Suppression de clauses redondantes dans des instances de SAT

Olivier Fourdrinoy, Eric Grégoire, Bertrand Mazure, Lakhdar Saïs

▶ **To cite this version:**

Olivier Fourdrinoy, Eric Grégoire, Bertrand Mazure, Lakhdar Saïs. Suppression de clauses redondantes dans des instances de SAT. Troisièmes Journées Francophones de Programmationpar Contraintes (JFPC07), Jun 2007, INRIA, Domaine de Voluceau, Rocquencourt, Yvelines France. inria-00151033

HAL Id: inria-00151033
https://inria.hal.science/inria-00151033

Submitted on 1 Jun 2007

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Suppression de clauses redondantes dans des instances de SAT

Olivier Fourdrinoy Éric Grégoire Bertrand Mazure Lakhdar Saïs

CRIL CNRS FRE 2499
Université d'Artois, Faculté Jean Perrin
Rue Jean Souvraz
SP 18, F 62307 Lens Cedex
{fourdrinoy,gregoire,mazure,sais}@cril.fr

Résumé

Dans ce papier, nous étudions comment la suppression d'une partie des clauses redondantes d'une instance de SAT permet d'améliorer l'efficacité des solveurs SAT modernes. Le problème consistant à détecter si une instance contient ou non des clauses redondantes est NP-Complet. Nous proposons une méthode de suppression des clauses redondantes incomplète mais polynomiale que nous utilisons comme pré-traitement pour des solveurs SAT. Cette méthode basée sur la propagation unitaire offre des résultats intéressants notamment sur des instances très difficiles issues du monde réel.

Abstract

In this paper, we investigate to which extent the elimination of a class of redundant clauses in SAT instances could improve the efficiency of modern satisfiability provers. Since testing whether a SAT instance does not contain any redundant clause is NP-complete, a logically incomplete but polynomial-time procedure to remove redundant clauses is proposed as a pre-treatment of SAT solvers. It relies on the use of the linear-time unit propagation technique and often allows for significant performance improvements of the subsequent satisfiability checking procedure for really difficult real-world instances.

1 Introduction

Le problème SAT consiste à tester si une formule booléenne mise sous forme normale conjonctive est satisfiable ou non. C'est un problème central dans de nombreux domaines de l'informatique et de l'intelligence artificielle tels que la planification, les raisonnement non monotones, la vérification de circuits, la vérification et la validation de bases de connaissance. Au cours de ces deux dernières décennies, de nombreuses approches ont été proposées pour résoudre des instances SAT difficiles. On divise généralement ces approches en deux catégories : les approches logiquement complètes et les approches incomplètes. Les variantes élaborées de l'algorithme DPLL (pour Davis, Putnam, Loveland et Logemann) [4](e.g. [20, 11]) ainsi que les algorithmes de recherche locale (e.g. [24]) sont aujourd'hui capables de résoudre bon nombre de problèmes très difficiles issus du monde réel.

Récemment, plusieurs auteurs se sont penchés sur la détection d'informations cachées dans les instances issues du monde réel (e.g backbones [8], backdoors [26], équivalences [16] et dépendances fonctionnelles [13]), permettant ainsi d'expliquer et d'améliorer l'efficacité des solveurs SAT sur certaines instances très difficiles. En particulier, le codage sous forme normale conjonctive peut amener à cacher les informations structurales du problème initial [13]. Plus généralement, il apparaît que bon nombre de problèmes issus du monde réel comportent des informations redondantes qui pourraient être supprimées sans risque. Les instances ainsi simplifiées restent équivalentes mais peuvent devenir plus faciles à résoudre.

Dans ce papier, nous étudions la possibilité de supprimer une partie des clauses redondantes des instances issues du monde réel dans le but d'améliorer l'efficacité des solveurs modernes sur ces instances. Une clause redondante est une clause qui peut être supprimée d'une instance tout en gardant la possibilité de la déduire du reste de l'instance. Si le test consistant à savoir si une instance SAT contient ou non des clauses redondantes est NP-complet[16], nous propo-

sons un algorithme incomplet mais polynomial afin de supprimer une partie des clauses redondantes, et ce, en pré-traitement des solveurs modernes. Nous utilisons pour ce faire la propagation unitaire qui est une technique linéaire en temps. Il est intéressant de constater que ce pré-traitement permet régulièrement d'améliorer de façon importante les performances des solveurs sur des instances issues du monde réel.

Le reste du papier est organisé comme suit. Après avoir introduit quelques bases de logiques et quelques concept liés à SAT dans la section 2, nous présentons la notion de redondance dans SAT ainsi que la notion de redondance modulo la propagation unitaire dans la section 3. Dans la section 4, nos résultats expérimentaux sont présentés et analysés. Les travaux en relation avec notre approche sont discutés dans la section 5 avant de conclure par quelques remarques et quelques perspectives concernant nos futurs travaux dans la dernière section.

