

# Réduction d'instances de SAT vers des instances polynomiales

## Reducing SAT instances to polynomial ones

Olivier Fourdrinoy

Éric Grégoire

Bertrand Mazure

Lakhdar Saïb

CRIL CNRS & Université d'Artois  
Rue Jean Souvraz SP18  
F-62307 Lens Cedex France  
{fourdrinoy, gregoire, mazure, sais}@cril.fr

### Résumé

*Ces dernières années, le raisonnement en calcul propositionnel a fait l'objet de nombreuses recherches de la part de la communauté d'IA. L'efficacité actuelle des procédures de vérification de la satisfaisabilité d'une formule booléenne a permis l'utilisation du cadre booléen dans de nombreux paradigmes de raisonnement. Le problème SAT, consistant à vérifier la satisfaisabilité d'un ensemble de clauses propositionnelles est au coeur de ce cadre. Dans ce papier, nous proposons un nouveau pré-traitement des instances de SAT. Ce pré-traitement, linéaire en temps, permet de reconnaître une nouvelle classe polynomiale d'instances de SAT, c'est-à-dire des instances pour lesquelles la reconnaissance et la résolution peuvent s'effectuer en temps polynomial. Nous montrons que certaines instances utilisées lors des compétitions internationales de solveurs SAT appartiennent à cette nouvelle classe polynomiale et sont donc faciles à résoudre.*

### Mots Clefs

SAT, classe polynomiale, propagation unitaire, simplification

### Abstract

*This last decade, propositional reasoning and search has been one of the hottest topics of research in the A.I. community, as the Boolean framework has been recognized as a powerful setting for many reasoning paradigms thanks to dramatic improvements of the efficiency of satisfiability checking procedures. SAT, namely checking whether a set of propositional clauses is satisfiable or not, is the technical core of this framework. In the paper, a new linear-time pre-treatment of SAT instances is introduced. Interestingly, it allows us to discover a new polynomial-time fragment of SAT that can be recognized in linear-time, and show that some benchmarks from international SAT competitions that were believed to be difficult ones, are actually polynomial-time and thus easy-to-solve ones.*

### Keywords

SAT, polynomial class, unit propagation, simplification.

### Introduction

De nombreuses approches ont été proposées pour résoudre les instances difficiles de SAT. Des approches directes se sont concentrées sur le développement d'algorithmes logiquement complets ou non. Les techniques de recherche locale (cf. [26]) ainsi que les variantes plus ou moins élaborées de la procédure de Davis-Loveland-Logemann (DPLL) [7] (cf. [22, 12]) permettent de résoudre de nombreuses classes d'instances difficiles. Les approches indirectes tentent de résoudre les instances en utilisant des approximations ou des techniques de compilation [6, 2, 25]. En particulier, les techniques de compilation, qui ont été développées dans le cadre plus général de la déduction propositionnelle, visent à transformer l'ensemble de clauses booléennes en une forme déductivement équivalente appartenant à un fragment polynomial, n'hésitant pas pour cela à recourir à des transformations parfois exponentielles en taille pourvu que la forme compilée soit de taille traitable. Enfin, d'autres approches se sont concentrées sur la découverte et l'étude de fragments de SAT reconnaissables et solvables en temps polynomial (cf. [9, 3, 1, 4]).

La contribution de ce papier se revendique des trois familles d'approches décrites précédemment. Nous introduisons un nouveau prétraitement des instances de SAT : il peut être lancé sur une instance avant de recourir à des approches directes. Il peut être considéré comme une tentative de compilation de l'instance SAT en une autre plus facile à résoudre. Cependant, contrairement aux techniques de compilation habituelles, le processus de transformation requiert un temps de traitement polynomial, et ne garantit pas que l'ensemble de clauses résultant de la transformation appartient à une classe polynomial. La pertinence de ce prétraitement est néanmoins validée par la détection d'instances polynomiales ou encore par

la simplification d'instances qui en favorise la résolution. Enfin, une nouvelle classe polynomiale de SAT, appelée  $U_{Horn}SAT$  (*HORN modulo la propagation unitaire*), est mise en évidence. Il est intéressant de constater que cette classe peut être détectée par le prétraitement en temps polynomial. En d'autres termes, les instances de SAT pouvant être plongées dans la classe *HORNSAT* par notre prétraitement appartiennent à la classe  $U_{Horn}SAT$ .

Schématiquement, ce prétraitement fonctionne comme suit. La base de ce prétraitement est un mécanisme polynomial de déduction (par exemple la propagation unitaire (PU), qui est un mécanisme déductif linéaire en temps). Étant donnée une classe polynomiale de SAT reconnaissable syntaxiquement (par exemple *HORNSAT*), n'importe quelle instance SAT peut être partitionnée en deux sous-ensembles de clauses : le premier contient les clauses qui appartiennent à la classe polynomiale visée tandis que le second contient les clauses qui n'appartiennent pas à celle-ci. Pour chacune de ces dernières clauses, nous essayons de découvrir une sous-clause appartenant à la classe polynomiale en utilisant le mécanisme polynomial de déduction. En cas de succès, cette sous-clause peut remplacer la clause initiale, et augmente la taille du sous-ensemble polynomial. En cas d'échec, nous vérifions également si la clause elle-même est redondante modulo la propagation unitaire [14] ou pas. Les travaux présentés ici se concentrent sur le mécanisme de déduction restreint à la propagation unitaire et sur la classe polynomiale *HORNSAT*.

Le papier est organisé comme suit. Dans la prochaine section, les notions techniques de base sont introduites ainsi que les fragments polynomiaux étudiés dans ce papier. Ensuite le prétraitement est décrit avant de reporter et d'analyser les expérimentations. Enfin, nous montrons les liens entre nos travaux et des travaux antérieurs notamment en ce qui concerne l'exploitation de la propagation unitaire.

