T$  contains or SubRes-prop generates the clause  $l'_1 \vee l'_2 \vee \dots \vee l'_m$  such that for each  $l'_k$ ,  $1 \leq k \leq m$  it holds that  $l'_k = \neg l_p$  for some  $1 \leq p \leq n$ .

*Proof.* By induction on the sequence of values  $v_1, v_2, \dots, v_f$  deleted by the AC enforcing algorithm.

*Base Case:* Follows from Proposition 7.

*Inductive Hypothesis:* Assume that the proposition holds for all  $v_i$ ,  $1 \leq i < k < f$ .

*Inductive Step:* Assume that the AC enforcing algorithm, after deleting the values  $v_1, v_2, \dots, v_{k-1}$ , deletes the value  $v_k$  from the domain of variable  $x_i$  of  $D(T)$  because it has no support in the domain of variable  $x_j$ , that corresponds to clause  $c_j$  in  $T$ . The case where the original domain of  $x_j$  contained no support for  $v_k$  is covered by Proposition 7. Consider now the case where value  $v_k$  has the supports  $S = \{s_1, s_2, \dots, s_r\}$  in the original domain  $x_j$ , but these supports are deleted by AC. Define  $A = at(c_i) \cap at(c_j)$ . Consider first the case where  $A = \emptyset$ . Then,  $S$  coincides with the domain of  $x_j$ , ie. it contains all possible valuations on  $at(c_j)$ , except the valuation that falsifies  $c_j$ . The fact that  $v_k$  has no support in  $x_j$  means that the domain of  $x_j$  is empty. Therefore, the AC enforcing algorithm must have been terminated, which contradicts the assumption that it deletes  $v_k$ . Hence, this situation can never arise.

Consider now the case where  $A \neq \emptyset$ , and let  $v_k^A$  be the projection of value  $v_k$  on the atoms of  $A$ , defined as  $v_k^A = (l_1, l_2, \dots, l_m)$ . The set  $S$  contains all possible valuations on the set of atoms  $at(c_j)$  that assign the same values as  $v_k$  to the variables of  $A$ , except the valuation that falsifies  $c_j$ . Since the AC enforcing

algorithm deletes value  $v_k$ , all the supports of  $v_k$  in the domain of variable  $x_j$  (ie. the elements of  $S$ ) must have been deleted by the algorithm, and therefore belong to the set of values  $v_1, v_2, \dots, v_{k-1}$ . From the inductive hypothesis we know that SubRes-prop derives a set of clauses  $R = \{c'_1, c'_2, \dots, c'_q\}$ ,  $q \leq r$ , that satisfy the properties stated in the proposition. Therefore, for each possible assignment on the atoms of  $at(c_j)$  that assigns the same value as  $v_k$  on the atoms of  $A$ , the set  $R \cup \{c_j\}$  contains a clause  $c'$  such that the assignment is not a model of  $c'$ . Hence,  $R \cup \{c_j\}$  has no models where all the atoms of  $A$  are assigned the same values as in  $v_k$ . Therefore,  $R \cup \{c_j\} \models \neg l_1 \vee \neg l_2 \vee \dots \vee \neg l_m$ . Since for each clause  $c_R$  of  $R \cup \{c_j\}$  it holds that  $at(c_R) \subseteq at(c_j)$ , SubRes-prop is able to derive, from the set of clauses  $R \cup \{c_j\}$ , a prime implicant that subsumes  $\neg l_1 \vee \neg l_2 \vee \dots \vee \neg l_m$ .  $\square$

The next result is the reciprocal of the previous proposition and both together imply that enforcing AC on  $D(T)$  is equivalent to SubRes-prop in  $T$ .

**Proposition 9.** *Given a propositional theory  $T$ , if SubRes-prop on  $T$  derives a clause  $l_1 \vee l_2 \vee \dots \vee l_m$ , then enforcing AC on  $D(T)$  deletes all values  $(l'_1, l'_2, \dots, l'_n)$  from the domains of the variables of  $D(T)$  such that for each  $l_i$ ,  $1 \leq i \leq m$  there is some  $l'_j$ ,  $1 \leq j \leq n$ , such that  $l'_j = \neg l_i$ .*

*Proof.* By induction on the set of clauses  $S = \{c_1, c_2, \dots, c_k\}$  defined inductively as follows:  $c_i \in S$  if  $c_i \in T$  or  $c_i$  is the subset resolvent of two clauses  $c_a, c_b \in S$ .

