Recovering and Exploiting Structural Knowledge from CNF Formulas

Richard Ostrowski, Éric Grégoire, Bertrand Mazure, and Lakhdar Saïs

CRIL CNRS – Université d'Artois
rue Jean Souvraz SP-18, F-62307 Lens Cedex France
{ostrowski,gregoire,mazure,sais}@cril.univ-artois.fr

Abstract. In this paper, a new pre-processing step is proposed in the resolution of SAT instances, that recovers and exploits structural knowledge that is hidden in the CNF. It delivers an hybrid formula made of clauses together with a set of equations of the form $y = f(x_1, \ldots, x_n)$ where f is a standard connective operator among ($\vee$, $\wedge$, $\Leftrightarrow$) and where y and x_i are boolean variables of the initial SAT instance. This set of equations is then exploited to eliminate clauses and variables, while preserving satisfiability. These extraction and simplification techniques allowed us to implement a new SAT solver that proves to be the most efficient current one w.r.t. several important classes of instances.

Keywords: SAT, Boolean logic, propositional reasoning and search

1 Introduction

Recent impressive progress in the practical resolution of hard and large SAT instances allows real-world problems that are encoded in propositional clausal normal form (CNF) to be addressed (see e.g. [20,10,27]). While there remains a strong competition about building more efficient provers dedicated to hard random k-SAT instances [8], there is also a real surge of interest in implementing powerful systems that solve difficult large real-world SAT problems. Many benchmarks have been proposed and regular competitions (e.g. [6,1,22,23]) are organized around these specific SAT instances, which are expected to encode structural knowledge, at least to some extent.

Clearly, encoding knowledge under the form of a conjunction of propositional clauses can flatten some structural knowledge that would be more apparent in a full propositional logic representation, and that could prove useful in the resolution step [21,12].

In this paper, a new pre-processing step is proposed in the resolution of SAT instances, that extracts and exploits some structural knowledge that is hidden in the CNF. It delivers an hybrid formula made of clauses together with a set of equations of the form $y = f(x_1, \ldots, x_n)$ where f is a standard connective operator among $\{\vee, \wedge, \Leftrightarrow\}$ and where y and x_i are Boolean variables of the initial SAT instance. Such an hybrid formula exhibits a twofold interest. On the one hand, the structural knowledge in the equations could be exploited by the SAT

P. Van Hentenryck (Ed.): CP 2002, LNCS 2470, pp. 185–199, 2002.

solver. On the other hand, these equations can allow us to determine equivalent variables and implied ones, in such a way that clauses and variables can be eliminated, while preserving satisfiability. These extraction and simplification techniques allowed us to implement a new SAT solver that proves to be the most efficient current one w.r.t. several important classes of instances.

The paper is organized as follows. After some preliminary definitions, it is shown how such a kind of equations can be extracted from the CNF, using a graph of clauses. Then, the task of simplifying the set of clauses using these equations is addressed. Experimental results showing the efficiency of the proposed approach are provided. Finally, promising paths of research are discussed in the conclusion.

2 Technical Preliminaries

Let $\mathcal{L}$ be a Boolean (i.e. propositional) language of formulas built in the standard way, using usual connectives ($\vee$, $\wedge$, $\neg$, $\Rightarrow$, $\Leftrightarrow$) and a set of propositional variables. A *CNF formula* is a set (interpreted as a conjunction) of *clauses*, where a clause is a disjunction of *literals*. A literal is a positive or negated propositional variable. An *interpretation* of a Boolean formula is an assignment of truth values $\{true, false\}$ to its variables. A *model* of a formula is an interpretation that satisfies the formula. Accordingly, SAT consists in finding a model of a CNF formula when such a model does exist or in proving that such a model does not exist. Let c_1 be a clause containing a literal a and c_2 a clause containing the opposite literal $\neg a$, one *resolvent* of c_1 and c_2 is the disjunction of all literals of c_1 and c_2 less a and $\neg a$. A resolvent is called *tautological* when it contains opposite literals. Let us recall here that any Boolean formula can be translated thanks to a linear time algorithm in CNF, equivalent with respect to SAT (but that can use additional propositional variables). Most satisfiability checking algorithms operate on clauses, where the structural knowledge of the initial formulas is thus flattened.

Other useful definitions are the following ones. An *equation* or *gate* is of the form $y = f(x_1, \ldots, x_n)$ where f is a standard connective among $\{\vee, \wedge, \Leftrightarrow\}$ and where y and x_i are propositional variables. An equation is satisfied iff the left and right hand sides of the equation are simultanousely $true$ or $false$. An interpretation of a set of equations is a model of this set iff it satisfies each equation of this set.

The first technical goal of this paper is to extract gates from a CNF formula. A propositional variable y (resp. $x_1, \ldots, x_n$) is an *output variable* (resp. are *input variables*) of a gate of the form $y = f(x_1, \ldots, x_n)$. An output variable is also called *definable*.

A propositional variable z is an *output variable of a set of gates* iff z is an output variable of at least one gate in the set. An *input variable of a set of gates* is an input variable of a gate which is not an output variable of the set of gates.

Clearly, the truth-value of an y output variable depends on the truth value of the x_i input variables of its gate. Moreover, the set of definable variables of a CNF formula is a subset of the so-called *dependent* variables as defined in [15]. Knowing output variables can play an important role in solving the consistency status of a CNF formula. Indeed, the truth value of such variables can be obtained by propagation, and *e.g.* they can be omitted by selection heuristics of DPLL-like algorithms [4]. In the general case, knowing n' output variables of a CNF formula using n variables allows the size of the set of interpretations to be investigated to decrease from 2^n to $2^{n-n'}$.