## 1 Rappels techniques

Soit  $\mathcal{L}$  un langage logique booléen construit sur un ensemble fini de variables booléennes notées  $a, b, c$ , etc. Les formules sont représentées par des lettres majuscules comme  $C$ . Des lettres grecques telles que  $\Gamma$  ou  $\Sigma$  correspondent à des ensembles de formules. Une interprétation est une fonction qui affecte des valeurs de vérités  $\{\text{vrai}, \text{faux}\}$  à toutes les variables. Une formule est consistante ou satisfiable quand il y a au moins une interprétation qui la satisfait, c.-à-d. qui la rend *vraie*. Une interprétation est notée par des lettres majuscules telles que  $I$  et est représentée par l'ensemble des littéraux qu'elle satisfait. Concrètement, toute formule de  $\mathcal{L}$  peut être représentée (tout en préservant la satisfiabilité) par un ensemble de clauses (interprété comme une conjonction de clauses). Une clause est une disjonction finie de littéraux. Un littéral est une variable booléenne qui peut

être falsifiée. Les clauses sont représentées par l'ensemble des littéraux qu'elles contiennent. Par exemple, la clause  $C = a \vee b \vee \neg c \vee \neg d$  sera représentée par l'ensemble  $\{a, b, \neg c, \neg d\}$ . Une clause est dite positive (resp. négative) si elle ne contient pas de littéraux négatifs (resp. positifs). La taille d'une clause est le nombre de littéraux qu'elle contient. Une clause unitaire contient exactement un littéral alors qu'une clause binaire en contient exactement deux. La clause vide est notée  $\perp$  et elle représente l'inconsistance. Une clause  $C'$  est appelée sous-clause de  $C$  si  $C' \subset C$ . Une sous-clause  $C'$  de  $C$  est dite maximale si  $|C| - |C'| = 1$ . La résolvente de  $C_1 = (p \vee \alpha)$  et  $C_2 = (\neg p \vee \beta)$  est définie par  $Res(C_1, C_2) = (\alpha \vee \beta)$  (règle de résolution). Cette résolvente est une conséquence logique de  $C_1$  et  $C_2$ .

SAT est le problème NP-Complet qui consiste à tester si un ensemble de clauses booléennes (appelé également CNF) est satisfiable ou pas, c.-à-d. s'il existe une interprétation qui satisfait toutes les clauses de l'ensemble ou non.

Un mécanisme de déduction présente un rôle central dans ce papier, il s'agit de la propagation unitaire (PU). PU est un processus linéaire qui simplifie récursivement une instance de SAT en propageant les contraintes exprimées par des clauses unitaires. Soit  $\Sigma$  une instance de SAT,  $PU(\Sigma)$  se définit comme étant la formule obtenue par propagation unitaire. Une clause  $C$  est une PU conséquence de  $\Sigma$ , notée  $\Sigma \models^* C$ , si et seulement si  $PU(\Sigma \wedge \neg C)$  permet de dériver la clause vide.

Quelques fragments de  $\mathcal{L}$  permettent d'utiliser des algorithmes polynomiaux en temps pour résoudre le problème SAT. Ces fragments sont appelés classes polynomiales de SAT. Parmi celles-ci, notons la classe des formules de *HORNSAT*, qui est constituée uniquement de clauses de Horn. Une clause de *HORN* (resp. *reverseHORN*) contient au plus un littéral positif (resp. négatif). Les formules *2SAT* (formules composées de clauses binaires) ou encore les clauses *Horn-renommables* constituent également des classes polynomiales : les clauses *HORN-renommables* sont des clauses qui peuvent être transformées en clauses de Horn en remplaçant certains littéraux par leurs opposés. Mentionnons la hiérarchie de classes de Dalal et Etherington [3] ainsi que la classe des formules  $Q-HORN$  [1] qui contiennent strictement toutes les formules *2SAT*, *HORNSAT*, *reverse-HORNSAT* et *HORN-renommables*. Toutes ces formules peuvent être reconnues et résolues en temps polynomial. Un fragment polynomial de  $\mathcal{L}$  présente pour nous un intérêt particulier : la classe Quad introduite par Dalal [4].

M. Dalal dans [4] propose une nouvelle classe polynomiale *QUAD* pour **SAT**, qui peut être vue comme une extension des hiérarchies de Dalal-Etherington.

### Définition 1 classe *ROOT*

Une formule  $F$  appartient à la classe *ROOT* si l'une des

conditions suivantes est satisfaite :

- $F$  contient la clause vide ;
- toute clause de  $F$  contient au moins un littéral positif ;
- toute clause de  $F$  contient au moins un littéral négatif ;
- toute clause de  $F$  est de longueur 2.

Soit  $\Sigma$  une formule CNF. Dalal utilise un ordre total sur l'ensemble des clauses  $\Sigma$ . Soit  $\prec$  un ordre total arbitraire sur un ensemble de littéraux (par exemple,  $x_1 \prec x_2 \prec \dots \prec x_n \prec \neg x_1 \prec \neg x_2 \prec \dots \prec \neg x_n$ ), cet ordre induit un ordre total sur les clauses :  $c' \prec c$  si et seulement si  $c' \subsetneq c$  ou il existe un littéral  $l$  dans  $c \setminus c'$  tel que  $l \prec q$  pour tout littéral  $q$  de  $c' \setminus c$ .  $\prec$  détermine un ordre total sur les clauses de  $\Sigma$ . Pour toute clause de  $\Sigma$ ,  $\prec$  détermine également un ordre total sur les sous-clauses maximales.

**Définition 2** *QUAD*

Une formule CNF  $\Sigma$  appartient à la classe *QUAD* si l'une des conditions suivantes est vérifiée.

1.  $\Sigma$  appartient à la classe *ROOT* ;
2. soit  $c'$  la première sous-clause maximale de la première clause  $c$  de  $\Sigma$  pour laquelle  $\Sigma \cup \{-c'\}$  appartient à la classe *ROOT*. Une des deux propriétés suivantes est vérifiée :

- a  $\Sigma \cup \{-c'\}$  est satisfiable ;
- b  $(\Sigma \setminus \{c\}) \cup \{c'\}$  appartient à *QUAD*.

