Actes JNPC'02

Techniques de simplification de formules CNF

Richard Ostrowski **Éric Grégoire** **Bertrand Mazure**

CRIL (CNRS FRE2499) – Université d'Artois
Faculté Jean Perrin – Rue Jean Souvraz
SP-18 F62307, Lens Cedex France
{ostrowski,gregoire,mazure}@cril.univ-artois.fr

Lakhdar Saïs

IRIT – Université Paul Sabatier
118 route de Narbonne
31062 Toulouse Cedex 4
sais@irit.fr

Résumé

Dans cet article, nous proposons un nouveau pré-traitement pour les méthodes de résolution de SAT. Ce pré-traitement permet d'identifier une partie des informations structurelles masquées par le codage sous forme normale conjonctive. Plus précisément, nous cherchons à exhiber, à partir des clauses de la formule, des équations de la forme : $y = f(x_1, x_2, \ldots, x_n)$ où f est un connecteur logique ($\vee$, $\wedge$, $\Leftrightarrow$, ...) et où y et x_i sont des variables propositionnelles de l'instance originale. La formule hybride obtenue après ce premier pré-traitement est constituée d'une conjonction d'équations et de clauses. Dans un deuxième temps, nous nous sommes intéressés à simplifier l'ensemble de clauses restantes. Nous proposons à cet effet, différentes techniques permettant de supprimer des clauses et des variables tout en préservant la satisfaisabilité. L'ensemble des techniques d'extraction et de simplification est implanté dans un nouvel algorithme de résolution qui donne d'excellents résultats comparativement aux meilleurs solveurs SAT connus.

Mots clés : Représentation des connaissances, logique propositionnelle, SAT.

1 Introduction

Les progrès effectués ces dernières années dans la résolution pratique du problème SAT permettent d'envisager aujourd'hui la résolution d'instances issues d'applications réelles. En effet, les instances k-SAT aléatoires au seuil ne sont plus aujourd'hui les seules « *benchmarks* » de référence, preuve en est les compétitions organisées autour de la résolution pratique de SAT [5, 1, 16, 17] qui proposent de nombreuses instances dites structurées.

Il reste que la représentation sous forme normale conjonctive (CNF) de problèmes issus d'applications réelles ne permet pas une représentation simple et compréhensible des connaissances, alors qu'à l'inverse la logique propositionnelle lorsqu'elle est utilisée dans sa totalité permet de représenter simplement une grande variété de connaissances. En effet, de nombreux problèmes se modélisent naturellement par des fonctions booléennes générales. La transformation sous forme clausale fait perdre de nombreuses informations pertinentes (*e.g.* structurelles) qui peuvent s'avérer importantes pour la résolution.

Dans cet article, nous proposons un nouveau pré-traitement pour les méthodes de résolution de SAT. Ce pré-traitement permet d'extraire une partie des informations structurelles masquées par le codage sous forme normale conjonctive. Plus précisément, nous cherchons à exhiber, à partir des clauses de la formule, des équations de la forme : $y = f(x_1, x_2, \ldots, x_n)$ où f est un connecteur logique ($\vee$, $\wedge$, $\Leftrightarrow$, ...) et où y et x_i sont des variables propositionnelles de l'instance originale. Cette recherche présente un double intérêt : le premier est d'offrir la possibilité au solveur d'exploiter les propriétés intrinsèques de chaque équation ; le second est d'identifier des variables dont la valeur de vérité est toujours définie en fonction d'autres variables. La formule hybride obtenue après ce premier pré-traitement est constituée d'une conjonction d'équations et de clauses. Dans un deuxième temps, nous nous sommes intéressés à simplifier l'ensemble de clauses restantes. Nous proposons à cet effet, différentes techniques permettant de supprimer des clauses et des variables tout en préservant la satisfaisabilité.

Dans la première partie, nous énonçons quelques définitions. Dans la deuxième partie, nous discutons de l'extraction d'équations, à partir de la CNF. Nous montrons notamment comment il est possible d'exploiter un graphe de clauses à cette fin. La troisième partie est dédiée à la simplification de l'ensemble Γ des clauses restantes après l'extraction des équations. La quatrième partie est dédiée à la présentation des résultats expérimentaux obtenus montrant l'efficacité de notre approche. Quelques remarques et pistes de recherche sont avancées en conclusion.

2 Définitions

Un des objectifs de ce travail est d'extraire à partir d'une formule CNF des équations de la forme : $y = f(x_1, x_2, \ldots, x_n)$ où f est un connecteur logique ($\vee$, $\wedge$, $\Leftrightarrow$, ...) et où y et x_i sont des variables propositionnelles de la CNF.

Défintion 2.1 (Équations ou portes[1])
On appelle **équation** *ou* **porte** *une formule propositionnelle s'écrivant :* $y = f(x_1, x_2, \ldots, x_n)$ *avec* f *un connecteur logique (*$\vee$*,* $\wedge$*,* $\Leftrightarrow$*) et* y *et* x_i *des variables propositionnelles.*

[1] Nous empruntons cette définition à [15].