Unfortunately, extracting gates from a CNF formula can be a time-consuming operation in the general case, unless some depth-limited search resources or heuristic criteria are provided. Indeed, showing that $y = f(x_1, \ldots, x_i)$ (where $y, x_1, \ldots, x_i$ belong to Σ), follows from a given CNF Σ, is coNP-complete [15].

3 Gates Extraction

To the best of our knowledge, only equivalent gates were subject of previous investigation. Motivated by Selman et-al. challenge [24] about solving the parity-32 problems, Warners and van Maaren [26] have proposed an approach that succeeds in solving such class of hard CNF formulas. More precisely, a two steps algorithm is proposed: in the first one, a polynomially solvable subproblem (a set of equivalent gates) is identified thanks to a linear programming approach. Using the solution to this subproblem, the search-space is dramatically restricted for the second step of the algorithm, which is an extension of the well-known DPLL procedure [4]. More recently, Chu Min Li [18] proposed a specialised DPLL procedure called `EqSatz` which dynamically search for lists of equivalent literals, lists whose length is lower or equal to 3. Such an approach is costly as it performs many useless syntactical tests and suffers from restrictions (*e.g.* on the length of the detected lists).

In this paper, in order to detect hidden gates in the CNF formula, it is proposed to make use of an original concept of partial graph of clauses to limit the number of syntactical tests to be performed. Moreover, this technique allows gates $y = f(x_1, \ldots, x_n)$ (where $f \in \{\Leftrightarrow, \vee, \wedge\}$ and where no restriction on n is *a priori* given) to be detected.

Definition 1 (Graph of clauses)
Let Σ be a CNF formula. A **graph of clauses** $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ *is associated to Σ s.t.*

- *each vertex of $\mathcal{V}$ corresponds to a clause of Σ;*
- *each edge (c_1, c_2) of $\mathcal{E}$ corresponds to a pair of clauses c_1 and c_2 of Σ exhibiting a resolvent clause;*
- *each edge is labeled either by $\mathbb{T}$ (when the resolvent is tautological) or $\mathbb{R}$ (when the resolvent is not tautological).*

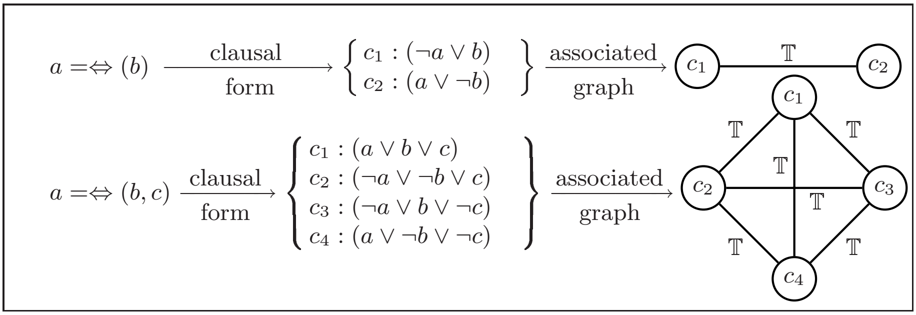
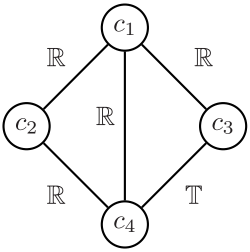

$a = \Leftrightarrow (b)$
clausal form
$c_1 : (\neg a \vee b)$
$c_2 : (a \vee \neg b)$
associated graph
c_1
$\mathbb{T}$
c_2
$a = \Leftrightarrow (b, c)$
clausal form
$c_1 : (a \vee b \vee c)$
$c_2 : (\neg a \vee \neg b \vee c)$
$c_3 : (\neg a \vee b \vee \neg c)$
$c_4 : (a \vee \neg b \vee \neg c)$
associated graph
c_1
c_2
c_3
c_4
$\mathbb{T}$

Fig. 1. Clausal and graphical representations of ⇔ gates

Example 1 (Graph of clauses)

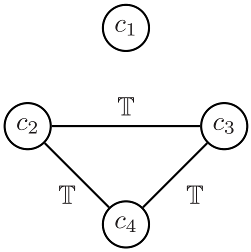
$$\Sigma = \left\{ \begin{array}{l} c_1 : (a \vee b \vee \neg c) \\ c_2 : (\neg a \vee b \vee d) \\ c_3 : (\neg a \vee b \vee \neg c) \\ c_4 : (a \vee c \vee \neg e) \end{array} \right\} \xrightarrow{\text{graph of clauses}} \mathcal{G} =$$



Definition 2 (Partial graph of clauses)
A **partial graph of clauses** *$\mathcal{G}'$ of a CNF formula Σ is the graph of clauses $\mathcal{G}$ of Σ that is restricted to edges labelled by $\mathbb{T}$.*

Example 2 (Partial graph of clauses)

$$\Sigma = \left\{ \begin{array}{l} c_1 : (a \vee d \vee e) \\ c_2 : (\neg a \vee \neg b \vee \neg c) \\ c_3 : (a \vee b \vee \neg c) \\ c_4 : (\neg a \vee b \vee c) \end{array} \right\} \xrightarrow[\text{of clauses}]{\text{partial graph}} \mathcal{G}' =$$



In Figure 1, both graphical and clausal representations of an equivalence gate $y = \Leftrightarrow (x_1, \ldots, x_n)$ ($n = 1$ and $n = 2$) are given. In the general case an equivalence gate will be represented by a partial graph that is a clique since any pair of clauses gives rise to a tautological resolvent.