*Base Case:* The proposition follows for all clauses of  $T$  from proposition 7.

*Inductive Step:* Let  $c = c_1 \cup c_2$  be the clause that is generated by subset resolution from the clauses  $c'_1 = \{l\} \cup c_1$  and  $c'_2 = \{\neg l\} \cup c_2$ . Our inductive hypothesis is that the proposition holds for clauses  $c'_1$  and  $c'_2$ . Let  $c_1 = \{l_1^1, l_2^1, \dots, l_x^1\}$  and  $c_2 = \{l_1^2, l_2^2, \dots, l_y^2\}$ . Since  $c'_1$  and  $c'_2$  are resolved by subset resolution, it must hold that  $T$  contains a clause  $c_g$  such that  $at(c'_1) \subseteq at(c_g)$  and  $at(c'_2) \subseteq at(c_g)$ . Let  $x_g$  be the corresponding variable of  $D(T)$ . From the inductive hypothesis we know that enforcing AC deletes all values of  $x_g$  with projection  $(\neg l, \neg l_1^1, \neg l_2^1, \dots, \neg l_x^1)$  and  $(l, \neg l_1^2, \neg l_2^2, \dots, \neg l_y^2)$  on  $at(c'_1)$  and  $at(c'_2)$  respectively. Therefore,  $x_g$  can not contain a value in its domain with projection<sup>3</sup>  $(\neg l_1^1, \neg l_2^1, \dots, \neg l_x^1, \neg l_1^2, \neg l_2^2, \dots, \neg l_y^2)$  (no value at all if  $c_1 = c_2 = \emptyset$ ). Moreover, no domain of the other variables can include a value with projection  $(\neg l_1^1, \neg l_2^1, \dots, \neg l_x^1, \neg l_1^2, \neg l_2^2, \dots, \neg l_y^2)$  on the set  $at(c'_1) \cup at(c'_2)$ , as it has no support in the domain of  $x_g$ .  $\square$

Having introduced a new resolution technique to that is equivalent to AC on the dual encoding, we can also define similar methods that capture even higher consistency levels. We now introduce *extended subset resolution*, a slightly extended form of subset resolution, that is intended to capture the domain reductions performed by PIC on the dual encoding. Note that if  $c = \{l_1, l_2, \dots, l_n\}$  is a clause,  $\bar{c} = \{\neg l_1, \neg l_2, \dots, \neg l_n\}$ .

**Definition 2.** Extended Subset resolution (*ESR*) resolves two clauses  $c_i$  and  $c_j$  of a theory  $T$  if either  $T$  contains a clause  $c$  such that  $at(c_i) \subseteq at(c)$  and  $at(c_j) \subseteq$

<sup>3</sup> To facilitate the discussion we assume that  $c_1 \cap c_2 = \emptyset$ .

$at(c)$  or  $c_i \cap \bar{c}_j = \{l\}$  and  $T$  contains a clause  $c$  such that  $at(c_i) - at(\{l\}) \subseteq at(c)$  and  $at(c_j) - at(\{l\}) \subseteq at(c)$ .

We now characterize the cases where PIC performs domain reductions.

**Proposition 10.** *Value  $v = (l_1, l_2, \dots, l_n)$  in the domain of a variable  $x$  of the dual encoding  $D(T)$  of a theory  $T$  is not PIC iff  $T$  either contains a clause  $l'_1 \vee l'_2 \vee \dots \vee l'_m$  or a pair of clauses  $l'_1 \vee l'_2 \vee \dots \vee l'_a \vee l'$  and  $l'_{a+1} \vee l'_{a+2} \vee \dots \vee l'_m \vee \neg l'$  such that for each  $l'_i$ ,  $1 \leq i \leq m$  there is some  $l_j$ ,  $1 \leq j \leq n$ , such that  $l'_i = \neg l_j$ .*

Based on the above, the following result shows the relation between the clauses generated by the propagation algorithm *ESR-prop* that repeatedly applies ESR on  $T$ , and the domain reductions performed by a PIC enforcing algorithm on  $D(T)$ . The proof proceeds in a similar fashion as the proof of Proposition 9.