2 Notions techniques

Soit $\mathcal{L}$ un langage logique standard construit sur un ensemble fini de variables booléennes, notées x, y, etc. Les formules seront notées par des lettres telles que c. Les ensembles de formules seront représentées par des lettre grecques comme Γ ou Σ. Une interprétation est une fonction qui à chaque variable booléenne affecte une valeur issue de $\{vrai, faux\}$. Une formule est satisfiable ou consistante quand il existe au moins une interprétation qui la satisfait, c.-à-d. qui la rend *vraie*. Une telle interprétation est appelée modèle de l'instance. Une interprétation sera représentée par une lettre majuscule comme I et sera représentée par l'ensemble des littéraux qu'elle satisfait. Concrètement, toute formule dans $\mathcal{L}$ peut être représentée (tout en préservant la satisfiabilité) en utilisant un ensemble (interprété comme une conjonction) de clauses, où une clause est une disjonction finie de littéraux, où un littéral est une variable booléenne signée. Une clause sera également représentée par l'ensemble de ses littéraux. En conséquence, la taille d'une clause c correspond au nombre de littéraux qu'elle contient, et elle est notée $|c|$.

SAT est le problème NP-complet[2] qui consiste à tester si un ensemble fini de clauses booléennes de $\mathcal{L}$ est satisfiable ou non, i.e. si il existe une interprétation qui satisfait l'ensemble des clauses ou non.

La déduction logique sera notée $\models$: soit c une clause de $\mathcal{L}$, $\Sigma \models c$ ssi c est *vraie* dans tous les modèles de Σ. La clause vide qui représente l'inconsistance est notée $\bot$.

Par la suite, nous faisons plusieurs fois référence aux fragments binaire et Horn de $\mathcal{L}$ pour lesquels le problème SAT peut être résolu en temps polynomial [25, 9, 6]. Une clause binaire est une clause constituée d'au plus deux littéraux alors qu'une clause de Horn est une clause contenant au plus un littéral positif. Une clause unitaire est une clause ne contenant qu'un seul littéral.

Dans ce papier, l'algorithme de la *Propagation Unitaire* (ou PU) joue un rôle central. PU simplifie récursivement une instance SAT en propageant -à travers l'instance- les valeurs de vérité des clauses unitaires dont les variables sont déjà affectées, comme on peut le voir dans l'algorithme 1.

Nous définissons la déduction modulo la Propagation Unitaire, notée $\models^*$, la relation de déduction $\models$ limitée à la technique de Propagation Unitaire.

Définition 1 *Soient Σ une instance SAT et c une clause de $\mathcal{L}$, $\Sigma \models^* c$ si et seulement si $\Sigma \wedge \neg c \models^* \bot$ si et seulement si $PU(\Sigma \wedge \neg c)$ contient la clause vide.*

Il est clair que $\models^*$ est logiquement incomplet. Comme PU est un processus linéaire en temps, $\models^*$ peut être testé en temps polynomial.

Soient c_1 et c_2 deux clauses de $\mathcal{L}$. Quand $c_1 \subseteq c_2$, nous avons $c_1 \models c_2$ et on dit que c_1 (resp. c_2) subsume (resp. est subsumée par) c_2 (resp. c_1). Une clause $c_1 \in \Sigma$ est subsumée dans Σ si et seulement si il existe $c_2 \in \Sigma$ telle que $c_1 \neq c_2$ et c_2 subsume c_1. Σ est clos par subsumption si et seulement si quelque soit $c \in \Sigma$, c n'est pas subsumée dans Σ.

3 Redondances dans les instances SAT

Intuitivement, une instance SAT est redondante si elle contient des sous-ensembles de clauses qui peuvent être logiquement inférés du reste de la formule. Supprimer les parties redondantes d'une instance SAT dans le but d'améliorer le test de satisfiabilité pose au moins deux questions fondamentales.

- Tout d'abord, il n'est pas évident que de telles suppressions rendent le test de satisfiabilité plus efficace. La redondance de certaines informations peut accélérer le processus de résolution des solveurs (en donnant par exemple du poids à certaines informations plutôt qu'à d'autres).
- De plus, il est bien connu que le test d'irredondance sur une formule SAT est NP-complet [16]. Il est donc aussi difficile de tester cette irredondance que de résoudre l'instance elle-même.

Dans le but de résoudre ces problèmes, nous considérons un algorithme incomplet qui permet de détecter certaines clauses redondantes tout en restant polynomial. Intuitivement, nous n'allons pas chercher à supprimer toutes les clauses redondantes. Certains types

Input: une instance SAT Σ
Output: Une instance SAT Γ équivalente à Σ du point de vue de la satisfiabilité t.q. Γ ne contient pas de clauses unitaires

```
1  begin
2      if Σ contient la clause vide then
3          return Σ;
4      else
5          if Σ contient une clause unitaire c = {l} then
6              forall c ∈ Σ t.q. l ∈ c do
7                  Σ ⟵ Σ \ {c}
8              forall c ∈ Σ t.q. ¬l ∈ c do
9                  c ⟵ c \ {¬l}
10             return Propagation_Unitaire(Σ);
11         else
12             return Σ;
13 end
```

Algorithm 1 – Propagation_Unitaire

de clauses ont la particularité de faciliter le test de satisfiabilité en cela qu'elles appartiennent à des classes polynomiales de SAT, en particulier les clauses binaires et les clauses de Horn. En conséquence, nous proposons une approche à deux niveaux : d'une part, nous lançons notre processus de détection et de suppression des clauses redondantes qui est à la fois rapide et incomplet. D'autre part, nous regardons s'il est pertinent ou non de supprimer les clauses binaires et les clauses de Horn.