In Figure 2, both graphical and clausal representations of gates $a = \wedge(b, c, d)$ and $a = \vee(b, c, d)$ are provided.

Let us note that graphical representations of gates $\vee$ and $\wedge$ are identical since their clausal representations only differ by the variables signs. We also note that one clause plays a pivotal role and exhibits tautological resolvents with all clauses in the clausal representation of a $\vee$ or $\wedge$ gate. This property also applies for gates whose number of involved literals is greater than 3:

$$a = \wedge(b,c,d) \xrightarrow[\text{form}]{\text{clausal}} \left\{ \begin{array}{l} c_1 : (a \vee \neg b \vee \neg c \vee \neg d) \\ c_2 : (\neg a \vee b) \\ c_3 : (\neg a \vee c) \\ c_4 : (\neg a \vee d) \end{array} \right\}$$

$$a = \vee(b,c,d) \xrightarrow[\text{form}]{\text{clausal}} \left\{ \begin{array}{l} c_1 : (\neg a \vee b \vee c \vee d) \\ c_2 : (a \vee \neg b) \\ c_3 : (a \vee \neg c) \\ c_4 : (a \vee \neg d) \end{array} \right\} \xrightarrow[\text{graph}]{\text{associated}}$$

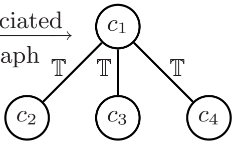


Fig. 2. Clausal and graphical representations of $\wedge$ and $\vee$ gates

$$y = \vee(x_1,\ldots,x_n) \xrightarrow[\text{form}]{\text{clausal}} \left\{ \begin{array}{l} (\neg y \vee x_1 \cdots \vee x_n) \\ (y \vee \neg x_1) \\ \cdots \\ (y \vee \neg x_n) \end{array} \right\}$$

It is also easy to check that any resolvent from any pair of clauses from a same gate is tautological. Accordingly, the clauses from a same gate ($\Leftrightarrow$, $\vee$ and $\wedge$) are connected in the partial graph of clauses. Thus, a necessary (but not sufficient) condition for clauses to belong to a same gate is to form a connected subgraph. An example showing that such a condition is not sufficient is given in Example 2.

Building the graph of clauses is quadratic in the size of the set of clauses Σ but the representation of the graph can be too much space-consuming. Accordingly, finding gates will be performed in a dynamic manner (without representing the graph explicitly) by checking for each clause c which clauses of Σ exhibit tautological resolvents with c. This step can be achieved using two stacks:

- a stack of clauses sharing literals with c;
- a stack of clauses containing opposite literals to the literals of c.

At this step, the initial CNF formula is reduced to an hybrid formula, *i.e.* a set of equations together with initial clauses not taking part in these equations. For the uniformity of the representation each such remaining clause $c : (x_1 \vee \cdots \vee x_n)$ can be interpreted as a or gate of the form $true = \vee(x_1,\ldots,x_n)$ with its output variable assigned the value *true* (but we shall sometimes still call them clauses, indifferently).

4 Exploiting Structural Knowledge

In this section, it is shown how such an obtained representation can even be simplified, and how its intrinsic properties can lead to more efficient satisfiability checking algorithms (by eliminating variables and clauses).

First of all, the n' output variables of the set of gates are clearly fixed by the remaining input variables. Accordingly, DPLL-like satisfiability algorithms [4] can restrict their search to input variables only.

Moreover, in the following it is shown how some properties of gates can allow even more variables and equations to be eliminated.

4.1 Equivalence Gates

First, let us recall that when only equivalence gates are involved, then they can be solved in polynomial time [9]. These classes of formulas are known as chains of biconditionals. In the following, some basic properties of equivalence gates are presented. For more details, see, Dunham and Wang's paper [9]. For commodity, we use chains of biconditionals instead of equivalences gates.

Property 1 (about $\Leftrightarrow$ gates [9])

1. *$\Leftrightarrow$ is commutative and associative.*
2. *$(a \Leftrightarrow a \Leftrightarrow B)$ (resp. $(\neg a \Leftrightarrow a \Leftrightarrow B)$) with B a chain of biconditionnals is equivalent to B (resp $\neg B$).*
3. *$\neg(a \Leftrightarrow b \Leftrightarrow c)$ is equivalent to $(\neg a \Leftrightarrow b \Leftrightarrow c)$*
4. *$(\neg a \Leftrightarrow \neg b \Leftrightarrow \neg c)$ is equivalent to $(\neg a \Leftrightarrow b \Leftrightarrow c)$.*
5. *$(l \Leftrightarrow A_1), (l \Leftrightarrow A_2), \ldots, (l \Leftrightarrow A_m)$ is SAT iff $(A_1 \Leftrightarrow A_2), \ldots, (A_{m-1} \Leftrightarrow A_m)$ is SAT.*

It is easy to see that the first four equivalence gates properties apply on hybrid formulas.