**Proposition 11.** *If *ESR-prop* on a propositional theory  $T$  derives a clause  $l_1 \vee l_2 \vee \dots \vee l_n$ , then enforcing PIC on  $D(T)$  deletes all values  $(l'_1, l'_2, \dots, l'_m)$  from the domains of the variables of  $D(T)$  such that for each  $l_i$ ,  $1 \leq i \leq n$  there is some  $l'_j$ ,  $1 \leq j \leq m$ , such that  $l'_j = \neg l_i$ .*

### 4.3 Non-Binary Encoding

We denote by  $N(T)$  the translation of a propositional theory  $T$  under the non-binary encoding. Note that in  $N(T)$  all clauses of  $T$  involving the same set of variables are encoded together in one constraint. From [14] we know that under the non-binary encoding, GAC is stronger than UP. We first show that if the propositional theory  $T$  does not contain clauses that are on the same variables, GAC on  $N(T)$  can do no more pruning than UP on  $T$ . The proof uses induction in the number of deletions performed by GAC.

**Proposition 12.** *Let  $T$  be a propositional theory that does not contain two clauses  $c_i$  and  $c_j$  such that  $at(c_i) = at(c_j)$ , and let  $x$  be a variable in  $T$ . If GAC deletes the value  $a$  from  $D(x)$  in  $N(T)$  then UP assigns the value  $\neg a$  to  $x$  in  $T$ .*

If the above restriction does not hold then GAC enforced on  $N(T)$  can achieve a high level of consistency. The following example shows that FL-prop, BinRes-prop, and KromS-prop are all incomparable to enforcing GAC.

*Example 3.* Let  $T$  be the theory containing all possible clauses in three variables:  $l_1 \vee l_2 \vee l_3, l_1 \vee l_2 \vee \neg l_3, \dots, \neg l_1 \vee \neg l_2 \vee \neg l_3$ . FL-prop, BinRes-prop and KromS-prop on this theory do not lead to any simplifications, whereas GAC on  $N(T)$  shows that the problem is not solvable. Note that GSubs-prop applied to  $T$  also determines insolubility.

Now consider the theory  $l_1 \vee l_4, \neg l_1 \vee l_2, \neg l_1 \vee l_3, \neg l_2 \vee \neg l_3$ . FL-prop and BinRes-prop determine that  $l_1$  must be assigned false, whereas GAC leads to no reductions. Finally, consider the theory  $l_1 \vee l_2 \vee l_3, \neg l_1 \vee l_2, \neg l_2 \vee l_3$ . KromS-prop determines that  $l_3$  must be assigned true, whereas GAC leads to no reductions.

To put an upper bound in the pruning power of GAC, we first characterize the cases where a value is not GAC in  $N(T)$ .

**Proposition 13.** *Value 0 (1) of a variable  $x_i$  in the non-binary encoding  $N(T)$  of a propositional theory  $T$  is not GAC iff  $T$  contains either the unit clause  $x_i$  ( $\neg x_i$ ) or all possible clauses in variables  $x_i, x_1, \dots, x_k$  ( $k \geq 1$ ) that include literal  $x_i$  ( $\neg x_i$ ).*

Using the above proposition, we can prove that GSubs-prop applied to  $T$  is strictly stronger than enforcing GAC on  $N(T)$ .

**Proposition 14.** *Let  $T$  be a propositional theory. If enforcing GAC deletes the value 0 (1) from the domain of a variable in  $N(T)$  then GSubs-prop assigns the value 1 (0) to the corresponding variable in  $T$ .*

*Proof.* Suppose GAC makes a sequence of value deletions  $(x_1, a_1), (x_2, a_2), \dots, (x_j, a_j)$ . The proof uses induction on  $j$ .

*Base Case:* The first value  $a_1 \in x_1$  will be deleted because it has no support in some constraint  $C$  in  $N(T)$ . From Proposition 13 we know that either  $T$  contains the unit clause  $x_1$ , if  $a_1 = 0$  ( $\neg x_1$  if  $a_1 = 1$ ), or all possible clauses in the variables of the constraint that include literal  $x_1$ , if  $a_1 = 0$ , and  $\neg x_1$  if  $a_1 = 1$ . In both cases, if we apply GSubs-prop on the clauses, we will entail  $x_1$  ( $\neg x_1$ ).

*Inductive Hypothesis:* We assume that for any  $1 \leq m < j$  the proposition holds.