Défintion 2.2 (Variable de sortie ou définissable)
Une variable propositionnelle y est appelée **variable de sortie** *d'une porte, si elle s'écrit $y = f(x_1, x_2, \ldots, x_n)$. On dit que cette variable est* **définissable**. *Plus généralement, on appelle variables définissables d'une CNF, l'ensemble des variables de sortie des portes extraites de la CNF.*

La valeur de vérité d'une variable de sortie est dépendante de celles des variables x_i. En effet, l'affectation des variables x_i aura pour effet d'attribuer, par propagation, une valeur à y. L'ensemble des variables définissables d'une CNF est un sous-ensemble de l'ensemble des variables dépendantes tel que défini dans [11]. Par ailleurs, cet ensemble peut jouer un rôle capital dans la résolution de la CNF. En effet, si l'on est capable de déterminer l'ensemble des variables définissables, ces variables peuvent être exclues de l'heuristique de choix de variables d'un DPLL [3], puisque ces variables pourront être affectées par propagation de valeurs. Plus précisément, pour une CNF de n variables, si l'on est capable d'identifier n' variables de sorties, alors la complexité diminue de $2^{n'}$ (elle passe de 2^n à $2^{n-n'}$).

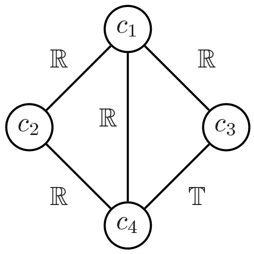
L'extraction de portes à partir d'une CNF est une tâche fastidieuse et consommatrice de temps CPU si l'on ne limite pas la recherche ou si l'on ne dispose pas de critères permettant d'élire des clauses susceptibles d'appartenir à une même porte. En section 3, nous montrons comment il possible d'exploiter un *graphe de clauses* pour exhiber les variables et les clauses d'une CNF pouvant former une équation.

Défintion 2.3 (Graphe de clauses)
Pour une CNF Σ, le **graphe de clauses** $\mathcal{G} = (\mathcal{C}, \mathcal{R})$ *associé est tel que :*

- *l'ensemble $\mathcal{C}$ des sommets de $\mathcal{G}$ correspond à l'ensemble des clauses de Σ ;*
- *il existe une arête (c_1, c_2) si et seulement s'il existe une résolvante entre les clauses c_1 et c_2 ; l'ensemble des arêtes de $\mathcal{G}$ est noté $\mathcal{R}$;*
- *chaque arête est étiquetée : $\mathbb{T}$ si la résolvante est tautologique, $\mathbb{R}$ sinon.*

Exemple 2.1 (Graphe de clauses)

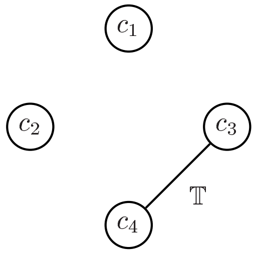
$$\Sigma = \left\{ \begin{array}{c} c_1 = (a \vee b \vee \neg c) \\ c_2 = (\neg a \vee b \vee d) \\ c_3 = (\neg a \vee b \vee \neg c) \\ c_4 = (a \vee c \vee \neg e) \end{array} \right\} \xrightarrow[\text{associé}]{\text{Graphe de clauses}} \mathcal{G} =$$



Défintion 2.4 (Graphe partiel de clauses)
On appelle **graphe partiel de clauses** $\mathcal{G}'$ *d'une formule CNF Σ, le graphe de clauses $\mathcal{G}$ de Σ restreint aux arêtes étiquetées $\mathbb{T}$.*

Exemple 2.2 (Graphe partiel de clauses)

$$\Sigma = \left\{ \begin{array}{c} c_1 = (a \vee b \vee \neg c) \\ c_2 = (\neg a \vee b \vee d) \\ c_3 = (\neg a \vee b \vee \neg c) \\ c_4 = (a \vee c \vee \neg e) \end{array} \right\} \xrightarrow[\text{de clauses associé}]{\text{Graphe partiel}} \mathcal{G}' =$$



3 Extraction d'équations

Étant donnée une formule CNF, différentes approches peuvent être envisagées pour détecter les portes. L'approche « directe » consiste à effectuer une recherche symbolique des fonctions. Cette approche est utilisée dans le solveur `EqSatz` [14], où l'auteur se limite à la recherche de chaînes d'équivalences de taille inférieure ou égale à 3. Cette approche entraîne beaucoup de tests syntaxiques inutiles et impose des restrictions (*e.g.* sur la longueur des chaînes d'équivalences détectées).

Nous proposons d'utiliser le graphe partiel de clauses précédemment défini (cf. Déf. 2.4) afin de limiter le nombre de tests syntaxiques effectués. L'objectif est ici de caractériser les portes en fonction de leur représentation graphique. Notons que nous nous intéressons à la reconnaissance de portes $y = f(x_1, x_2, \ldots, x_n)$ où $f \in \{\Leftrightarrow, \vee, \wedge\}$ et n quelconque (> 0).