Définition 2
Soient Σ une instance SAT et $c \in \Sigma$, c est redondante dans Σ si et seulement si $\Sigma \setminus \{c\} \models c$.

Clairement, la redondance peut être testée en utilisant une procédure de réfutation. En effet, c est redondante dans Σ si et seulement si $\Sigma \setminus \{c\} \cup \{\neg c\} \models \bot$. Nous affaiblissons cette procédure de réfutation en remplaçant $\models$ par $\models^*$ dans le but d'obtenir une approche du test de la redondance incomplète mais polynomiale en temps.

Définition 3
Soient Σ une instance SAT et $c \in \Sigma$, c est PU-redondant dans Σ si et seulement si $\Sigma \setminus \{c\} \models^* c$.

Par conséquent, tester la PU-redondance de c dans Σ revient à propager l'opposé de chaque littéral de c dans $\Sigma \setminus \{c\}$.

Considérons l'exemple 1 décrit ci-après. Dans cet exemple, il est facile de montrer que la clause $w \vee x$ est PU-redondante dans Σ, bien qu'elle ne soit pas subsumée dans Σ. Considérons $\Sigma \setminus \{w \vee x\} \wedge \neg w \wedge \neg x$. Clairement, $w \vee \neg y$ et $x \vee \neg z$ permettent respectivement de déduire $\neg y \wedge \neg z$. La propagation de ces deux littéraux génère une contradiction, montrant ainsi que $w \vee x$ est PU-redondante dans Σ. Par contre, $w \vee x$ n'est pas subsumée dans Σ puisqu'il n'existe pas de clause $c' \in \Sigma$ telle que $c' \subseteq c$.

Exemple 1

$$\Sigma = \begin{cases} w \vee x \\ y \vee z \\ w \vee \neg y \\ x \vee \neg z \\ \cdots \end{cases}$$

Dans l'algorithme 2, un pré-traitement PU (basique) est décrit et peut être expliqué comme suit. Dans le cas général, il existe un nombre exponentiel de formules irredondantes différentes qui peuvent être extraites d'une instance. En effet, irredondance et inconsistance minimale coïncident dans les formules insatisfiables[16]. Concrètement, l'instance simplifiée générée par l'algorithme 1 dépend de l'ordre dans lequel sont traitées les clauses de Γ. Comme les clauses de petite taille réduisent l'espace de recherche de façon plus efficace, nous avons implémenté une politique qui teste les plus grandes clauses d'abord. Par conséquent, cela revient à considérer les clauses de l'ensemble Γ suivant l'ordre décroissant de leurs tailles.

Exemple 2 *Soit Σ l'instance SAT ci-dessous :*

$$\Sigma = \begin{cases} w \vee x \\ w \vee x \vee y \vee z \\ w \vee \neg y \\ x \vee \neg z \end{cases}$$

Input: une instance SAT Σ
Output: une instance SAT PU-irredondante Γ équivalente à Σ du point de vue de la satisfiabilité

```
1 begin
2   Γ ⟵ Σ ;
3   forall clauses c = {l_1, ..., l_n} ∈ Σ ordonnées suivant leurs tailles décroissantes do
4     if PU(Γ \ {c} ∪ {¬l_1} ∪ ... ∪ {¬l_n}) contient la clause vide then
5       Γ ⟵ Γ \ {c} ;
6   return Γ;
7 end
```

Algorithm 2 – Génération d'une formule PU-irredondante

– *Si l'on ne tient pas compte de la taille et que l'on commence par la clause $w \vee x$, l'ensemble PU-irredondant est le suivant :*

$$\Gamma_1 = \begin{cases} w \vee x \vee y \vee z \\ w \vee \neg y \\ x \vee \neg z \end{cases}$$

– *En utilisant l'algorithme 2, qui teste $w \vee x \vee y \vee z$ en premier, nous obtenons un ensemble résultat de même taille (en terme de nombre de clauses) mais différent.*

$$\Gamma_2 = \begin{cases} w \vee x \\ w \vee \neg y \\ x \vee \neg z \end{cases}$$

En commençant par les plus grandes clauses, nous obtenons le plus petit Γ en nombres de clauses mais aussi en nombres de littéraux. Cet ordre garantit par ailleurs que toutes les clauses subsumées dans Σ sont supprimées et que seules les clauses subsumantes sont préservées. Comme l'illustre cet exemple, les clauses subsumées sont supprimées, conduisant ainsi à des clauses plus courtes dans Γ, qui est ainsi plus contraint, et dans une certaine mesure, plus facile à résoudre.