*Inductive Step:* Consider the final deletion of value  $a_j$  from  $D(x_j)$ . This value is deleted because it has no supporting tuple in some constraint  $C$  on variables  $x_j, y_1, \dots, y_k$ , which corresponds to one or more clauses on the corresponding propositional variables in  $T$ . This means that for each tuple  $\tau$  that supported value  $a_j$ , at least one of the values in  $\tau$  has been deleted from a variable among  $y_1, \dots, y_k$ . According to the hypothesis, for every such deletion, the corresponding propositional variable was set to the opposite value by GSubs-prop, and the associated literals were set to false in all the associated clauses. Now consider the subset  $y_l, \dots, y_m$  of  $y_1, \dots, y_k$  which consists of the variables that have not been set. Assume that the (reduced) clauses associated with constraint  $C$  do not contain all possible combinations of literals for variables  $y_l, \dots, y_m$ . Then  $C$  will contain at least one supporting tuple for  $a_j$  which contradicts the fact that  $a_j$  is deleted. Therefore, the clauses associated with constraint  $C$  contain all possible combinations of literals for variables  $y_l, \dots, y_m$ . If we apply GSubs-prop on these clauses, we will entail  $x_j$ , if  $a_j = 0$ , and  $\neg x_j$  if  $a_j = 1$ .  $\square$

From the above proposition and the last problem in Example 3, it follows that GSubs-prop is strictly stronger than enforcing GAC.

## 5 Encoding CSP as SAT

### 5.1 Direct Encoding

We denote by  $Di(P)$  the translation of a CSP  $P$  into SAT under the direct encoding. From [14] we know that AC enforced on a CSP  $P$  can achieve more pruning than UP applied to  $Di(P)$ .

The following example demonstrates that FL-prop is incomparable to PIC while BinRes-prop is incomparable to AC, RPC, and PIC.

*Example 4.* Consider a CSP with four variables  $x_1, x_2, x_3, x_4$ , all with  $\{0,1\}$  domains, and constraints  $x_1 = x_2, x_2 = x_3, x_3 = x_4$  and  $x_4 \neq x_1$ . This problem is AC, RPC, and PIC. However, by setting  $x_{10}$  or  $x_{11}$  to true in the direct encoding, UP generates the empty clause. Therefore, FL-prop sets both  $x_{10}$  and  $x_{11}$  to false and determines that the problem is unsatisfiable. Accordingly, BinRes-prop generates the unit clauses  $\neg x_{10}$  and  $\neg x_{11}$  and therefore determines unsatisfiability.

Now consider a CSP with three variables  $x_1, x_2, x_3$ , all with  $\{0,1,2,3\}$  domain, and three constraints. Assume that values 0 and 1 (2 and 3) of  $x_1$  are supported by 0 and 1 (2 and 3) in  $D(x_2)$  and  $D(x_3)$ , and that values 0 and 1 (2 and 3) of  $D(x_2)$  are supported by 2 and 3 (0 and 1) in  $D(x_3)$ . FL-prop on the direct encoding of the problem does not generate the empty clause by setting any propositional variable to true or false. Therefore it does not achieve any inference. However, the problem is not PIC.

Finally, consider a CSP with two variables  $x_1$  and  $x_2$ , both with three values in their domains, where one of the values in  $D(x_1)$  has no support in  $D(x_2)$ . AC (and all stronger CSP consistencies) will delete this value. However, BinRes-prop cannot resolve any clauses and therefore infers nothing.

We now show that if a value in a CSP is not RPC then the FL rule will set the corresponding variable to false in  $Di(P)$ .

**Proposition 15.** *If a value  $a \in D(x_i)$  of a CSP  $P$  is not RPC then the FL rule assigns  $x_{ia}$  to false when applied to  $Di(P)$ .*

*Proof.* Assume that value  $a \in D(x_i)$  of a CSP  $P$  is not RPC. This means that  $(x_i, a)$  has a single support (say  $b \in D(x_j)$ ) in the constraint between  $x_i$  and  $x_j$  and the assignments  $(x_i, a)$  and  $(x_j, b)$  cannot be consistently extended to some third variable  $x_l$ . That is, there is no value in  $D(x_l)$  that supports both  $(x_i, a)$  and  $(x_j, b)$ . Now consider the direct encoding of the problem and assume that FL sets  $x_{ia}$  to true. UP will immediately set each  $x_{jb'}$ , where  $b' \in D(x_j)$  and  $b' \neq b$ , to false, and as a result  $x_{jb}$  will be set to true. UP will then set all propositional variables corresponding to the values in  $D(x_l)$  to false. Therefore, an empty clause is generated and as a result  $x_{ia}$  will be set to false.  $\square$

A corollary of the above proposition is that FL sets the corresponding propositional variable to false for any value of the original CSP that is not AC. The first problem in Example 4 together with Proposition 15 can help prove that FL-prop is strictly stronger than RPC and AC. The complete proof involves induction in the number of deletions performed by the algorithms. We now state that if FL-prop makes an inference in  $Di(P)$  then SAC also makes the corresponding inference in  $P$ .