La figure 1 illustre la représentation clausale et graphique d'une porte $y = \Leftrightarrow (x_1, x_2, \ldots, x_n)$ pour $n = 1$ et $n = 2$. Plus généralement, une chaîne d'équivalences se représente par un graphe partiel de clauses complet, ce qui signifie que toutes les clauses intervenant dans le codage de la porte se résolvent tautologiquement.

La figure 2 illustre la représentation clausale et graphique des portes $a = \wedge(b, c, d)$ et $a = \vee(b, c, d)$.

$$(a \Leftrightarrow b) \xrightarrow[\text{clausale}]{\text{Forme}} \left\{ \begin{array}{l} c_1 = (\neg a \vee b) \\ c_2 = (a \vee \neg b) \end{array} \right\} \xrightarrow[\text{associé}]{\text{Graphe}}$$

$$(a \Leftrightarrow b \Leftrightarrow c) \xrightarrow[\text{clausale}]{\text{Forme}} \left\{ \begin{array}{l} c_1 = (a \vee b \vee c) \\ c_2 = (\neg a \vee \neg b \vee c) \\ c_3 = (\neg a \vee b \vee \neg c) \\ c_4 = (a \vee \neg b \vee \neg c) \end{array} \right\} \xrightarrow[\text{associé}]{\text{Graphe}}$$

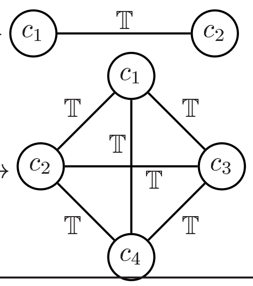


FIG. 1 – Représentation clausale et graphique d'une porte $\Leftrightarrow$

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

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

$$\xrightarrow[\text{associé}]{\text{Graphe}}$$

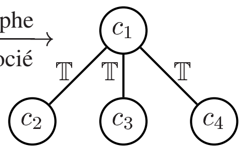


FIG. 2 – Représentation clausale et graphique des portes $\wedge$ et $\vee$

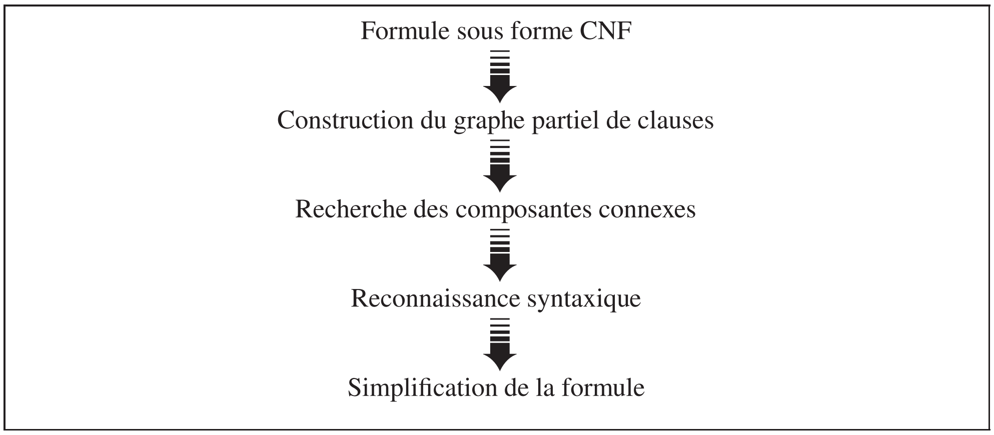


FIG. 3 – Algorithme d'extraction de portes

Les représentations graphiques des portes $\vee$ et $\wedge$ sont identiques. En effet, les formes clausales ne diffèrent que par le signe des variables. Ceci est une conséquence directe du fait que : $(a = \vee(b, c, d)) \equiv (\neg a = \wedge(\neg b, \neg c, \neg d))$. Par ailleurs, nous constatons qu'une clause joue un rôle pivot et possède des résolvantes tautologiques avec toutes les autres clauses intervenant dans la représentation clausale d'une porte $\vee$ ou $\wedge$. Ceci se généralise pour une taille de porte supérieure à 3. En effet, une représentation clausale de $y = \vee(x_1, x_2, \ldots, x_n)$ est :

$$\left\{ \begin{array}{l} (\neg y \vee x_1 \vee x_2 \vee \cdots \vee x_n) \\ (y \vee \neg x_1) \\ (y \vee \neg x_2) \\ \ldots \\ (y \vee \neg x_n) \end{array} \right\}$$

Remarque 3.1
Dans une formule CNF, toutes les résolvantes produites à partir de clauses intervenant dans une même équation sont tautologiques.