Dalal propose un algorithme quadratique<sup>1</sup> permettant pour un ordre donné de tester si une formule appartient à la classe *QUAD*. Il propose un algorithme de même complexité pour tester la satisfiabilité d'une formule *QUAD*.

On peut cependant noter que la classe *QUAD* n'est polynomiale que pour un ordre fixé des variables (comme pour les formules bien imbriquées<sup>2</sup>). De plus, il est prouvé dans [21] que la classe *QUAD* n'est pas stable par adjonction de clauses unitaires. En effet l'ajout de clauses unitaires dans la formule peut sortir la formule de la classe *QUAD*.

## 2 Un nouveau prétraitement

L'idée principale est de réduire le fragment non-polynomial d'une instance SAT  $\Sigma$  via la propagation unitaire. Étant donnée une classe polynomiale,  $\Sigma$  est partitionnée en deux sous-ensembles distincts. Le premier est constitué des clauses appartenant à la classe polynomiale fixée, telle que *HORN*, *reverse - HORN* ou

<sup>1</sup> $O(Nk)$  où  $N$  est la longueur totale de la formule et  $k$  la longueur de la plus longue clause.

<sup>2</sup>Inspirée de [19], la classe des formules bien imbriquées est introduite dans [16]. Un algorithme de reconnaissance linéaire pour un ordre fixé des variables est présenté dans [24]. De même, D.E. Knuth fournit un algorithme de décision linéaire pour un ordre fixé sur les variables. Nous ne disposons pas à l'heure actuelle d'algorithmes « complets » (c.-à-d. pour tout ordre sur les variables) qui soient polynomiaux. L'intérêt de cette classe pour un bon nombre d'applications en est par conséquent très amoindri.

encore les formules positives, etc. Le second contient les clauses restantes. Quand la seconde partie est vide, cela signifie que la formule  $\Sigma$  peut être résolue en un temps borné par un polynôme. Différentes formes de déduction de sous-clauses peuvent être définies en fonction de la classe polynomiale visée. Par exemple, si on considère les clauses binaires, le mécanisme de déduction de sous-clauses concernera les clauses dont la taille est supérieure à deux. Par la suite, nous limitons cette idée générale à la classe des clauses de *HORN SAT*.

Introduisons d'abord quelques définitions.

**Définition 3** *clause  $U_{Horn}$*

Soit  $C = \{\neg n_1, \dots, \neg n_n, p_1, \dots, p_p\}$ , avec  $n \geq 0$  et  $p > 1$  une clause de  $\Sigma$ .  $C$  est appelée *clause  $U_{Horn}$*  de  $\Sigma$  ssi  $\exists C' = \{\neg n_1, \dots, \neg n_n, p_i\}$  une sous-clause de  $C$ , telle que  $\Sigma \models^* C'$  ou  $\Sigma \models^* \{\neg n_1, \dots, \neg n_n\}$ .

**Exemple 1** *Clause  $U_{Horn}$*

$$\left. \begin{array}{l} \neg a \vee \neg b \vee c \vee d \\ \neg a \vee \neg b \vee e \\ c \vee \neg e \end{array} \right\} \Rightarrow \neg a \vee \neg b \vee c$$

En propageant  $a$ ,  $b$  et  $\neg c$ , on obtient une contradiction puisque la deuxième clause produit  $e$  et que la troisième produit  $\neg e$ . La première clause est donc réduite à  $\neg a \vee \neg b \vee c$ . Elle est ainsi plongée dans *HORN*.

**Propriété 1** Si  $C \in \Sigma$  est une clause *U<sub>HORN</sub>* et  $C' \subset C$  est une clause de *HORN* telle que  $\Sigma \models^* C'$  alors  $\Sigma$  est satisfiable si et seulement si  $(\Sigma \setminus \{C\}) \cup \{C'\}$  est satisfiable.

La propriété précédente montre que lorsqu'une clause  $C$  de  $\Sigma$  est *U<sub>Horn</sub>*, elle peut être remplacée dans  $\Sigma$  par une clause de Horn sans modifier la satisfiabilité de  $\Sigma$ .

Introduisons maintenant deux propriétés qui nous permettront d'ajouter deux nouveaux opérateurs de réduction, à savoir *U-NRes* et *U-PRes*.

**Propriété 2** Soit  $C = \{\neg n_1, \dots, \neg n_n, p_1, \dots, p_p\}$  une clause de  $\Sigma$ . Si  $\Sigma \models^* \{\neg n_1, \dots, \neg n_n\}$  ou  $\exists p_i \in C$  t.q.  $\Sigma \models^* \{\neg n_1, \dots, \neg n_n, p_i\}$  et  $\Sigma \models^* \{\neg n_1, \dots, \neg n_n, \neg p_i\}$ , alors  $\Sigma \models \{\neg n_1, \dots, \neg n_n\}$ .

**Définition 4** *clause U-NRes*

Quand une clause  $C = \{\neg n_1, \dots, \neg n_n, p_1, \dots, p_p\}$  de  $\Sigma$  satisfait la propriété 2, *U-NRes(C)* est définie comme  $\{\neg n_1, \dots, \neg n_n\}$ .

**Exemple 2** clause *U-NRes*

$$\left. \begin{array}{l} \neg a \vee \neg b \vee c \vee d \\ \neg a \vee \neg b \vee e \\ \neg a \vee \neg b \vee \neg e \end{array} \right\} \Rightarrow \neg a \vee \neg b$$

En propageant  $a$  et  $b$ , un conflit est détecté entre  $e$  et  $\neg e$ . La clause négative est ainsi produite.

**Exemple 3** autre clause *U-NRes*

$$\left. \begin{array}{l} \neg a \vee b \vee c \\ \neg a \vee d \\ \neg d \vee e \vee f \\ \neg e \vee b \\ \neg b \vee \neg e \\ \neg f \vee b \\ \neg b \vee \neg f \end{array} \right\} \Rightarrow \left. \begin{array}{l} \neg a \vee \neg b \\ \neg a \vee b \end{array} \right\} \Rightarrow \neg a$$

En propageant  $a$  et  $b$ ,  $\neg e$  et  $\neg f$  sont produits et la troisième clause est falsifiée. On peut ainsi produire la clause  $\neg a \vee \neg b$ . En relançant une propagation unitaire avec  $a$  et  $\neg b$ . La troisième clause est encore falsifiée, on peut donc en déduire une seconde clause  $\neg a \vee b$ . Par le biais du principe de résolution, on obtient alors la clause  $\neg a$ .