As a consequence of the first property, any variable in an equivalence gate can play the role of the output variable of this gate. Currently, we have selected a very simple way for choosing output variables of equivalence gates. An output variable of an equivalence gate is selected among the set of output variables already defined for other gates. When the intersection of this set and the set of variables of the equivalence gate is empty, the output variable is selected in a random way in the set of variables involved in the equivalence gate. Properties 1.2. and 1.5. can lead to the elimination of variables and thus to a reduction of the search space. Property 1.4. shows that negation can be eliminated in pairs in chains of biconditionals (i.e. at most one literal of a chain of biconditionals is a negative one).

Let us now give new simplification properties of equivalent gates in the context of hybrid formulas (set of gates).

Property 2

Let Σ be a set of gates (i.e. an hybrid formula), $B \subset \Sigma$ a set of equivalence gates, $b \in B$ s.t. its output variable y occurs only in B and Σ' the set of gates obtained by the substitution of y with its definition and removing b from Σ, then Σ is satisfiable iff Σ' is satisfiable

Remark 1

The previous property is a simple extension of property 1.5 to set of gates.

Property 3
Let Σ be a set of gates, any equivalence gate of Σ containing a literal which does not occur elsewere in Σ, can be removed from Σ without loss of satisfiability.

Consequently, each literal in an equivalence gate must occur at least twice in the formula.

4.2 "And" & "or" Gates

In the case of $\vee$ and $\wedge$ gates, the following property can be used to achieve useful simplifications.

Property 4 ($\vee$ and $\wedge$ gates)

- $a = f(b, c, b)$ *with* $f \in \{\vee, \wedge\}$ *is equivalent to* $a = f(b, c)$
- $a = \vee(b, c, \neg b)$ *(resp.* $a = \wedge(b, c, \neg b)$*) is equivalent to* a *(resp.* $\neg a$*)*
- $\neg a = \vee(b, c, d)$ *(resp.* $\neg a = \wedge(b, c, d)$*) is equivalent to* $a = \wedge(\neg b, \neg c, \neg d)$ *(resp.* $a = \vee(\neg b, \neg c, \neg d)$*)*
- *Property 2 and 3 hold for* $\vee$ *and* $\wedge$ *gates.*

4.3 Simplification of the Remaining Set of Clauses

Let Γ be the remaining set of clauses (that can be interpreted as $true = \vee(x_1, \ldots, x_n)$ equations). Many practical approaches to SAT focus on the reduction of the number of variables of the instance to be solved, in order to reduce the size of the search space. In this paper, we also try to reduce the number of involved clauses. As we shall show it, this can lead to the elimination of variables, too. Interestingly, reducing the number of clauses and variables involved in Γ, leads to a reduction of the number of input variables.

In the following, two types of simplification are proposed. The first one is derived from an extension of the definition of blocked clauses, as proposed in [13,14,9]. The second one takes its roots in the pre-processing step of many efficient implementations of the DPLL algorithm [4]: the introduction of constant-length resolvents in a C-SAT-like spirit [7]. Let us note that, we can use other useful simplification techniques (see for example recent works by Brafman [3] and Marques-Silva [19]) to achieve further reduction on the CNF part of the formula.

Generalization of the Blocked Clause Concept

Definition 3 (Blocked clause [13])
A clause c of a CNF formula Σ is **blocked** *iff there is a literal $l \in c$ s.t. for all $c' \in \Sigma$ with $\neg l \in c'$ the resolvent of c and c' is tautological.*

From a computational point of view, a useful property attached to the concept of blocked clause is the following one.

Property 5 (Blocked clause [13])
Let c be a clause belonging to a CNF formula Σ s.t. c is blocked. Σ is satisfiable iff $\Sigma \backslash \{c\}$ is satisfiable.

Example 3 (Blocked clause)
The following clause c_1 is blocked by the literal a.

$$\Sigma = \left\{\begin{array}{l} c_1 : (a \vee b \vee c) \\ c_2 : (\neg a \vee \neg b) \\ c_3 : (\neg b \vee c) \\ c_4 : (b \vee \neg c) \end{array}\right\} \text{ is SAT iff } \Sigma \backslash \{c_1\} = \left\{\begin{array}{l} c_2 : (\neg a \vee \neg b) \\ c_3 : (\neg b \vee c) \\ c_4 : (b \vee \neg c) \end{array}\right\} \text{ is SAT}$$

The concept of blocked clause can be generalized as follows, using the definition of non-fundamental clause.

Definition 4 (Non-fundamental clause)
A clause c belonging to a CNF formula Σ is **non-fundamental** *iff c is either tautological or is subsumed by another clause from Σ.*

From this, the concept of blocked clause is extended to *nf-blocked* clause.

Definition 5 (nf-blocked clause)
A clause c belonging to a CNF formula Σ is **nf-blocked** *iff there exists a literal l from c s.t. there does not exist any resolvent in l, or s.t. all resolvents are not fundamental.*

Property 5 can be extended to nf-blocked clauses.

Property 6 (nf-blocked clause)
Let c be a clause belonging to a CNF formula Σ s.t. c is nf-blocked. Σ is satifiable iff $\Sigma \backslash \{c\}$ is satisfiable.

Corollary 1
Blocked clauses and clauses containing a pure literal are nf-blocked.

The following example illustrates how the elimination of clauses can allow the consistency of a CNF formula to be proved.