Par conséquent, les clauses appartenant à une même porte ($\Leftrightarrow$, $\vee$ et $\wedge$) sont connectées dans le graphe partiel de clauses. Ainsi, pour identifier les clauses susceptibles d'appartenir à une même porte, il suffit à partir du graphe partiel de clauses d'isoler les composantes connexes. Cependant cette condition est nécessaire mais pas suffisante. En effet, des clauses peuvent être en résolvantes tautologiques sans pour autant appartenir à une même porte. L'exemple 2.2 l'illustre. Par conséquent, la reconnaissance syntaxique est nécessaire, cependant le nombre de tests inutiles (car infructueux) est fortement réduit.

Un algorithme d'extraction de portes est donné dans la Fig. 3. Cependant, en pratique construire le graphe de clauses même partiel est trop coûteux (en espace comme en temps). La reconnaissance se fait donc dynamiquement en identifiant, pour chaque clause c, les clauses de la CNF admettant des résolvantes tautologiques avec c. Cette étape peut être réalisée très simplement à l'aide de deux piles :

– une pile de clauses ayant des littéraux communs avec c ;

– une pile de clauses ayant des littéraux opposés avec c.

La dernière phase de l'algorithme présenté est la simplification de la formule. Cette simplification ne consiste pas uniquement à remplacer les clauses appartenant à une même porte par leur définition sous forme d'équations. En effet, il est possible d'exploiter certaines propriétés des portes afin de supprimer des variables.

Propriété 3.1 (des portes $\Leftrightarrow$[7])

1. *$\Leftrightarrow$ est commutatif et associatif.*
2. *$(a \Leftrightarrow a \Leftrightarrow b)$ est équivalent à b.*
3. *$(\neg a \Leftrightarrow \neg b \Leftrightarrow \neg c)$ est équivalent à $(\neg a \Leftrightarrow b \Leftrightarrow c)$.*
4. *$(l \Leftrightarrow A_1), (l \Leftrightarrow A_2), \ldots, (l \Leftrightarrow A_m)$ est SAT si et seulement si $(A_1 \Leftrightarrow A_2), \ldots, (A_{m-1} \Leftrightarrow A_m)$ est SAT.*

La conséquence de la première propriété est que dans la définition d'une porte $\Leftrightarrow$ n'importe quelle variable peut jouer le rôle de variable de sortie. Les propriétés 3.1.2. et 3.1.4. peuvent conduire à la suppression de variables et donc réduire l'espace de recherche lors de la résolution. La propriété 3.1.3. s'énonce plus généralement de la manière suivante : il est possible de supprimer par paire les négations dans une chaîne d'équivalences.

Remarque 3.2

Des propriétés 3.1.2. et 3.1.3., il découle que : $(a \Leftrightarrow \neg a \Leftrightarrow b)$ est équivalent à $\neg b$.

En ce qui concerne les portes $\vee$ et $\wedge$, la propriété de fusion peut également conduire à supprimer une variable.

Propriété 3.2 (fusion des portes $\vee$ et $\wedge$[2])

– $\{a = \vee(b, c)$ *et* $d = \vee(a, e)\} \models d = \vee(b, c, e)$
– $\{a = \wedge(b, c)$ *et* $d = \wedge(a, e)\} \models d = \wedge(b, c, e)$

Une fois ces simplifications opérées, nous obtenons une formule hybride contenant un ensemble d'équations et un ensemble de clauses. Nous sommes intéressés à la simplification de l'ensemble Γ de clauses restantes.

4 Simplification du Γ

Le but de cette simplification est de réduire au maximum la taille de l'ensemble de clauses restantes. Contrairement aux simplifications usuelles exploitées dans le cadre de la résolution pratique de SAT, nous ne limitons pas cette simplification à un réduction du nombre de variables. Il n'est pas à faire la preuve de l'utilité de diminuer le nombre de variables : plus le nombre de variables est petit, plus l'espace des interprétations à parcourir est petit. Néanmoins l'élimination de clauses peut conduire à la suppression de variables comme nous le montrons ci-après.

Nous proposons deux nouveaux types de simplification. Le premier est une extension de la définition de clauses bloquées proposée dans [9, 10]. Le second s'inspire d'un pré-traitement

[2]Dans cette propriété, le symbole $\models$ représente la conséquence logique.

effectué dans de nombreuses implantations de l'algorithme DPLL [3] : l'ajout de résolvantes de taille inférieure ou égale à une constante. À notre connaissance, le premier algorithme à avoir utilisé ce type de pré-traitement est C-SAT [6].

4.1 Généralisation de la notion de clauses bloquées

Nous rappelons ci-dessous la définition d'une clause bloquée.

Défintion 4.1 (Clause bloquée [9])
Une clause c d'une CNF Σ est dite **bloquée** *s'il existe un littéral l de c tel que toutes les résolvantes en l de c sont tautologiques.*

L'exploitation qui peut être faite des clauses bloquées résulte de la propriété suivante.