**Propriété 3** Soit  $C = \{\neg n_1, \dots, \neg n_n, p_1, \dots, p_p\}$  une clause de  $\Sigma$ .

Si  $\exists p_i \in C$  tel que  $\Sigma \not\models^* \{\neg n_1, \dots, \neg n_n, p_i\}$  et  $\Sigma \models^* \{\neg n_1, \dots, \neg n_n, \neg p_i\}$ , alors  $\Sigma \models^* \{\neg n_1, \dots, \neg n_n, p_1, \dots, p_{i-1}, p_{i+1}, \dots, p_p\}$ .

**Définition 5** clause *U-PRes*

Quand une clause  $C = \{\neg n_1, \dots, \neg n_n, p_1, \dots, p_p\}$  de  $\Sigma$  satisfait la propriété 3 pour les littéraux allant de  $p_i$  à  $p_j$ ,  $U-PRes(C)$  est définie comme  $\{\neg n_1, \dots, \neg n_n, p_1, \dots, p_{i-1}, p_{j+1}, \dots, p_p\}$ .

**Exemple 4** clause *U-PRes*

$$\left. \begin{array}{l} \neg a \vee \neg b \vee c \vee d \vee e \\ \neg a \vee \neg c \vee f \\ \neg b \vee \neg c \vee \neg f \end{array} \right\} \Rightarrow \neg a \vee \neg b \vee d \vee e$$

En propageant  $a$ ,  $b$  et  $c$ , un conflit est détecté entre  $f$  et  $\neg f$ . La clause  $\neg a \vee \neg b \vee \neg c$  peut être ajoutée mais l'idée de *U-PRes* est de réduire la clause initiale par le biais du principe de résolution sur  $c$ .

**Exemple 5** Détection des résolventes subsumantes

$$\left. \begin{array}{l} \neg a \vee \neg b \vee c \vee d \\ \neg a \vee \neg b \vee c \vee \neg d \end{array} \right\} \Rightarrow \neg a \vee \neg b \vee c$$

En propageant  $a$ ,  $b$  et  $\neg c$ , un conflit est détecté entre  $d$  et  $\neg d$ . Cette méthode permet de détecter les résolventes subsumantes.

Des précédentes propriétés, nous produisons une nouvelle classe traitable appelée  $U_{Horn}SAT$ . L'algorithme 1 décrit le mécanisme de reconnaissance de cette classe. Après

avoir enregistré les clauses de Horn dans  $\Sigma'$ , toutes les clauses restantes  $C$  sont testées successivement (ligne 3). Conformément à la propriété 2, quand la partie négative de  $C$  est PU-dérivable de  $\Sigma$ , cette partie négative est considérée comme une nouvelle clause de Horn et de fait est stockée dans  $\Sigma'$  (lignes 4). Sinon, la seconde partie de la propriété 2 est implémentée (lignes 8 et 9). Les tests effectués aux lignes 8 et 11 traduisent la propriété 3. Dans le but d'obtenir  $U-PRes(C)$ , les tests (lignes 12 à 14) permettent l'insertion dans  $\Sigma'$  de la plus petite clause (en terme de nombre de littéraux positifs). À la ligne 15, un appel à l'algorithme de PU-redondance [14] permet de supprimer les clauses redondantes modulo PU. Finalement, la formule initiale  $\Sigma$  est U-Horn si et seulement si la formule simplifiée  $\Sigma'$  est Horn. Le processus de simplification d'une clause est illustré par la figure 1.

### 3 Résultats expérimentaux

Dans le but d'évaluer l'intérêt pratique de notre prétraitement, une variante de l'algorithme 1 a été implémentée et expérimentée. Pour des raisons d'efficacité, nous ne manipulons pas deux CNF ( $\Sigma$  et  $\Sigma'$ ) comme le stipule l'algorithme 1 mais une seule CNF  $\Sigma$ . En conséquence, la CNF simplifiée est dépendante de l'ordre de considération des clauses. Exécuter le programme une fois ne garantit pas que toutes les simplifications possibles soient faites, tandis que l'algorithme 1 assure ce dernier point. Pour exécuter toutes les simplifications possibles, le programme doit être itéré jusqu'à ce qu'aucune nouvelle clause de Horn ne soit produite.

Le programme a été exécuté sur un panel de 1600 problèmes choisis de manière aléatoire parmi les instances DIMACS [8] et les instances issues des dernières compétitions SAT ([www.satcompetition.org](http://www.satcompetition.org)). Toutes les expérimentations ont été conduites sur un processeur Intel(R) Xeon(TM) CPU 3.00Ghz avec 2Go de mémoire sous Linux CentOS release 4.1.

Sur ces 1600 instances, nous avons extrait 99 instances appartenant à la classe polynomiale  $U_{HORN}SAT$ , identifiées après un passage unique de notre prétraitement sur les formules testées. En autorisant des passages multiples (jusqu'à l'obtention d'un point fixe) nous avons identifié 28 nouvelles instances répondant aux critères de la classe polynomiale  $U_{HORN}SAT$ . Notons que ces résultats sont très satisfaisants car à notre connaissance, aucune des instances détectées n'avait jusqu'alors été prouvée polynomiale. De plus, le nombre d'occurrences de la classe (6% ou 8% des instances testées suivant que l'on prenne ou pas en compte les instances identifiées après plusieurs prétraitements) est très intéressant, d'autant que les instances testées n'ont pas été construites pour démontrer l'existence de la classe polynomiale.