Propriété 1 *Soit Σ une instance SAT. Si Σ' est une formule obtenue depuis Σ en appliquant l'algorithme 2 alors Σ' est clos par subsumption.*

Preuve 1 *Supposons qu'il existe deux clauses c et c' de Σ' telles que c' subsume c. Nous pouvons déduire que $|c'| \leq |c|$. Comme les clauses de Σ testées par PU-redondance sont ordonnées en fonction de leurs tailles décroissantes, nous déduisons que c est PU-redondante. En conséquence, $c \notin \Sigma'$, ce qui contredit l'hypothèse.* □

En revanche, l'inverse de la propriété 1 est faux. En effet, la formule $\Sigma = \Gamma_1 \cup \{(y \vee e), (z \vee \neg e)\}$ est close par subsumption mais n'est par PU-irredondante.

Dans le meilleur des cas, le pré-traitement nous débarrasse de toutes les clauses non-polynômiales, et plonge ainsi l'instance dans un fragment polynômial. Devant la taille parfois énorme des instances, nous étudions la possibilité de protéger ou non du test de PU-redondance les fragments polynômiaux de $\mathcal{L}$. Dans notre étude, plusieurs fragments possibles ont été considérés pour la vérification de PU-redondance : à savoir Σ, toutes les clauses non Horn de Σ, toutes les clauses non-binaires de Σ, et toutes les clauses de non-Horn et non-binaires de Σ.

Enfin, nous avons expérimenté une approche visant à maximiser le nombre de propagations unitaires. L'idée est la suivante. Une clause qui contient des littéraux qui occurrent dans un grand nombre de clauses binaires conduira nécessairement à une cascade de propagations unitaires. Par exemple, soit $c = x \vee y \vee z$. Quand x apparaît dans plusieurs clauses binaires et que nous testons si c est redondante en utilisant la propagation unitaire, toutes ces clauses binaires seront réduites à l'état de clauses unitaires menant récursivement à d'autres propagations. Nous définissons donc un poids w associé à la clause c et noté $w(c)$ qui correspond à la somme des poids de chaque littéraux de c, le poids d'un littéral étant le nombre de clauses binaires auxquelles il appartient. Il est à noter que pour avoir une propagation unitaire lors du test de PU-redondance d'une clause $c_1 \in \Sigma$, il faut qu'il y ait une autre clause $c_2 \in \Sigma$ t.q. $|c_2 - \{c_1 \cap c_2\}| = 1$. Concrètement, quand $|c_2 - \{c_1 \cap c_2\}| = 0$, c_1 est PU-redondante. Comme le calcul et la détection de cette condition nécessaire peut être coûteuse en temps, nous avons développé un critère de poids moins restrictif mais plus facile à calculer. Si c_2 est une clause binaire, la condition précédente est satisfaite si seulement si c_1 (dont nous souhaitons tester la redondance) contient un littéral de c_2. Par conséquent, quand $w(c) = 0$, c n'est pas testée pour la redondance. Cette heuristique de pondération a été mixée avec les politiques liées aux classes polynomiales (binaire, Horn, binaire et Horn).