Propriété 4.1 (Clause bloquée [9])
Soit c une clause d'une CNF Σ. Si c est bloquée alors Σ est satisfiable si et seulement si $\Sigma\backslash\{c\}$ est satisfiable.

Exemple 4.1 (Clause bloquée)
La clause c_1 ci-dessous est bloquée par le littéral a.

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

Nous proposons une généralisation de la notion de clause bloquée. Pour ce faire, nous définissons la notion de clause non fondamentale.

Défintion 4.2 (Clause non fondamentale)
On dit qu'une clause c d'une CNF Σ est **non fondamentale** *si cette clause est tautologique ou est subsumée par une autre clause de Σ.*

De cette définition, nous étendons la défintion des clauses bloquées aux clauses nf-bloquées.

Défintion 4.3 (Clause nf-bloquée)
Une clause c d'une CNF Σ est **nf-bloquée** *s'il existe un littéral l de c tel qu'il n'existe pas de résolvante en l ou toutes les résolvantes sont non fondamentales.*

Il est clair que la propriété des clauses bloquées peut être étendue au cas des clauses nf-bloquées.

Propriété 4.2 (Clause nf-bloquée)
Soit c une clause d'une CNF Σ. Si c est nf-bloquée alors Σ est satisfiable si et seulement si $\Sigma\backslash\{c\}$ est satisfiable.

Remarques 4.1
– La preuve de cette propriété découle immédiatement de celle faite par Kullmann pour les clauses bloquées [9].

– Il est clair que les clauses bloquées et les clauses contenant un littéral pur sont des cas particuliers de cette propriété.

L'exemple ci-dessous montre que l'élimination de clauses peut conduire à prouver la cohérence d'une formule CNF.

Exemple 4.2 (Clause nf-bloquée)

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

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

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

$$\left\{\begin{array}{l} c_4 = (\neg b \vee c \vee \neg d) \\ c_5 = (a \vee b \vee \neg c) \end{array}\right\} \xrightarrow[\text{littéral pur}]{c_4 \text{ nf-bloquée par } d} \left\{ c_5 = (a \vee b \vee \neg c) \right\}$$

$$\left\{ c_5 = (a \vee b \vee \neg c) \right\} \xrightarrow[\text{littéral pur}]{c_5 \text{ nf-bloquée par } a} \text{SAT}$$

Par ailleurs, comme nous l'avons constaté pour les littéraux purs, cette technique peut conduire à l'élimination de variables. En effet, il est toujours possible de rendre une clause nf-bloquée. Pour cela, il suffit d'ajouter à la CNF toutes les résolvantes de la clause en un littéral donné de cette clause.

Propriété 4.3
Soit c une clause d'une CNF Σ, il est toujours possible de rendre c nf-bloquée, par ajout de clauses à Σ.

Pour supprimer une variable, il suffit donc de nf-bloquer toutes les clauses où elle apparaît, ce qui aura pour conséquence de supprimer les clauses où apparaît la variable et donc de supprimer la variable elle-même. En pratique, il faut restreindre cette technique aux variables produisant un minimum de résolvantes (*e.g.* les variables ayant un faible nombre d'occurrences). Cette idée proche de la technique d'élimination proposée dans [4] a également été revisitée dans [18].

Plus généralement, la notion de clauses redondantes peut se définir de la manière suivante :

Défintion 4.4 (Clause redondante [2])
Une clause c d'une CNF Σ est dite **redondante** *si et seulement si $\Sigma \backslash \{c\} \models c$.*

En pratique, rechercher les clauses redondantes revient à prouver que la formule $\Sigma \wedge \neg c$ est incohérente. Il n'est donc pas raisonnable de rechercher les clauses redondantes dans leur généralité. Cependant, il est possible de restreindre la recherche (*e.g.* limiter le temps de recherche). Une restriction envisageable est de rechercher les clauses (ou les littéraux) impliquées par simple propagation unitaire[12].

Défintion 4.5 (Clause u-redondante)
Une clause c d'une CNF Σ est dite **u-redondante** *si l'on peut montrer par propagation unitaire que $\Sigma \wedge \neg c$ est insatisfiable ($\Sigma \backslash \{c\} \underset{Unit}{\models} c$).*

Il est clair que l'ensemble des clauses nf-bloquées et l'ensemble des clauses u-redondantes d'une CNF sont des sous-ensembles de l'ensemble des clauses redondantes de la CNF. La question d'une relation entre ces deux ensembles peut alors être posée. À partir de l'exemple 4.2, nous pouvons déduire que :

- il existe des clauses nf-bloquées non u-redondantes : c'est le cas de la clause c_1 ;
- il existe des clauses u-redondantes non nf-bloquées : si l'on étend la CNF initiale de l'exemple précédent avec les clauses $c_6 = (a \vee \neg d)$ et $c_7 = (\neg a \vee \neg d)$, la clause c_1 devient u-redondante mais n'est plus nf-bloquée ;
- il existe des clauses u-redondantes et nf-bloquées : toujours en étendant la CNF initiale avec les clauses $c'_6 = (b \vee c)$ et $c'_7 = (b \vee \neg c)$ la clause c_1 reste nf-bloquée en a et devient u-redondante.