---

**Données :** Une instance SAT  $\Sigma$   
**Résultat :** *true* si  $\Sigma$  est U-Horn ; *false* sinon

```

1 début
2    $\Sigma' \leftarrow \{C \mid C \in \Sigma \text{ telles que isHorn}(C)\};$ 
3   pour chaque  $C \in \Sigma$  telles que  $C = \{\neg n_1, \dots, \neg n_n, p_1, \dots, p_p\}$  avec  $n \geq 0$  et  $p > 1$  faire
4     si  $\Sigma \models^* \{\neg n_1, \dots, \neg n_n\}$  alors  $\Sigma' \leftarrow \Sigma' \cup \{\{\neg n_1, \dots, \neg n_n\}\};$ 
5     sinon
6        $\Sigma'' \leftarrow \emptyset$  ;  $C' \leftarrow C$ ;
7       pour chaque  $p_i \in C$  faire
8         si  $\Sigma \models^* \{\neg n_1, \dots, \neg n_n, p_i\}$  alors
9           si  $\Sigma \models^* \{\neg n_1, \dots, \neg n_n, \neg p_i\}$  alors  $\Sigma' \leftarrow \Sigma' \cup \{\{\neg n_1, \dots, \neg n_n\}\};$ 
10          sinon  $\Sigma'' \leftarrow \Sigma'' \cup \{\{\neg n_1, \dots, \neg n_n, p_i\}\};$ 
11          sinon si  $\Sigma \models^* \{\neg n_1, \dots, \neg n_n, \neg p_i\}$  alors  $C' \leftarrow C' \setminus \{p_i\};$ 
12          si  $\{\neg n_1, \dots, \neg n_n\} \notin \Sigma'$  alors
13            si  $\Sigma'' = \emptyset$  alors  $\Sigma' \leftarrow \Sigma' \cup \{C'\};$ 
14            sinon  $\Sigma' \leftarrow \Sigma' \cup \Sigma'';$ 
15    $\Sigma' \leftarrow PU\_redondance(\Sigma')$ ;
16   retourner isHorn( $\Sigma'$ );
17 fin

```

---

#### Algorithme 1 – isU-Horn

---

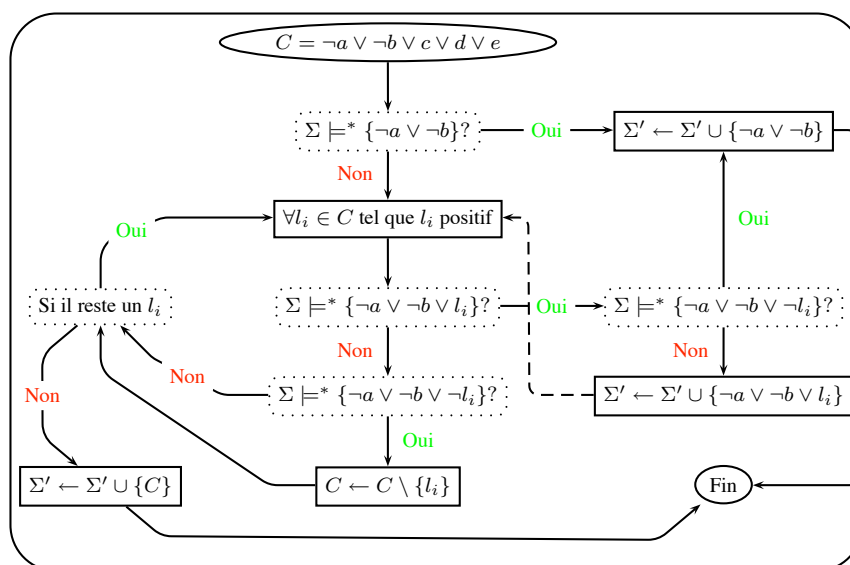
Ces instances, sont consultables en annexes dans les tableaux 1 et 2. Pour chaque instance, son nom, sa taille (#var. #cla.), le nombre de propagations (#UP) et le temps passé en secondes pour plonger l'instance dans  $U_{HORN}SAT$  sont donnés. Quand le programme est itéré jusqu'à ce qu'aucune nouvelle clause de Horn ne soit produite, 28 instances supplémentaires sont plongées dans  $U_{HORN}SAT$ . Ces instances sont également consultables en annexe dans le tableau 2 où « cla sup » (resp. « var sup ») représente pourcentage des clauses (resp. variables) enlevées par la méthode et où « #lit » représente le nombre de littéraux qui ont été enlevés.

Cependant, notre prétraitement ne permet pas toujours de plonger une instance dans un fragment polynomial. Généralement, la taille globale de l'instance s'en trouve tout de même diminuée de façon significative, tandis que la partie polynomiale est augmentée en conséquence. Il est intéressant de constater que cette réduction semble valable du point de vue du problème de résolution. Dans le tableau 3, le temps requis pour résoudre des instances en utilisant Minisat [12] est comparé avec le temps passé par une combinaison de notre prétraitement avec Minisat. Dans cette table, les colonnes « Minisat » représentent le temps consommé par Minisat pour résoudre l'instance originale (« original ») et l'instance simplifiée (« simplifié »). Les colonnes « % gain » représentent les gains (en pourcentage) obtenus par le prétraitement quand le temps de simplification est pris en considération (« total ») ou pas (« par-

tiel »). Les gains varient suivant les instances testées. Les méthodes de résolution actuelles ayant tendance à ajouter des clauses, on comprendra aisément que la simplification via la subsumption de clause puisse dans certains cas affaiblir les résultats des solveurs. Néanmoins, sur bon nombre d'instances, on observe une amélioration du temps de résolution. Cette simplification est un atout notable de notre classe polynomiale. En effet, dans la majorité des classes polynomiales, l'algorithme de détection ne permet que la détection de l'appartenance. En cas d'échec de cette détection, le temps utilisé par l'algorithme aura été dépensé en pure perte.