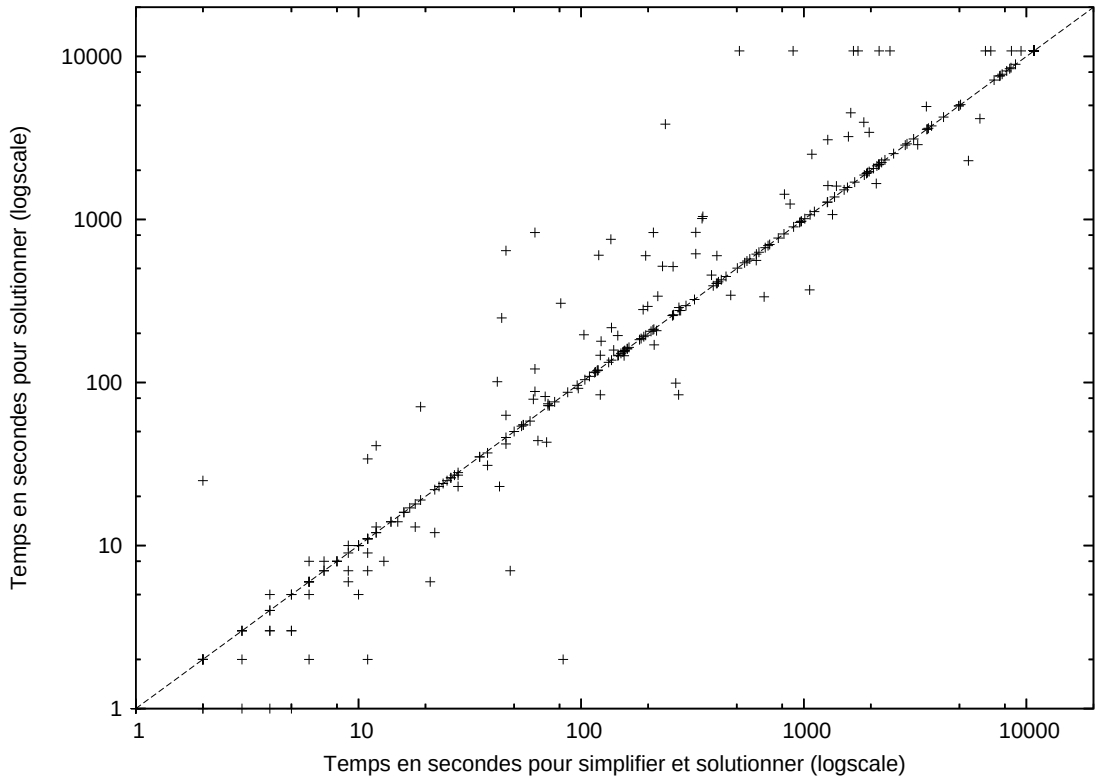

FIG. 1 – Resultats sur les 700 instances testées

4 Résultats expérimentaux

Nous avons testé les différents pré-traitements sur un panel de près de 700 instances SAT issues des dernières compétitions SAT (`www.satcompetition.org`). Ces instances ont été choisies parmi différentes catégories de problèmes (industriels, aléatoires, hand-made, coloration de graphe, etc.). Nous avons utilisé trois solveurs choisis parmi les vainqueurs des trois dernières compétitions : ZChaff, Minisat et SatELite. Toutes nos expérimentations ont été conduites sur des Pentium IV, 3Ghz sous linux Fedora Core 4.

Dans un premier temps, nous avons lancé les trois solveurs sur toutes les instances afin de déterminer leur temps de résolution de référence sur chaque instance. Nous avons fixé un temps limite de 3 heures. Ensuite nous avons lancé nos différents pré-traitements à base de PU sur ces instances. Nous avons collecté à la fois les temps de simplification et les temps de résolution des solveurs sur les instances ainsi simplifiées. Plus précisément, nous avons expérimenté différents pré-traitements à base de PU. Pour le premier, nous avons appliqué la redondance modulo PU sur toutes les clauses. Pour les suivants, les clauses non-binaires et les clauses non-Horn ont été testées successivement. Nous avons également attaqué ces clauses ensemble. Enfin, toutes ces expérimentations ont été répétées en ajoutant cette fois l'heuristique de pondération avec $w(c) > 0$.

Dans Fig.1, nous montrons le gain sur les 700 instances testées. Sur l'axe des abscisses (x), nous représentons le temps mis pour simplifier et résoudre une instance dans le meilleur des cas. Sur l'axe des ordonnées (y), nous représentons le meilleur temps de résolution obtenu avant simplification. Par conséquent, la diagonale qui part de l'origine représente l'absence de gain. Les instances qui sont au dessus de cette ligne bénéficient de notre algorithme. Cette figure montre clairement que notre technique obtient de meilleurs résultats sur les instances difficiles dont la résolution requiert un temps CPU important. En effet, pour ces instances, nous obtenons assez souvent des gains importants. En particulier, tous les points alignés à 10800 secondes sur l'axe des ordonnées représentent des instances SAT non résolues dans le temps imparti sans notre méthode de simplification basée sur la redondance modulo PU.

Sans surprise, les résultats montrent que la méthode de simplification appliquée à toutes les clauses consomme plus de ressource que les différentes restrictions étudiées. Sans surprise également, les instances simplifiées qui en résultent sont les plus petites. Cepen-

Instances	nom court	#C	#V
gensys-icl004.shuffled-as.sat05-3825.cnf	gensys	15960	2401
rand_net60-30-5.miter.shuffled.cnf	rand_net	10681	3000
f3-b25-s0-10.cnf	f3-b25	12677	2125
ip50.shuffled-as.sat03-434.cnf	ip50	214786	66131
f2clk_40.shuffled-as.sat03-424.cnf	f2clk	80439	27568
f15-b3-s3-0.cnf	f15-b3	469519	132555
IBM_FV_2004_rule_batch_23_SAT_dat.k45.cnf	IBM_k45	381355	92106
IBM_FV_2004_rule_batch_22_SAT_dat.k70.cnf	IBM_k70	327635	63923
f15-b2-s2-0.cnf	f15-b2	425316	120367
IBM_FV_2004_rule_batch_22_SAT_dat.k75.cnf	IBM_k75	979230	246053
okgen-c2000-v400-s553070738-553070738.cnf	okgen	2000	400

TAB. 1 – Sélection d'instances typiques choisies parmi les 700 testées au cours de nos expérimentations tous les résultats sont disponibles sur `www.cril.fr/~fourdrinoy/eliminating_redundant_clauses.html`

dant, ce gain en taille ne conduit pas systématiquement à une amélioration du temps de résolution (en tenant compte du temps de simplification ou non). En effet, certaines clauses, notamment celles appartenant aux fragments polynomiaux permettent parfois -bien qu'elles soient redondantes- d'accélérer la résolution.

Globalement, nos expérimentations montrent que la meilleure stratégie de simplification consiste à tester toutes les clauses (polynomiales ou non) ayant un poids $w(c) > 0$.

Dans les tables 1 à 5, nous présentons un échantillon représentatif et détaillé des résultats obtenus au cours des différentes expérimentations susmentionnées. Dans la table 1, chaque instance se voit affecter un nom court afin de clarifier la présentation des résultats. La taille des instances y est ensuite présentée en nombre de clauses ($\#C$) ainsi qu'en nombre de variables ($\#V$). Dans la table 2, on peut consulter le temps de simplification en secondes (T_s) ainsi que la réduction de l'instance exprimée à la fois en nombre de clauses ($\#c_r$) et en pourcentage. La colonne intitulée « Pas de Restriction » présente les résultats obtenus pour une politique testant la redondance de toutes les clauses. Dans les colonnes suivantes, le test est restreint aux clauses non-$Horn$, puis non-$Binaire$, et enfin non-$Horn$ et non-$Binaire$. La partie basse du tableau correspond à ces mêmes politiques augmentées de l'heuristique de pondération.