4.2 Résolvante subsumante

De nombreuses implantations de DPLL [3] utilisent un pré-traitement consistant à ajouter toutes les résolvantes d'une taille inférieure ou égale à une constante (souvent 2). Cet ajout de clauses permet souvent d'augmenter l'efficacité des méthodes de résolution. Cependant, même si l'on borne la taille des résolvantes ajoutées, le nombre de clauses ajoutées peut être très grand et ce calcul devient alors prohibitif.

Nous proposons de limiter l'ajout de clauses à des résolvantes permettant de supprimer des clauses.

Défintion 4.6 (Résolvante subsumante)
Soit Σ une formule CNF, on appelle **résolvante subsumante** *une résolvante (obtenue à partir de deux clauses de Σ) qui subsume au moins une clause de Σ.*

Il est possible immédiatement d'identifier deux gains obtenus grâce à cette simplification :

1. une clause plus courte. En effet la clause produite subsumant au moins une clause, il résulte que cette résolvante est plus courte que la ou les clauses subsumées. Par ailleurs, pour ajouter cette résolvante, il suffit d'effacer le ou les littéraux de la clause subsumée ;
2. l'effacement du ou des littéraux de la clause en résolvante peut conduire à supprimer ou rendre pur ou unitaire une variable. Dans ces trois cas, l'espace des interprétations à parcourir en est fortement réduit.

Exemple 4.3 (Résolvante subsumante)
Les clauses $(a \vee b \vee c)$ et $(a \vee \neg c)$ génèrent la résolvante $(a \vee b)$ qui subsume la clause ternaire et permet d'effacer le littéral c de cette clause.

5 Résultats expérimentaux

Nous présentons dans cette partie les premiers résultats expérimentaux obtenus. Il ne s'agit ici que de valider l'approche proposée. Les résultats présentés dans cette partie constituent les préliminaires d'une validation expérimentale plus poussée qui sera conduite à court terme. Les techniques d'extraction et de simplification présentées ont été programmées en `C` sous Linux [3]. Les expérimentations ont été conduites sur un PC équipé d'un processeur Intel Pentium III cadencé à 1Ghz, doté de 256Mo de mémoire et fonctionnant sous un système Mandrake Linux 8.2.

Avant de poursuivre plus avant notre étude, nous nous sommes intéressés à la faisabilité de l'approche d'extraction d'équations. En effet, bien que convaincus de la présence de portes dans de nombreux « benchmarks » pour SAT, celles-ci n'ont pour l'instant jamais été exhibées. Par ailleurs, une autre inconnue est le temps d'extraction. Bien que l'exploitation du graphe partiel de clauses doit permettre de limiter les tests syntaxiques, rien ne garantit que cette extraction n'est pas prohibitive en temps.

Dans la table 1 nous répondons à ces interrogations. Cette table regroupe pour chaque instance testée :

– le nombre de clauses ($\#C$) et de variables ($\#V$) de l'instance initiale ;
– le nombre de portes détectées réparties en deux catégories : les chaînes d'équivalences ($\# \Leftrightarrow$), les portes $\vee$ et $\wedge$ ($\# \vee \wedge$) ;
– la taille de l'ensemble Γ de clauses restantes ($\#C_\Gamma$ et $\#V_\Gamma$) ;
– et le temps d'extraction.

Les instances sont issues des dernières compétitions internationales organisées sur la résolution pratique du problème SAT [5, 1, 16].

instance	$\#C$	$\#V$	$\# \Leftrightarrow$	$\# \vee \wedge$	$\#C_\Gamma$	$\#V_\Gamma$	tps (s)
par8-1-c	254	64	56	15	30	31	0.00
par8-1	1149	350	135	15	30	31	0.00
par16-1-c	1264	317	270	61	184	124	0.03
par16-1	3310	1015	560	61	184	124	0.03
par32-1-c	5254	1315	1158	186	622	375	0.14
par32-1	10277	3176	2261	186	622	375	0.15
barrel5	5383	1407	1065	152	1163	430	0.10
barrel6	8931	2306	1746	254	2013	821	0.20
barrel7	13765	3523	2667	394	3195	1337	0.40
barrel8	20083	5106	3864	578	4763	2158	0.73
ssa7552-125	3523	1512	1033	154	1270	501	0.03
ssa2670-130	3321	1359	859	254	1352	530	0.02
ssa0432-001	1027	435	225	43	244	124	0.03
bf1355-348	7271	2286	1082	383	3533	962	0.08
dubois100	800	300	200	0	0	0	0.02

TAB. 1 – Nombre d'équations extraites et temps d'extraction

[3]Le programme exécutable est disponible sur demande par mail : `ostrowski@cril.univ-artois.fr`.