Example 4 (nf-blocked)

$$\left\{\begin{array}{l} c_1 : (a \vee b \vee c) \\ c_2 : (\neg a \vee b \vee d) \\ c_3 : (b \vee c \vee d) \\ c_4 : (\neg b \vee c \vee \neg d) \\ c_5 : (a \vee b \vee \neg c) \end{array}\right\} \xrightarrow[\text{nf-blocked clause}]{c_1 \text{ nf-blocked by } a} \left\{\begin{array}{l} c_2 : (\neg a \vee b \vee d) \\ c_3 : (b \vee c \vee d) \\ c_4 : (\neg b \vee c \vee \neg d) \\ c_5 : (a \vee b \vee \neg c) \end{array}\right\}$$

$$\left\{\begin{array}{l} c_2 : (\neg a \vee b \vee d) \\ c_3 : (b \vee c \vee d) \\ c_4 : (\neg b \vee c \vee \neg d) \\ c_5 : (a \vee b \vee \neg c) \end{array}\right\} \xrightarrow[\text{blocked clause}]{c_2 \text{ nf-blocked by } b} \left\{\begin{array}{l} c_3 : (b \vee c \vee d) \\ c_4 : (\neg b \vee c \vee \neg d) \\ c_5 : (a \vee b \vee \neg c) \end{array}\right\}$$

$$\left\{\begin{array}{l} c_3 : (b \vee c \vee d)\\ c_4 : (\neg b \vee c \vee \neg d)\\ c_5 : (a \vee b \vee \neg c)\end{array}\right\} \xrightarrow[\text{blocked clause}]{c_3 \text{ nf-blocked by } d} \left\{\begin{array}{l} c_4 : (\neg b \vee c \vee \neg d)\\ c_5 : (a \vee b \vee \neg c)\end{array}\right\}$$

$$\left\{\begin{array}{l} c_4 : (\neg b \vee c \vee \neg d)\\ c_5 : (a \vee b \vee \neg c)\end{array}\right\} \xrightarrow[\text{pure literal}]{c_4 \text{ nf-blocked by } d} \left\{ c_5 : (a \vee b \vee \neg c) \right\}$$

$$\left\{ c_5 : (a \vee b \vee \neg c) \right\} \xrightarrow[\text{pure literal}]{c_5 \text{ nf-blocked by } a} \text{SAT}$$

As it can be done when pure literals are involved, this technique can lead to the elimination of variables. Indeed, it is always possible to nf-block a clause. To this end, we just have to add to the CNF all resolvents of the clause w.r.t. a given literal of this clause.

Property 7
Any clause c from a CNF formula Σ can be nf-blocked, introducing additional clauses in Σ.

In order to eliminate a variable, we just have to nf-block all clauses where it occurs. From a practical point of view, such a technique should be limited to variables giving rise to a minimum number of resolvents (*e.g.* variables which do not occur often). This idea is close to the elimination technique proposed in [5] and has been revisited in [25].

More generally, a concept of *redundant clause* can be defined as follows.

Definition 6 (Redundant clause [2])
A clause c belonging to a CNF formula Σ is **redundant** *iff $\Sigma \backslash \{c\} \models c$.*

From a practical computational point of view, looking for redundant clauses amounts to proving that $\Sigma \wedge \neg c$ is inconsistent. Accordingly, it should not be searched for such clauses in the general case. However, it is possible to limit the search effort, *e.g.* by looking for implicates clauses or literals by unit propagation [16].

Definition 7 (u-redundant clause)
A clause c from a CNF formula Σ is **u-redundant** *iff the unsatisfiability of $\Sigma \wedge \neg c$ can be obtained using unit propagation, only (i.e. $\Sigma \backslash \{c\} \underset{Unit}{\models} c$).*

Clearly, the set of u-redundant clauses of a CNF formula is a subset of the set of redundant clauses of this formula. Using Example 4, the relationship between both nf-blocked and u-redandant clauses can be illustrated :

- nf-blocked clauses can be non u-redundant. (See clause c_1 in Example 4)
- u-redundant clauses can be non nf-blocked. (In the same example, if the initial CNF formula is extended with a clause $c_6 : (a \vee \neg d)$ and $c_7 : (\neg a \vee \neg d)$, then clause c_1 becomes u-redundant but is not nf-blocked anymore).
- Clauses can be u-redundant and nf-blocked at the same time. (In the same example, extending the initial CNF formula with both clauses $c'_6 : (b \vee c)$ and $c'_7 : (b \vee \neg c)$, clause c_1 remains nf-blocked and becomes u-redundant)

Limited Form of Resolution Many recent efficient implementations of DPLL [4] contain a preprocessing step introducing limited-length resolvents (the maximal length being generally fixed to 2), which increases the performance of the solver. However, the number of resolvents possibly introduced in this way can be prohibitive. Accordingly, we propose to limit the introduction of clauses to resolvents allowing clauses to be eliminated.

Definition 8 (Subsuming resolvent)
Let Σ be a CNF formula, a **subsuming resolvent** *is a resolvent from two clauses from Σ that subsumes at least one clause of Σ.*

Taking subsuming resolvents into account entails at least two direct useful consequences from a computational point of view. First, the subsuming resolvent is a shorter clause. Indeed, a subsuming clause is shorter than the subsumed one. From a practical point of view, we just need to eliminate one or some literals from the subsumed clause to get the subsuming one. Secondly, the elimination of such literals in the clause can lead to the suppression of a variable, or make it a unit literal or a pure literal. In all three cases, the search space is clearly reduced accordingly.

Example 5
Clauses $(a \vee b \vee c)$ and $(a \vee \neg c)$ generate the resolvent $(a \vee b)$, which subsumes the ternary clause and allows the literal c to be eliminated.