## 4 Liens avec des travaux existant

La classe  $U_{HORN}SAT$  présente quelques similitudes avec la classe QUAD proposée par Dalal [4]. En effet, les deux approches utilisent une procédure de déduction de sous-clause basée sur la propagation unitaire. Cependant, notre approche diffère de celle de Dalal sur plusieurs points. Tout d'abord, elle est indépendante de l'ordre de choix des littéraux. Ensuite, chaque classe de notre famille de classes polynomiales est basée sur un seul fragment polynomial alors que Dalal en considère plusieurs simultanément. Enfin, la suppression des clauses redondantes [14] ainsi que les opérateurs de réductions ajoutés rendent ces deux classes incomparables.

FIG. 1 – Traitement d’une clause par la procédure *isU-Horn*

Bien évidemment, l’idée de prétraiter les instances SAT n’est pas nouvelle. Beaucoup de solveurs SAT modernes incluent des techniques de prétraitement. Par exemple,  $C - SAT$  [10] opère un usage restreint du principe de résolution comme prétraitement polynomial. D’autres algorithmes complets utilisent la recherche locale en premier lieu et lorsque celle-ci ne fournit pas de solution, utilisent des informations collectées au cours de cette recherche locale pour guider leur exploration de l’arbre de recherche [20, 13]. Plus récemment, SatELite, qui est le prétraitement employé dans un des meilleurs solveurs SAT actuels, simplifie l’instance en éliminant des variables [11].

En raison de sa linéarité temporelle, l’algorithme de PU a été exploité de plusieurs manières dans le contexte SAT. Ajoutons à cela que la PU est une composante clé des algorithmes de type *DPLL*. Par exemple,  $C - SAT$  et *Satz* emploient un traitement local pendant des étapes importantes de l’exploration de l’espace de recherche, basé sur la PU. L’idée est de dériver des littéraux impliqués, de détecter des contradictions locales et de guider le choix de la prochaine variable à assigner [10, 18]. Dans [17], l’idée d’une double PU est explorée dans le contexte du problème SAT. Dans [23, 15], la PU est employée comme un outil efficace pour détecter des dépendances fonctionnelles dans des instances SAT. La technique PU a été également exploitée dans [5] afin de dériver

des sous-clauses en employant le graphe d’implication de l’instance SAT, et accélérer ainsi le processus de résolution.

## Conclusions et perspectives

Dans ce papier, une nouvelle technique de prétraitement linéaire en temps pour les instances de SAT est introduite. Cette technique est basée sur l’efficacité de l’algorithme de propagation unitaire qui est utilisé pour augmenter la partie polynomiale de l’instance SAT testée. Il est intéressant de constater que des problèmes issus des compétitions SAT se révèlent être polynomiaux et même résolus directement par notre prétraitement. De plus, le prétraitement se révèle performant pour faciliter la résolution de certaines instances par les meilleurs solveurs actuel. Nous projetons d’étendre cette technique à d’autres classes polynomiales de SAT.

## Remerciements

Ces travaux de recherche ont été financés par le CNRS, l’Union Européenne (fonds Feder) et la région *Nord Pas-de-Calais*.

CNF instances	#var.	#cla.	#PU	Temps	CNF instances	#var.	#cla.	#PU	Temps
aim-100-1_6-yes1-4	100	160	179	0	IBM_FV_2004_rule_batch...				
aim-100-2_0-yes1-2	100	200	456	0	IBM_...04...15	15300	65598	397812	0,25
aim-100-6_0-yes1-1	100	600	2502	0	IBM_...05...15	25128	134922	1708357	1,22
aim-100-6_0-yes1-2	100	600	2534	0	IBM_...15...100	226970	893496	2432156	2,46
aim-100-6_0-yes1-3	100	600	777	0	IBM_...15...15	30790	119911	184301	0,19
aim-100-6_0-yes1-4	100	600	568	0	IBM_...15...20	42330	165416	252596	0,26
aim-200-6_0-yes1-2	200	1200	6113	0,01	IBM_...15...25	53870	210921	329216	0,33
aim-200-6_0-yes1-4	200	1200	696	0	IBM_...15...30	65410	256426	413391	0,42
aim-50-2_0-yes1-2	50	100	218	0	IBM_...15...35	76950	301931	506031	0,5
aim-50-2_0-yes1-3	50	100	250	0	IBM_...15...40	88490	347436	606086	0,6
aim-50-2_0-yes1-4	50	100	156	0	IBM_...15...45	100030	392941	714746	0,71
aim-50-6_0-yes1-1	50	300	516	0	IBM_...15...50	111570	438446	830681	0,83
aim-50-6_0-yes1-2	50	300	692	0	IBM_...15...55	123110	483951	955361	0,99
aim-50-6_0-yes1-3	50	300	440	0	IBM_...15...60	134650	529456	1087176	1,07
aim-50-6_0-yes1-4	50	300	1621	0	IBM_...15...65	146190	574961	1227876	1,22
cnf-r1-b3-k1.2	660004	5281	56944	0,21	IBM_...15...70	157730	620466	1375571	1,38
cnf-r1-b4-k1.1	397893	7089	105048	0,18	IBM_...15...75	169270	665971	1532291	1,53
cnf-r1-b4-k1.2	922148	6818	60079	0,29	IBM_...15...80	180810	711476	1695866	1,69
cnf-r2-b2-k1.2	406052	6064	54402	0,15	IBM_...15...85	192350	756981	1868606	1,88
cnf-r2-b3-k1.2	668180	9169	100807	0,27	IBM_...15...90	203890	802486	2048061	2,06
cnf-r2-b4-k1.1	406052	12784	178182	0,25	IBM_...15...95	215430	847991	2236821	2,26
cnf-r2-b4-k1.2	930282	12464	175575	0,37	IBM_...22...10	18919	77414	596987	0,4
jnh10	100	850	6737	0,02	IBM_...22...15	29833	122814	1249118	0,96
jnh11	100	850	11187	0,02	IBM_...22...20	40753	168249	1845706	1,48
jnh12	100	850	5323	0,01	iso-brn005.shuffled	1130	9866	13572	0,02
jnh13	100	850	4940	0,01	f19-b21-s0-0	746	3517	23805	0,03
jnh14	100	850	3362	0,01	f27-b10-s0-0	193	1113	8268	0,01
jnh15	100	850	7544	0,01	f27-b1-s0-0	193	1113	9401	0,01
jnh18	100	850	16943	0,03	f27-b2-s0-0	193	1113	5614	0,01
jnh19	100	850	10836	0,02	f27-b3-s0-0	193	1113	8716	0,01
jnh202	100	800	4641	0,01	f27-b4-s0-0	193	1113	5992	0,01
jnh203	100	800	18563	0,03	f27-b5-s0-0	193	1113	5626	0,01
jnh208	100	800	16108	0,03	f27-b8-s0-0	193	1113	7702	0,01
jnh20	100	850	8478	0,02	f27-b9-s0-0	193	1113	8684	0,01
jnh211	100	800	3030	0,01	f83-b11-s0-0	1000	43900	318968	0,74
jnh214	100	800	12131	0,02	f83-b14-s0-0	1000	43540	811348	1,61
jnh215	100	800	10558	0,02	f83-b17-s0-0	1000	43900	180456	0,37
jnh216	100	800	12821	0,02	par8-1-c	64	254	5613	0
jnh2	100	850	2201	0	par8-1	350	1149	9224	0
jnh302	100	900	246	0	par8-2	350	1157	7641	0
jnh303	100	900	13452	0,03	par8-4-c	67	266	6216	0
jnh304	100	900	1720	0	par8-4	350	1155	10248	0,01
jnh305	100	900	5348	0,01	par8-5	350	1171	7978	0
jnh307	100	900	2211	0	pitch.boehm	1192	6361	656	0,01
jnh308	100	900	15155	0,03	qg5-10.shuffled	1000	43900	318968	0,69
jnh309	100	900	2460	0,01	qg6-10.shuffled	1000	43540	811348	1,62
jnh310	100	900	3054	0,01	qg7-10.shuffled	1000	43900	180456	0,37
jnh4	100	850	5955	0,01	3col20_5_5.shuffled	40	176	774	0
jnh5	100	850	4151	0,01	3col20_5_6.shuffled	40	176	656	0
jnh8	100	850	4749	0,01	3col20_5_7.shuffled	40	176	903	0
jnh9	100	850	3099	0,01	3col20_5_9.shuffled	40	176	438	0