instance	$\#C$	$\#V$	$\#V_\Gamma$	$\#V_{nd}$	tps (s)
par8-1	1149	350	31	8	0.00
par16-1	3310	1015	124	16	0.03
par16-2	3310	1015	124	16	0.03
par16-3	3310	1015	124	16	0.03
par16-4	3310	1015	124	16	0.03
par16-5	3310	1015	124	16	0.03
par32-1	10277	3176	375	32	0.14
par32-2	10277	3176	375	32	0.14
par32-3	10277	3176	375	32	0.15
par32-4	10277	3176	375	32	0.14
par32-5	10277	3176	375	32	0.15

TAB. 2 – Nombre de variables non définissables des instances « *parity* »

Ces résultats sont plus qu'encourageants puisqu'en effet, il existe effectivement de nombreuses portes dans différentes catégories d'instances et le temps d'extraction de ces portes est quasi négligeable (très inférieur à 1 seconde) si l'on considère que ce temps comprend également le temps de chargement de l'instance. D'autre part, la taille de l'ensemble Γ de clauses restantes après extraction est considérablement réduit (nombre de clauses divisé en moyenne par un facteur supérieur à 4 et nombre de variables divisé d'un facteur variant de 2 à 10) et peut même être nulle pour certains types d'instances (*e.g.* `dubois100`).

Il reste que l'ensemble des variables de Γ et celui des équations ne sont pas disjoints. Nous nous sommes donc intéressés à déterminer le nombre de variables étant véritablement non définies, c'est-à-dire le nombre de variables n'apparaissant jamais comme variable de sortie. La table 2 donne pour les instances « *parity* » le nombre de variables non définissables (ou variables d'entrée : $\#V_{nd}$) et le temps nécessaire à leur détermination. Ces instances ont été choisies car elles constituent un challenge pour la communauté SAT. Les résultats obtenus sont surprenants puisque sur les 3176 variables de l'instance `par32-1` seules 32 sont non définissables. C'est-à-dire que les valeurs de vérité des 3144 autres variables dépendent uniquement des 32 variables isolées par l'extraction. Ceci a pour effet de diminuer la complexité dans le pire des cas de 2^{3176} à 2^{32} !

Nous avons intégré nos techniques d'abstraction et de simplification comme pré-traitement à la procédure DPLL [3] utilisant une heuristique de branchement classique *à la* Jeroslow-Wang [8]. Cet algorithme appelé **LSAT** effectue un DPLL sur Γ et vérifie au cours de la recherche si l'interprétation construite ne contredit pas une des portes identifiées, auquel cas un retour arrière est opéré. Nous avons comparé notre algorithme avec les dernières versions des prouveurs SAT reconnus comme les meilleurs actuellement, à savoir : `Satz` [13], `EqSatz` [14], `Zchaff` [19]. Les résultats obtenus sont regroupés dans la table 3 où sont exprimés en secondes les temps de résolution de chaque prouveur[4].

Ces résultats montrent que sur de nombreuses instances, LSAT est largement plus efficace. De plus, LSAT résout certaines instances en moins d'une seconde alors que plus de 16 minutes ne furent pas suffisantes aux autres méthodes de résolution pour donner une réponse.

[4] $> n$ dans la table signifie que l'instance n'a pu pas être résolue après n secondes.

instance	$\#C$	$\#V$	SAT	Satz	EqSatz	Zchaff	**LSAT**
par8-1	1149	350	oui	0.05	0.1	0.01	**0.01**
par8-2	1149	350	oui	0.04	0.1	0.01	**0.01**
par8-3	1149	350	oui	0.07	0.1	0.01	**0.01**
par8-4	1149	350	oui	0.09	0.1	0.01	**0.01**
par8-5	1149	350	oui	0.05	0.1	0.01	**0.01**
par16-1	3310	1015	oui	8.96	0.19	0.47	**0.05**
par16-2	3310	1015	oui	0.48	0.20	0.88	**0.05**
par16-3	3310	1015	oui	16.79	0.22	4.07	**0.02**
par16-4	3310	1015	oui	11.15	0.17	0.82	**0.06**
par16-5	3310	1015	oui	1.59	0.18	0.41	**0.03**
par32-1-c	5254	1315	oui	>1000	540	>1000	**6**
par32-2-c	5254	1315	oui	>1000	24	>1000	**28**
par32-3-c	5254	1315	oui	>1000	1891	>1000	**429**
par32-4-c	5254	1315	oui	>1000	377	>1000	**16**
par32-5-c	5254	1315	oui	>1000	4411	>1000	**401**
par32-1	10227	3176	oui	>1000	471	>1000	**27**
par32-2	10227	3176	oui	>1000	114	>1000	**7**
par32-3	10227	3176	oui	>1000	4237	>1000	**266**
par32-4	10227	3176	oui	>1000	394	>1000	**3**
par32-5	10227	3176	oui	>1000	5645	>1000	**471**
barrel5	5383	1407	non	86	0.38	1.67	**0.19**
barrel6	8931	2306	non	853	0.71	8.29	**0.55**
barrel7	13765	3523	non	>1000	0.96	21.55	**6.23**
barrel8	20083	5106	non	>1000	1.54	53.76	**412**
dubois10	80	30	non	0.03	0.08	0.01	**0.01**
dubois20	160	60	non	26.53	0.03	0.01	**0.01**
dubois30	240	90	non	>1000	0.05	0.01	**0.01**
dubois50	400	150	non	>1000	0.08	0.01	**0.01**
dubois100	800	300	non	>1000	0.06	0.06	**0.01**
Urquhart3	578	49	non	>1000	>1000	190	**0.02**
Urquhart4	764	81	non	>1000	>1000	>1000	**0.03**
Urquhart5	1172	125	non	>1000	>1000	>1000	**0.06**
Urquhart15	11514	1143	non	>1000	>1000	>1000	**0.42**
Urquhart20	18528	1985	non	>1000	>1000	>1000	**0.64**
Urquhart25	29670	3122	non	>1000	>1000	>1000	**1.05**