5 Implementation and Experimental Results

In this section, some preliminary -but significant- experimental results are presented. All algorithms have been programmed in C under Linux. All experimentations have been conducted using a 1 Ghz Pentium III processor, with 256 MB RAM, under Mandrake Linux 8.2.

Before we implemented the solver, we addressed the *a priori* feasibility of the equations extraction technique, at least w.r.t. standard benchmarks. Indeed, although it is naturally expected that gates do exist in such benchmarks, these gates have never been exhibited. Moreover, despite the fact that the use of the partial graphs limits the number of syntactical tests to be performed, the extraction technique could appear too much time-consuming from a practical point of view.

The results given in Table 1 answer these issues for benchmarks from the last SAT competitions [6,1,22,23]. For every tested instance, we have listed:

- the number of clauses ($\#C$) and of variables ($\#V$) of the initial instance;
- the number of discovered gates, using two separate categories: equivalence ($\# \Leftrightarrow$) and $\vee$ and $\wedge$ gates ($\# \vee \wedge$);
- the size of the set Γ of remaining clauses ($\#C_{\Gamma}$ & $\#V_{\Gamma}$);
- the time spent by the extraction process.

Table 1. Number of extracted equations and time spent for the extraction

instance	$\#C$	$\#V$	$\# \Leftrightarrow$	$\# \vee \wedge$	$\#C_{\Gamma}$	$\#V_{\Gamma}$	time(s)
par8-1-c	254	64	56	15	30	31	0.00
par8-1	1149	350	135	15	30	31	0.07
par16-1-c	1264	317	270	61	184	124	0.08
par16-1	3310	1015	560	61	184	124	0.25
par32-1-c	5254	1315	1158	186	622	375	0.38
par32-1	10277	3176	2261	186	622	375	0.64
barrel5	5383	1407	1065	152	1163	430	0.3
barrel6	8931	2306	1746	254	2013	821	0.53
barrel7	13765	3523	2667	394	3195	1337	0.96
barrel8	20083	5106	3864	578	4763	2158	1.80
ssa7552-125	3523	1512	1033	154	1270	501	0.33
ssa2670-130	3321	1359	859	254	1352	530	0.26
ssa0432-001	1027	435	225	43	244	124	0.1
bf1355-348	7271	2286	1082	383	3533	962	0.46
dubois100	800	300	200	0	0	0	0.05
2dlx_cc_mc_ex_bp_f2_bug091	55424	5259	0	4053	7575	5214	4.50
dlx1_c	1592	295	0	209	139	291	0.01
dlx2_cc_bug18	19868	2047	0	1567	1312	2039	0.92
dlx2_cc	12812	1516	0	1063	1137	1508	0.39
1dlx_c_mc_ex_bp_f	3725	776	0	542	378	755	0.05
2dlx_ca_mc_ex_bp_f	24640	3250	0	2418	1627	3223	0.94
2dlx_cc_mc_ex_bp_f	41704	4583	0	3534	2159	4538	2.88

The results from Table 1 are really promising since they show that there exist many gates in many classes of benchmarks and that the time spent to find them is negligible (far less than 1 second, including the time spent to load the instance). Moreover, the size of the set Γ of remaining clauses after the extraction process is reduced in a significant manner (on average, the number of clauses is divided by a factor ranging from 2 to 10) and can even be zero for certain types of instances (*e.g.* `Dubois100`).

However, the set of variables from Γ and of the equations are not disjoint. We thus then focused on determining the number of variables that are really non defined, i.e. the variables that are never output ones. Table 2 provides the number of non defined variables (or input variables $\#V_{nd}$) for several instances, notably for "parity" instances. These instances were selected because solving them is recognized as a challenge in the research community about SAT [24]. The results are quite surprising since only 32 variables are not defined w.r.t. the 3176 ones in the `par32-1` instance. This means that the truth value of the 3144 other variables depends only on these 32 variables obtained by the extraction technique. Accordingly, the search space is reduced from 2^{3176} to 2^{32} !

These abstraction and simplification techniques have been grafted as a preprocessing step to the DPLL procedure [4], using a branching heuristics *à la* Jeroslow-Wang [11]. This algorithm, called **LSAT** runs a DPLL-like algorithm on Γ and checks during the search process if the current interpretation being built does not contradict any detected gate. In the positive case, a backtrack step is performed. This new algorithm has been compared with the last versions

Table 2. Number of undefinable variables

instance	$\#C$	$\#V$	$\#V_{\Gamma}$	$\#V_{nd}$	$time(s)$
par8-1	1149	350	31	8	0.00
par16-1	3310	1015	124	16	0.05
par32-1	10277	3176	375	32	0.10
ssa0432-001	1027	437	106	63	0.00
ssa2670-140	3201	1327	444	196	0.01
ssa7552-001	3614	1534	408	246	0.02
bf2670-001	3434	1393	439	210	0.02
bf1355-160	7305	2297	866	526	0.06
bf0432-001	3668	1040	386	294	0.02
2dlx_cc_mc_ex_bp_f2_bug091	55424	5259	5214	1170	4.45
dlx1_c	1592	295	291	82	0.01
dlx2_cc_bug18	19868	2047	2039	477	0.92
dlx2_cc	12812	1516	1508	448	0.38
1dlx_c_mc_ex_bp_f	3725	776	755	214	0.0
2dlx_ca_mc_ex_bp_f	24640	3250	3223	807	0.97
2dlx_cc_mc_ex_bp_f	41704	4583	4538	1012	2.87

of the most efficient SAT solvers, namely `Satz` [17], `EqSatz` [18], `Zchaff` [27]. The obtained results are given in Table 3 (time is given in seconds)[1].