TAB. 1 – Réduction d'instance SAT en instances polynomiales

## Références

- [1] E. Boros, P.L. Hammer, and X. Sun. Recognition q-horn formulae in linear time. *Journal of Discrete Applied Mathematics*, 55 :1–13, 1994.
- [2] Marco Cadoli and Francesco M. Donini. A survey on knowledge compilation. *AI Communications-The European Journal for Artificial Intelligence*, 10(3-4) :137–150, 1997.
- [3] Mukesh Dalal. Efficient propositional constraint propagation. In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI'92)*, pages 409–414, San-Jose, California (USA), 1992.

Instances CNF	# variables	# clauses	suppression(%)			propagations unitaires	temps en secondes
			clauses	variables	littéraux		
een-tipb-sr06-par1	163647	484831	-94	-95	252004	68362283	38,98
ezfact16_10.shuffled	193	1113	-26	-34	335	5614	0,01
ezfact16_3.shuffled	193	1113	-37	-44	479	5992	0,01
f32-b2-s0-0	40	176	-70	-69	178	941	0
f32-b4-s0-0	40	176	-85	-77	163	919	0
f33-b9-s0-0	80	346	-88	-80	391	5867	0
f6-b2-s2-20	478	1007	-95	-92	532	14216	0
IBM_03_SAT_dat.k30	29079	118925	-44	-55	31075	1393665	1,07
IBM_05_SAT_dat.k10	15399	81447	-87	-93	33203	1252239	0,76
IBM_05_SAT_dat.k20	34863	188452	-74	-82	72024	7798669	5,49
IBM_05_SAT_dat.k25	44598	241982	-67	-75	86760	18851484	16,38
IBM_05_SAT_dat.k30	54333	295512	-60	-67	99477	31503131	26,84
IBM_06_SAT_dat.k15	17501	75616	-43	-49	18130	1278040	1,07
IBM_06_SAT_dat.k20	23826	103226	-71	-78	41764	12961178	10,17
IBM_10_SAT_dat.k15	40278	159501	-33	-35	26022	8285670	6,89
IBM_1_11_SAT_dat.k10	28280	111519	-47	-49	25573	58410957	42,46
IBM_18_SAT_dat.k10	17141	69989	-48	-55	19878	13050828	8,7
IBM_19_SAT_dat.k10	21823	83902	-24	-31	13250	298260	0,26
IBM_19_SAT_dat.k15	34697	134023	-17	-22	14917	508638	0,47
IBM_19_SAT_dat.k20	47577	184178	-17	-23	23258	14607263	12,98
IBM_20_SAT_dat.k10	17567	72087	-36	-41	14004	5226452	3,63
IBM_21_SAT_dat.k10	15919	65180	-35	-39	11897	267966	0,21
IBM_21_SAT_dat.k15	25213	103881	-25	-28	13564	471438	0,39
IBM_21_SAT_dat.k20	34513	142616	-26	-30	21454	9624852	7,38
IBM_22_SAT_dat.k25	51673	213684	-24	-27	28739	30219471	22,32
IBM_23_SAT_dat.k10	18612	76086	-41	-48	16035	69713	0,09
IBM_27_SAT_dat.k10	6477	27070	-62	-70	10054	3826810	2,15
rip08.boehm	471	263	-92	-59	145	8728	0,01
x6dn.boehm	521	1255	-86	-84	1022	137818	0,07

TAB. 2 – Réduction d'instances SAT en instances polynomiales après plusieurs passages