TAB. 3 – Comparasion de LSAT avec Satz, EqSatz et Zchaff

6 Conclusion

Dans ce papier, nous avons proposé une technique d'extraction d'équations de la forme $y = f(x_1, x_2, \ldots, x_n)$ avec $f \in \{\vee, \wedge, \Leftrightarrow\}$ à partir d'une formule CNF. L'extraction de ces formules permet de réécrire la CNF sous la forme d'une conjonction d'équations et de clauses. Par ailleurs, les équations identifiées permettent de catégoriser les variables propositionnelles de la CNF en variables définissables ou non. Les variables définissables peuvent être vues comme les sorties des portes logiques caractérisées et permettent de réduire de manière significative la complexité dans le pire des cas. Nous avons également proposé différentes techniques de simplification de la forme clausale restante. Ces techniques permettent souvent d'éliminer des variables, ce qui a encore pour conséquence de réduire la complexité. Enfin, nous avons intégré ces techniques d'extraction et de simplification comme pré-traitement d'un nouvel algorithme de résolution pour SAT : LSAT. Cet algorithme s'est révélé extrêmement performant comparativement aux meilleures approches connues actuellement.

Ce travail nous semble donc aussi novateur que prometteur, aussi il ouvre de nombreuses pistes de recherche. La version proposée de l'algorithme LSAT est une version basique qui se contente de réaliser un DPLL sur Γ et de vérifier que l'interprétation courante ne contredit pas une des équations. Il nous semble que cette première version peut être très largement améliorée. En effet, l'heuristique de branchement actuellement utilisée ne travaille que sur Γ. Il serait intéressant de développer des heuristiques spécifiques à la résolution de ces formules hybrides, c'est-à-dire de développer des heuristiques travaillant à la fois sur les portes et sur les clauses restantes. D'autre part, il serait également intéressant que l'algorithme exploite les propriétés intrinsèques de chaque équation. Par ailleurs, la recherche d'équations est pour l'instant réalisée en pré-traitement. Nous avons vu que cette recherche n'est pas extrêmement coûteuse. Cependant dans quelle mesure est-il possible d'extraire dynamiquement des équations au cours de la recherche ? Enfin, nous envisageons une extension de ces travaux à la simplification et à la résolution de formules booléennes générales.

Remerciements

Ce travail est soutenu par la communauté européenne au travers un programme FEDER, par la Région Nord/Pas-de-Calais et par l'Institut Universitaire de Technologie de Lens.

Références

[1] First international competition and symposium on satisfiability testing, March 1996. Beijing (China).

[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.

[3] Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem proving. *Journal of the Association for Computing Machinery*, 5 :394–397, 1962.

[4] Martin Davis and Hilary Putnam. A computing procedure for quantification theory. *Journal of the Association for Computing Machinery*, 7 :201–215, 1960.

[5] Second Challenge on Satisfiability Testing organized by the Center for Discrete Mathematics and Computer Science of Rutgers University, 1993. `http ://dimacs.rutgers.edu/Challenges/`.

[6] 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.

[7] B. Dunham and H. Wang. Towards feasible solution of the tautology problem. *Annals of Mathematical Logic*, 10 :117–154, 1976.

[8] Robert G. Jeroslow and Jinchang Wang. Solving propositional satisfiability problems. *Annals of Mathematics and Artificial Intelligence*, 1 :167–187, 1990.

[9] 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.

[10] Oliver Kullmann. New methods for 3-sat decision and worst-case analysis. *Theoretical Computer Science*, pages 1–72, 1997.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] Workshop on theory and applications of satisfiability testing, 2001. http ://www.cs.washington.edu/homes/kautz/sat2001/.

[17] Fifth international symposium on the theory and applications of satisfiability testing, May 2002. http ://gauss.ececs.uc.edu/Conferences/SAT2002/.

[18] 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.

[19] 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.