**Proposition 16.** *If FL-prop sets a propositional variable  $x_{ia}$  to false in the direct encoding of a CSP  $P$  then SAC deletes value  $a$  from  $D(x_i)$  in  $P$ . If FL-prop sets  $x_{ia}$  to true then SAC deletes all values in  $D(x_i)$  apart from  $a$ .*

Now consider the second problem in Example 4. This problem is not SAC, but FL-prop applied to its direct encoding infers nothing. This, together with the above proposition, prove that SAC is strictly stronger than FL-prop, and therefore also BinRes-prop.

We now show that GSubs applied to  $Di(P)$  can do no more work than Unit Resolution.

**Proposition 17.** *Given a CSP  $P$ , GSubs applied to  $Di(P)$  draws an inference iff only Unit Resolution does.*

*Proof.* It suffices to show that GSubs can only be applied to pairs of clauses of the form  $x, \neg xY$ . Consider two clauses of the form  $xY, \neg xYC$ . First assume that both clauses are non-binary. In this case both clauses are necessarily at-least-one clauses. But if  $x$  is present in one of the clauses,  $\neg x$  cannot be present in the other, since the two clauses encode domains of different variables. Now assume that at least one of the clauses is of the form  $xy$ . There are three cases depending on what kind of clause  $xy$  is. If  $xy$  is a conflict clause then  $\neg x$  can only be found in an at-least-one clause. However, literal  $y$  cannot be present in this clause. If  $xy$  is an at-most-one clause then  $\neg x$  can again only be found in the corresponding at-least-one clause. However, this clause will contain  $\neg y$  and not  $y$ . Finally, if  $xy$  is an at-least-one clause then  $\neg x$  and  $y$  cannot occur together in any conflict or at-most-one clause. Hence, in all cases GSubs cannot be applied.  $\square$

## 5.2 Support Encoding

We denote by  $Sup(P)$  the translation of a CSP  $P$  into SAT under the support encoding. From [10, 9] we know that AC applied to a CSP  $P$  is equivalent to UP applied to  $Sup(P)$ . We now elaborate on the relationship between FL-prop and SAC.

**Proposition 18.** *FL-prop sets a propositional variable  $x_{ia}$  to false in the support encoding of a CSP  $P$  iff value  $a$  is deleted by SAC in  $P$ . FL-prop sets  $x_{ia}$  to true iff SAC deletes all values in  $D(x_i)$  apart from  $a$ .*

The proof is rather simple and proceeds by induction in the number of assignments made by FL-prop and the number of value deletions performed by SAC, exploiting the equivalence between AC and UP.

It is easy to see that BinRes-prop is strictly stronger than AC. Consider a problem with three variables  $x, y, z$ , each with domain  $\{0, 1\}$  and constraints  $x = y$ ,  $x = z$  and  $y \neq z$ . This problem is AC, but BinRes-prop applied to its support encoding shows that it is inconsistent. This example, coupled with the fact that BinRes-prop subsumes UP (which is equivalent to AC), proves that BinRes-prop is strictly stronger than AC.

The following example demonstrates that BinRes-prop is incomparable to RPC, and PIC.

*Example 5.* Consider the first problem in Example 4. This problem is AC, RPC, and PIC. However, BinRes-prop generates the unit clauses  $\neg x_{10}$  and  $\neg x_{11}$  and therefore determines unsatisfiability.

Now consider a CSP with three variables  $x_1, x_2, x_3$ , all with  $\{0, 1, 2, 3\}$  domain, and three constraints. Assume that value 0 of  $x_1$  is supported by 0 in  $D(x_2)$  and 0,1 in  $D(x_3)$ , and that value 0 of  $D(x_2)$  is supported by 2,3 in  $D(x_3)$ . All other values have at least two supports in any constraint. The problem is not RPC (or PIC), but BinRes-prop applied to the support encoding does not detect the inconsistency.

As in the direct encoding, GSubs applied to  $Sup(P)$  can do no more work than Unit Resolution. The proof proceeds by case analysis and is similar to the proof of Proposition 17.