Dans les tables 3 à 5, nous observons les temps de résolution pour ses mêmes instances en utilisant respectivement Zchaff, Minisat et SatElite. TO signifie "Time-Out" et correspond à un dépassement du temps imparti à la résolution des instances (ndlr : 3heures). Dans la première colonne, T_b est le temps de base mis par le solveur pour résoudre l'instance initiale. Dans les colonnes suivantes, le temps de résolution de l'instance simplifiée (T_r) est accompagné des gains relatifs au temps de base défini en première colonne. $\%_p$ et $\%_g$ correspondent aux gains partiel ou global, c.-à-d. qui tiennent compte ou non du temps de simplification. Pour chacune des instances, les meilleurs temps obtenus avant et après simplification sont en caractères gras. Lorsque ce temps est le meilleur (avec ou sans simplications), il est écrit en caractères de taille supérieure. À noter que si les 9 premières instances obtiennent des résultats positifs, l'instance *IBM_k75* voit son temps de résolution dégradé par notre prétraitement. L'instance *okgen* quant à elle est représentative du comportement global des instances alétoires face à notre prétraitement. En effet, si les instances industrielles ou hand-made comportent plus ou moins de clauses PU-redondantes, il est à noter que la plupart des instances aléatoires en sont quasiment dépourvues.

5 Liens avec les travaux existant

Introduire une étape de pré-traitement rapide - polynomiale en temps - à l'intérieur de solveurs logiquement complets n'est pas une idée nouvelle en soit. Principalement, C-SAT [7] proposait un pré-traitement à base de résolution bornée calculant toutes les résolvantes de tailles plus petite que la taille de leur parents. À son époque, C-SAT fut le meilleur solveur sur les instances aléatoires k-SAT. Satz utilise la même technique, mais avec une taille de résolvante limitée à trois [15]. Récemment, Niklas Eén et Armin Biere ont introduit une méthode d'élimination de variable basée sur la subsumption, les résolvantes subsumantes et l'élimination de variables par substitution [10] comme une étape de pré-traitement pour les solveurs SAT modernes, étendant ainsi le précédent pré-traitement NiVER [22]. Dans [27], un algorithme permet de maintenir la non-subsumption d'un ensemble de clauses au fur et à mesure que de nouvelles clauses sont ajoutées.

instances	Pas de Restriction T_s	Pas de Restriction $\#c_r$ (%)	Horn T_s	Horn $\#c_r$ (%)	Binaire T_s	Binaire $\#c_r$ (%)	Horn & Binaire T_s	Horn & Binaire $\#c_r$ (%)
gensys	1.26	2861 (17.92)	1.18	2208 (13.83)	1.21	2560 (16.04)	1.13	2171 (13.60)
rand_net	1.80	608 (5.69)	0.62	178 (1.66)	0.62	0 (0.00)	0.30	0 (0.00)
f3-b25	1.66	1502 (11.84)	1.04	926 (7.30)	1.64	1502 (11.84)	1.03	926 (7.30)
ip50	65.06	1823 (0.84)	22.02	504 (0.23)	14.11	194 (0.09)	9.10	110 (0.05)
f2clk	6.39	344 (0.42)	2.08	119 (0.14)	1.29	77 (0.09)	0.75	54 (0.06)
f15-b3	359.52	55816 (11.88)	116.73	14167 (3.01)	55.38	1010 (0.21)	37.62	640 (0.13)
IBM_k45	53.26	32796 (8.59)	10.33	2122 (0.55)	6.22	717 (0.18)	5.03	715 (0.18)
IBM_k70	36.68	22720 (6.93)	5.36	4628 (1.41)	3.81	0 (0.00)	2.78	0 (0.00)
f15-b2	306.06	50717 (11.92)	100.14	12909 (3.03)	47.74	979 (0.23)	34.00	609 (0.14)
IBM_k75	172.47	116841 (11.93)	40.14	5597 (0.57)	24.64	3912 (0.39)	22.34	3911 (0.39)
okgen	0.00	0 (0.00)	0.00	0 (0.00)	0.00	0 (0.00)	0.00	0 (0.00)