These results show that LSAT is really more efficient that those solvers for many instances. Moreover, LSAT solves some instances in less than 1 second, whereas the other solvers took more than 16 minutes to give an answer.

6 Future Work

This work opens promising paths for future research. Indeed, the current version of the LSAT solver is just a basic prototype that runs a DPLL procedure on the remaining clauses and checks that the current interpretation does not contradict the other equations. Clearly, such a basic prototype can be improved in several directions. First, it would be interesting to develop DPLL-specific branching heuristics that take all the equations into account (and not only the remaining clauses). It would also be interesting to explore how the algorithm could exploit the intrinsic properties of each type of equation.
In this paper, the simplification process of $\wedge$, $\vee$ gates and clauses has been described, but not yet implemented in the current LSAT version. On many classes of formulas (*e.g.* formal verification instances) containing a large part of such gates, we attempt further improvements using such simplification properties. More generally, it might be useful to extend this work to the simplification and resolution of general Boolean formulas. Finally, this work suggests that to model real-world problems, one might directly use more general and extended boolean formulas.

[1] In the table, $> n$ means that the instance could not be solved within n seconds.

Table 3. Comparison of LSAT, Satz, EqSatz and Zchaff

instance	$\#C$	$\#V$	SAT	Satz	EqSatz	Zchaff	**LSAT**
par8-1	1149	350	yes	0.05	0.01	0.01	**0.01**
par8-2	1149	350	yes	0.04	0.01	0.01	**0.01**
par8-3	1149	350	yes	0.07	0.01	0.01	**0.01**
par8-4	1149	350	yes	0.09	0.01	0.01	**0.01**
par8-5	1149	350	yes	0.05	0.01	0.01	**0.01**
par16-1	3310	1015	yes	8.96	0.19	0.47	**0.05**
par16-2	3310	1015	yes	0.48	0.20	0.88	**0.05**
par16-3	3310	1015	yes	16.79	0.22	4.07	**0.02**
par16-4	3310	1015	yes	11.15	0.17	0.82	**0.06**
par16-5	3310	1015	yes	1.59	0.18	0.41	**0.03**
par32-1-c	5254	1315	yes	¿1000	540	¿1000	**6**
par32-2-c	5254	1315	yes	¿1000	24	¿1000	**28**
par32-3-c	5254	1315	yes	¿1000	1891	¿1000	**429**
par32-4-c	5254	1315	yes	¿1000	377	¿1000	**16**
par32-5-c	5254	1315	yes	¿1000	4411	¿1000	**401**
par32-1	10227	3176	yes	¿1000	471	¿1000	**27**
par32-2	10227	3176	yes	¿1000	114	¿1000	**7**
par32-3	10227	3176	yes	¿1000	4237	¿1000	**266**
par32-4	10227	3176	yes	¿1000	394	¿1000	**3**
par32-5	10227	3176	yes	¿1000	5645	¿1000	**471**
barrel5	5383	1407	no	86	0.38	1.67	**0.19**
barrel6	8931	2306	no	853	0.71	8.29	**0.55**
barrel7	13765	3523	no	¿1000	0.96	21.55	**6.23**
barrel8	20083	5106	no	¿1000	1.54	53.76	**412**
dubois10	80	30	no	0.03	0.08	0.01	**0.01**
dubois20	160	60	no	26.53	0.03	0.01	**0.01**
dubois30	240	90	no	¿1000	0.05	0.01	**0.01**
dubois50	400	150	no	¿1000	0.08	0.01	**0.01**
dubois100	800	300	no	¿1000	0.06	0.06	**0.01**
Urquhart3	578	49	no	¿1000	¿1000	190	**0.02**
Urquhart4	764	81	no	¿1000	¿1000	¿1000	**0.03**
Urquhart5	1172	125	no	¿1000	¿1000	¿1000	**0.06**
Urquhart15	11514	1143	no	¿1000	¿1000	¿1000	**0.42**
Urquhart20	18528	1985	no	¿1000	¿1000	¿1000	**0.64**
Urquhart25	29670	3122	no	¿1000	¿1000	¿1000	**1.05**

7 Conclusion

In this paper, a technique of extraction of equations of the form $y = f(x_1, \ldots, x_n)$ with $f \in \{\vee, \wedge, \Leftrightarrow\}$ from a CNF formula has been presented. This extraction technique allows us to rewrite the CNF formula under the form of a conjunction of equations. These equations classify variables into defined ones and undefined ones. The defined variables can be interpreted as the output of the logical gates discovered by the extraction process, and allow us to reduce the search space in a significant way very often. Another contribution of this paper was the introduction of various simplification techniques of the remaining equations. In their turn, these latter techniques allow us to eliminate variables, reducing the search space again. These new techniques of extraction and simplification have been grafted as a pre-processing step of a new solver for SAT: namely, LSAT. This solver proves extremely competitive w.r.t. the best current techniques for several classes of structured benchmarks.

Acknowledgements

We are grateful to the anonymous referees for their comments on the previous version of this paper. This work has been supported in part by the CNRS, the “Conseil Régional du Nord/Pas-de-Calais”, by the EC under a FEDER program, the “IUT de Lens” and the “Université d'Artois”.