- [4] Mukesh Dalal. An almost quadratic class of satisfiability problems. In W. Wahlster, editor, *Proceedings of the Twelfth European Conference on Artificial Intelligence (ECAI'96)*, pages 355–359, Budapest (Hungary), August 1996. John Wiley & Sons, Ltd.
- [5] S. Darras, G. Dequen, L. Devendeville, B. Mazure, R. Ostrowski, and L. Sais. Using boolean constraint propagation for sub-clause deduction. In *Proceedings of the 11th International Conference on Principles and Practice of Constraint Programming (CP'05)*, pages 757–761, 2005.
- [6] Adnan Darwiche and Pierre Marquis. A knowledge compilation map. *Journal Artificial Intelligence Research (JAIR)*, 17 :229–264, 2002.
- [7] Martin Davis, George Logemann, and Donald Loveland. A machine program for theorem proving. *Journal of the Association for Computing Machinery*, 5 :394–397, 1962.
- [8] Second Challenge on Satisfiability Testing organized by the Center for Discrete Mathematics and Computer Science of Rutgers University, 1993. <http://dimacs.rutgers.edu/Challenges/>.
- [9] William H. Dowling and Jean H. Gallier. Linear-time algorithms for testing satisfiability of propositional horn formulae. *Journal of Logic Programming*, 3 :267–284, 1984.
- [10] 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 Theoretical Computer Science, American Mathematical Society, pages 415–436, 1996.
- [11] Niklas Eén and Armin Biere. Effective preprocessing in sat through variable and clause elimination. In *Proceedings of the Eighth International Conference on Theory and Applications of Satisfiability Testing (SAT'05)*, pages 61–75, 2005.
- [12] Niklas Eén and Niklas Sörensson. An extensible sat-solver. In *Proceedings of the Sixth International Conference on Theory and Applications of Satisfiability Testing (SAT'03)*, pages 502–518, 2003.
- [13] O. Fourdrinoy, É. Grégoire, B. Mazure, and L. Sais. Exploring hybrid algorithms for sat. In *Proceedings of the 12th International Conference on Logic for*

CNF Instance	Minisat		% gain		instance		suppression			#PU	Temps
	Original	Simplifié	partiel	total	#var	#cla	var	cla	#lit		
f2clk_40	293,4	265,19	9	7	27568	80439	36%	36%	13619	5009951	5,52
f3-b29-s0-10	76,72	26,58	65	65	2125	12677	24%	35%	3520	127851	0,18
f28-b4-s0-0	3,15	0,04	98	96	769	4777	13%	22%	927	45554	0,08
f81-b3-s0-0	2081,34	1654,61	20	17	33385	163232	26%	30%	24855	36519004	63,69
fifo8_100	14,31	11,12	22	-48	64762	176313	42%	46%	42718	8435460	9,98
fifo8_200	43,74	77,5	-78	-134	129762	353513	37%	40%	76361	18309357	24,51
fifo8_300	349,92	152,11	56	45	194762	530713	35%	39%	109878	28532439	39,94
fifo8_400	500,73	428,59	14	3	259762	707913	34%	38%	143413	38349604	55,85
IBM_03...k60	28,33	11,99	57	17	59649	244535	22%	27%	33386	12915029	11,33
IBM_03...k90	195,45	173,44	11	1	90219	370145	17%	20%	38507	21481064	18,32
IBM_05...k100	204,54	28,02	86	21	190623	1044932	21%	27%	167316	143847825	133,17
IBM_05...k60	55,59	10,52	81	-37	112743	616692	31%	37%	123076	73459545	65,46
IBM_16...dat.k95	14,62	2,18	85	29	50492	203817	23%	26%	24509	9440470	8,16
ip50	92,63	307,54	-233	-266	66131	214786	36%	44%	47569	23195373	30,61
logistics-rotate-09t6	80,07	6,5	91	-55	8186	887558	15%	30%	908	157186029	117,23

TAB. 3 – Résultat de Minisat avant et après simplification de la formule par isU-Horn

*Programming, Artificial Intelligence and Reasoning (LPAR'05) (short papers)*, pages 33–37, 2005.

- [14] Olivier Fourdrinoy, Éric Grégoire, Bertrand Mazure, and Lakhdar Saïs. Eliminating redundant clauses in sat instances. In *Proceedings of the The Fourth International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems CPAIOR 2007*, pages 122–132, 2007.
- [15] Éric Grégoire, Richard Ostrowski, Bertrand Mazure, and Lakhdar Saïs. Automatic extraction of functional dependencies. In *Proceedings of the Seventh International Conference on Theory and Applications of Satisfiability Testing (SAT'04)*, pages 122–132, 2004.
- [16] Donald E. Knuth. Nested satisfiability. *Acta Informatica*, 28 :1–6, 1990.
- [17] 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.
- [18] 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.
- [19] David Lichtenstein. Planar formulae and their uses. *SIAM Journal of Computing*, 11(2) :329–343, May 1982.
- [20] B. Mazure, L. Saïs, and É. Grégoire. Boosting complete techniques thanks to local search. *Annals of Mathematics and Artificial Intelligence*, 22 :309–322, 1998.
- [21] Bertrand Mazure. *De la Satisfaisabilité à la Compilation de Bases de Connaissances Propositionnelles*. Thèse de doctorat, Université d'Artois, Centre de Recherche en Informatique de Lens (Faculté Jean Perrin), January 1999.
- [22] M.W. Moskewicz, C.F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff : Engineering and efficient sat solver. In *Proceedings of the 38th Design Automation Conference (DAC'01)*, pages 530–535, June 2001.
- [23] Richard Ostrowski, Bertrand Mazure, Lakhdar Saïs, and Éric Gregoire. Eliminating redundancies in SAT search trees. In *Proceedings of the 15th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'2003)*, pages 100–104, Sacramento, November 2003.
- [24] P. Rossa. Formules bien imbriquées : reconnaissance et satisfaisabilité. Mémoire de DEA, Université de Caen, Laboratoire d'Informatique, 1993.
- [25] Bart Selman and Henry A. Kautz. Knowledge compilation and theory approximation. *Journal of the Association for Computing Machinery*, 43(2) :193–224, 1996.
- [26] Bart Selman, Hector J. Levesque, and David Mitchell. Gsat : A new method for solving hard satisfiability problems. In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI'92)*, pages 440–446, 1992.