Heuristique $w(c) > 0$

	Pas de Restriction T_s	Pas de Restriction $\#c_r$ (%)	Horn T_s	Horn $\#c_r$ (%)	Binaire T_s	Binaire $\#c_r$ (%)	Horn & Binaire T_s	Horn & Binaire $\#c_r$ (%)
gensys	0.65	2560 (16.04)	0.63	2171 (13.60)	0.64	2560 (16.04)	0.64	2171 (13.60)
rand_net	1.66	514 (4.81)	0.58	148 (1.38)	0.62	0 (0.00)	0.30	0 (0.00)
f3-b25	0.21	60 (0.47)	0.15	44 (0.34)	0.21	60 (0.47)	0.14	44 (0.34)
ip50	53.95	1823 (0.84)	19.43	504 (0.23)	14.24	194 (0.09)	9.21	110 (0.05)
f2clk	6.06	267 (0.33)	1.96	100 (0.12)	1.30	77 (0.09)	0.80	54 (0.06)
f15-b3	229.84	24384 (5.19)	83.46	6393 (1.36)	55.69	1010 (0.21)	37.72	640 (0.13)
IBM_k45	34.53	11049 (2.89)	10.36	2122 (0.55)	6.23	717 (0.18)	4.98	715 (0.18)
IBM_k70	15.25	11464 (3.49)	5.36	4616 (1.40)	3.77	0 (0.00)	2.77	0 (0.00)
f15-b2	209.55	22217 (5.22)	77.22	5709 (1.34)	51.04	979 (0.23)	33.32	609 (0.14)
IBM_k75	125.26	39640 (4.04)	38.15	5597 (0.57)	26.02	3912 (0.39)	22.32	3911 (0.39)
okgen	0.00	0 (0.00)	0.00	0 (0.00)	0.00	0 (0.00)	0.00	0 (0.00)

TAB. 2 – Temps de simplification et tailles de réduction

Un sujet de recherche intéressant consisterait à comparer l'ensemble de ces pré-traitements tant du point de vue théorique que du point de vue expérimental.

En raison de sa linéarité temporelle, l'algorithme de propagation unitaire a été exploité dans de nombreuses voies autour de SAT. Il s'agit par ailleurs d'un élément clé des procédures de type DPLL. Par exemple, C-SAT et Satz utilisent un traitement local basé sur la PU pendant une grande partie de l'exploration de l'espace de recherche. En effet la PU y est utilisée pour dériver les littéraux impliqués et pour détecter ainsi des inconsistances locales et guider le choix de la prochaine variable à affecter [7, 15]. Dans [14], un schéma basé sur la double propagation unitaire est étudié pour résoudre le problème SAT. Dans [21, 13], la PU est utilisée pour détecter les dépendances fonctionnelles dans les instances SAT. La technique de PU est également exploitée dans [3] pour dériver des sous-clauses en utilisant le graphe d'implication de l'instance et permet d'accélérer le processus de résolution.

Bailleux, Roussel et Boufkhad ont travaillé sur l'impact des clauses redondantes sur la résolution des instances aléatoires k-sat [1]. Cependant, leur travail se focalise sur les instances aléatoires et ne peut s'appliquer aux instances issues du monde réel. De plus, ils étudient la redondance et non la PU-redondance ; leur objectif étant de mesurer le degré de redondance des instances aléatoires 3-SAT au seuil. Du point de vue la complexité, une étude complète de la redondance dans le contexte booléen est proposée dans [17].

Dans une certaine mesure, notre travail se rapproche des techniques de compilation [23, 5, 18, 19] dont le but est de transformer des formules booléennes en des formules équivalentes quoique plus facile à tester. L'idée est de passer plus de temps dans le pré-traitement afin de plonger l'instance dans un fragment polynomial de $\mathcal{L}$. De même, notre approche tend à réduire la taille des fragments non polynomiaux de l'instance. Cependant, cette réduction est partielle car toutes les clauses appartenant au fragment non polynomial ne sont pas nécessairement supprimées. De plus, quand les techniques de compilation s'octroient la possibilité de recourir à un temps exponentiel, nous garantissons que notre pré-traitement se fait en temps polynomial.

6 Conclusions

Supprimer les clauses redondantes dans une instance SAT au cours d'une étape de pré-traitement dans le but d'accélérer le test de satisfiabilité de cette même instance n'est pas chose aisée. En effet, le test de redondance est intraitable dans le pire des cas, et certaines informations redondantes permettent parfois d'accélérer le test de satisfiabilité. Dans ce papier, nous proposons un algorithme efficace quoique incomplet utilisant la propagation unitaire comme pré-