References

1. First international competition and symposium on satisfiability testing, March 1996. Beijing (China). 185, 194
2. Yacine Boufkhad and Olivier Roussel. Redundancy in random sat formulas. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI'00)*, pages 273–278, 2000. 193
3. Ronen I. Brafman. A simplifier for propositional formulas with many binary clauses. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI'01)*, 2001. 191
4. M. Davis, G. Logemann, and D. Loveland. A machine program for theorem proving. *Journal of the Association for Computing Machinery*, 5:394–397, 1962. 187, 190, 191, 194, 195
5. Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *Journal of the Association for Computing Machinery*, 7:201–215, 1960. 193
6. Second Challenge on Satisfiability Testing organized by the Center for Discrete Mathematics and Computer Science of Rutgers University, 1993. `http://dimacs.rutgers.edu/Challenges/`. 185, 194
7. Olivier Dubois, Pascal André, Yacine Boufkhad, and Jacques Carlier. Sat versus unsat. In D. S. Johnson and M. A. Trick, editors, *Second DIMACS Challenge*, DIMACS Series in Discrete Mathematics and Theorical Computer Science, American Mathematical Society, pages 415–436, 1996. 191
8. Olivier Dubois and Gilles Dequen. A backbone-search heuristic for efficient solving of hard 3–sat formulae. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI'01)*, volume 1, pages 248–253, Seattle, Washington (USA), August 4–10 2001. 185
9. B. Dunham and H. Wang. Towards feasible solution of the tautology problem. *Annals of Mathematical Logic*, 10:117–154, 1976. 190, 191
10. E. Giunchiglia, M. Maratea, A. Tacchella, and D. Zambonin. Evaluating search heuristics and optimization techniques in propositional satisfiability. In *Proceedings of International Joint Conference on Automated Reasoning (IJCAR'01)*, Siena, June 2001. 185
11. Robert G. Jeroslow and Jinchang Wang. Solving propositional satisfiability problems. *Annals of Mathematics and Artificial Intelligence*, 1:167–187, 1990. 195
12. Henry A. Kautz, David McAllester, and Bart Selman. Exploiting variable dependency in local search. In *Abstract appears in "Abstracts of the Poster Sessions of IJCAI-97"*, Nagoya (Japan), 1997. 185
13. Oliver Kullmann. Worst-case analysis, 3-sat decision and lower bounds: Approaches for improved sat algorithms. In *DIMACS Proceedings SAT Workshop*, DIMACS Series in Discrete Mathematics and Theorical Computer Science, American Mathematical Society, 1996. 191, 192

14. Oliver Kullmann. New methods for 3-sat decision and worst-case analysis. *Theoretical Computer Science*, pages 1–72, 1997. 191
15. Jérome Lang and Pierre Marquis. Complexity results for independence and definability in propositional logic. In *Proceedings of the Sixth International Conference on Principles of Knowledge Representation and Reasoning (KR'98)*, pages 356–367, Trento, 1998. 187
16. Daniel Le Berre. Exploiting the real power of unit propagation lookahead. In *Proceedings of the Workshop on Theory and Applications of Satisfiability Testing (SAT2001)*, Boston University, Massachusetts, USA, June 14th-15th 2001. 193
17. Chu Min Li and Anbulagan. Heuristics based on unit propagation for satisfiability problems. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI'97)*, pages 366–371, Nagoya (Japan), August 1997. 196
18. C. M. Li. Integrating equivalency reasoning into davis-putnam procedure. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI'00)*, pages 291–296, 2000. 187, 196
19. Joao P. Marques-Silva. Algebraic simplification techniques for propositional satisfiability. In *Proceedings of the 6th International Conference on Principles and Practice of Constraint Programming (CP'2000)*, September 2000. 191
20. Shtrichman Oler. Tuning sat checkers for bounded model checking. In *Proceedings of Computer Aided Verification (CAV'00)*, 2000. 185
21. Antoine Rauzy, Lakhdar Saïs, and Laure Brisoux. Calcul propositionnel : vers une extension du formalisme. In *Actes des Cinquièmes Journées Nationales sur la Résolution Pratique de Problèmes NP-complets (JNPC'99)*, pages 189–198, Lyon, 1999. 185
22. Workshop on theory and applications of satisfiability testing, 2001. http://www.cs.washington.edu/homes/kautz/sat2001/. 185, 194
23. Fifth international symposium on the theory and applications of satisfiability testing, May 2002. http://gauss.ececs.uc.edu/Conferences/SAT2002/. 185, 194
24. Bart Selman, Henry A. Kautz, and David A. McAllester. Computational challenges in propositional reasoning and search. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI'97)*, volume 1, pages 50–54, Nagoya (Japan), August 1997. 187, 195
25. A. Van Gelder. Extracting (easily) checkable proofs from a satisfiability solver that employs both preorder and postorder resolution. *Annals of Mathematics and Artificial Intelligence*, 2002. to appear. 193
26. Joost P. Warners and Hans van Maaren. A two phase algorithm for solving a class of hard satisfiability problems. *Operations Research Letters*, 23(3–5):81–88, 1999. 187
27. L. Zhang, C. Madigan, M. Moskewicz, and S. Malik. Efficient conflict driven learning in a boolean satisfiability solver. In *Proceedigns of ICCAD'2001*, pages 279–285, San Jose, CA (USA), November 2001. 185, 196