**Proposition 19.** *Given a CSP  $P$ , GSubs applied to  $Sup(P)$  draws an inference iff only Unit Resolution does.*

## 6 Conclusion

In this paper we presented theoretical results concerning the relative propagation power of various local consistency methods for CSP and SAT. More specifically, we studied AC, PIC, RPC and SAC for CSP, and FL, BinRes, KromS and GSubs for SAT. The results we obtained complement and tie together recent similar studies [3, 9, 14].

As it may be expected, in cases where AC is equivalent to Unit Propagation, SAC is equivalent to Failed Literal Detection. Under both translations of CSP to SAT we consider, FL can achieve a level of consistency that is higher than RPC in the original problem. Among the less powerful methods, that are nevertheless stronger than AC, BinRes arises as an appealing method that can achieve a relative high level of local consistency. Indeed, BinRes is stronger than PIC under the literal encoding, and stronger than AC under the support encoding. Generalized subsumption resolution finally achieves an intermediate level of consistency between GAC in  $N(T)$  and AC in  $D(T)$ .

Comparing among different encodings, in the case of translating a SAT to CSP, the dual encoding appears to achieve the highest level of local consistency among the different approaches studied. Indeed, AC in the dual encoding corresponds to subset resolution, a method that is stronger than generalized subsumption resolution. Although the cost of applying such a local consistency method in its general form may be prohibitive, restricted versions (eg. restricting the number of literals in the clauses considered) of the method may have some practical value. The non-binary encoding appears weaker than the dual, and enforcing GAC on the translated problem can achieve better propagation than UP only if the original theory contains clauses on the same variables. Finally, under the literal encoding, techniques that have been used in SAT and CSP are roughly equivalent, with the exception of BinRes which achieves a level

of consistency between PIC and SAC. When translating from CSP to SAT, the direct encoding appears to be rather weak. Indeed, strong resolution methods such as generalized subsumption resolution are weaker than AC in the original problem. The support encoding appears to behave better propagation-wise, as already suggested in [9]. A general conclusion that can be drawn from our analysis is that translating a CSP to SAT can be beneficial only if the support encoding is used, coupled with a propagation method that is at least as strong as BinRes.

An extended version of this paper contains a more detailed study of propagation in CSP and SAT, including CSP consistency methods such as Neighbor Inverse Consistency and SAT techniques such as Hyper-resolution. In the future we intend to extend our study to cover encodings of non-binary constraints into SAT, and also consider the relationship between learning techniques in CSP and SAT under the various encodings. Finally, but very importantly, an empirical evaluation of the different encodings and propagation methods is required.

## References

1. F. Bacchus. Enhancing Davis Putnam with Extended Binary Clause Reasoning. In *Proceedings of AAAI-02*, pages 613–619, 2002.
2. H. Benameur. The satisfiability problem regarded as a constraint satisfaction problem. In *Proceedings of ECAI-1996*, pages 125–130, 1996.
3. H. Benameur. A Comparison between SAT and CSP Techniques. *Constraints*, 9:123–138, 2004.
4. C. Bessière, E. Hebrard, and T. Walsh. Local Consistencies in SAT. In *Proceedings of SAT-2003*, pages 400–407, 2003.
5. R. Debruyne and C. Bessière. Domain Filtering Consistencies. *Journal of Artificial Intelligence Research*, 14:205–230, 2001.
6. J.W. Freeman. *Improvements to Propositional Satisfiability Search Algorithms*. PhD thesis, 1995.
7. E. Freuder. A Sufficient Condition for Backtrack-bounded Search. *JACM*, 32(4):755–761, 1985.
8. E. Freuder and C. Elfe. Neighborhood Inverse Consistency Preprocessing. In *Proceedings of AAAI'96*, pages 202–208, 1996.
9. I. Gent. Arc Consistency in SAT. In *Proceedings of ECAI-2002*, pages 121–125, 2002.
10. S. Kasif. On the Parallel Complexity of Discrete Relaxation in Constraint Satisfaction Networks. *Artificial Intelligence*, 45(3):275–286, 1990.
11. Chu Min Li and Anbulagan. Heuristics based on unit propagation for satisfiability problems. In *Proceedings of IJCAI-97*, pages 366–371, 1997.
12. S. Prestwich. Full Dynamic Substitutability by SAT Encoding. In *Proceedings of CP-2004*, pages 512–526, 2004.
13. A. van Gelder and Y. Tsuji. Satisfiability testing with more reasoning and less guessing. In *Cliques, Coloring and Satisfiability*, pages 559–586, 1996.
14. T. Walsh. SAT v CSP. In *Proceedings of CP-2000*, pages 441–456, 2000.