instances	T_b	Pas de Restriction T_r	($\%_p$,	$\%_g$)	Horn T_r	($\%_p$,	$\%_g$)	Binaire T_r	($\%_p$,	$\%_g$)	Horn & Binaire T_r	($\%_p$,	$\%_g$)
gensys	**3418.17**	2847.1	(16.70,	16.66)	3353.69	(1.88,	1.85)	1988.37	(41.82,	41.79)	3683.58	(-7.76,	-7.79)
rand_net	(1334.19)	942.15	(29.38,	29.24)	1067.01	(20.02,	19.97)	1334.19	(0.00,	-0.04)	1334.19	(0.00,	-0.02)
f3-b25	(790.96)	137.26	(82.64,	82.43)	155.32	(80.36,	80.23)	**134.91**	(82.94,	82.73)	157.44	(80.09,	79.96)
ip50	(2571.18)	675.11	(73.74,	71.21)	474.64	(81.53,	80.68)	1023.82	(60.18,	59.63)	1945.87	(24.31,	23.96)
f2clk	(6447.19)	4542.32	(29.54,	29.44)	9457.25	(-46.68,	-46.72)	3978.14	(38.29,	38.27)	3848.02	(40.31,	40.30)
f15-b3	(7627.95)	5620.25	(26.32,	21.60)	2926.38	(61.63,	60.10)	10157.9	(-33.16,	-33.89)	2419.52	(68.28,	67.78)
IBM_k45	(5962.46)	2833.1	(52.48,	51.59)	3656.05	(38.68,	38.50)	3244.61	(45.58,	45.47)	4751.79	(20.30,	20.22)
IBM_k70	(TO)	514.83	(∞,	∞)	5377.91	(∞,	∞)	TO	(–,	–)	TO	(–,	–)
f15-b2	(TO)	2287.73	(∞,	∞)	8891.1	(∞,	∞)	TO	(–,	–)	3969.27	(∞,	∞)
IBM_k75	(TO)	TO	(–,	–)	TO	(–,	–)	TO	(–,	–)	TO	(–,	–)
okgen	(1309.66)	1309.66	(0.00,	-0.00)	1309.66	(0.00,	-0.00)	1309.66	(0.00,	-0.00)	1309.66	(0.00,	-0.00)

Heuristique $w(c) > 0$

instances	T_b	Pas de Restriction T_r	($\%_p$,	$\%_g$)	Horn T_r	($\%_p$,	$\%_g$)	Binaire T_r	($\%_p$,	$\%_g$)	Horn & Binaire T_r	($\%_p$,	$\%_g$)
gensys	**3418.17**	1986.98	(41.87,	41.85)	3896.93	(-14.00,	-14.02)	**1967.22**	(42.44,	42.42)	3873.89	(-13.33,	-13.35)
rand_net	(1334.19)	555.13	(58.39,	58.26)	614.75	(53.92,	53.87)	1334.19	(0.00,	-0.04)	1334.19	(0.00,	-0.02)
f3-b25	(790.96)	839.54	(-6.14,	-6.16)	811.37	(-2.58,	-2.59)	791.97	(-0.12,	-0.15)	813.05	(-2.79,	-2.81)
ip50	(2571.18)	708.81	(72.43,	70.33)	465.13	(81.90,	81.15)	1091.54	(57.54,	56.99)	1958	(23.84,	23.48)
f2clk	(6447.19)	5196.02	(19.40,	19.31)	5766.78	(10.55,	10.52)	4042.8	(37.29,	37.27)	3965.68	(38.48,	38.47)
f15-b3	(7627.95)	TO	(−∞,	−∞)	TO	(−∞,	−∞)	10024	(-31.41,	-32.14)	2448.27	(67.90,	67.40)
IBM_k45	(5962.46)	4447.15	(25.41,	24.83)	3698.1	(37.97,	37.80)	3283.2	(44.93,	44.83)	4925.58	(17.39,	17.30)
IBM_k70	(TO)	4131.58	(∞,	∞)	5564.5	(∞,	∞)	TO	(–,	–)	TO	(–,	–)
f15-b2	(TO)	4456.24	(∞,	∞)	2880.15	(∞,	∞)	TO	(–,	–)	4028.04	(∞,	∞)
IBM_k75	(TO)	TO	(–,	–)	TO	(–,	–)	TO	(–,	–)	TO	(–,	–)
okgen	(1309.66)	1309.66	(0.00,	0.00)	1309.66	(0.00,	-0.00)	1309.66	(0.00,	-0.00)	1309.66	(0.00,	-0.00)

TAB. 3 – Résultats avec Zchaff

instances	T_b	Pas de Restriction T_r	($\%_p$,	$\%_g$)	Horn T_r	($\%_p$,	$\%_g$)	Binaire T_r	($\%_p$,	$\%_g$)	Horn & Binaire T_r	($\%_p$,	$\%_g$)
gensys	(7543.42)	4357.66	(42.23,	42.21)	7078.84	(6.15,	6.14)	4722.35	(37.39,	37.38)	7370.48	(2.29,	2.27)
rand_net	**41.15**	**11.00**	(73.26,	68.87)	35.12	(14.66,	13.14)	41.15	(0,	-1.52)	41.15	(0.00,	-0.73)
f3-b25	**755.90**	225.45	(70.17,	69.95)	246.97	(67.32,	67.18)	233.73	(69.07,	68.86)	243.39	(67.80,	67.66)
ip50	(88.43)	61.95	(29.94,	-43.62)	138.90	(-57.06,	-81.97)	64.47	(27.09,	11.13)	60.13	(31.99,	21.70)
f2clk	**280.88**	290.41	(-3.39,	-5.66)	**188.53**	(32.87,	32.13)	325.87	(-16.01,	-16.48)	274.77	(2.17,	1.90)
f15-b3	(875.37)	531.31	(39.30,	-1.76)	561.40	(35.86,	22.53)	759.43	(13.24,	6.91)	555.47	(36.54,	32.24)
IBM_k45	**3940.82**	3729.01	(5.37,	4.02)	3568.43	(9.44,	9.18)	3625.13	(8.01,	7.85)	3541.57	(10.13,	10.00)
IBM_k70	**643.67**	76.16	(88.16,	82.46)	527.58	(18.03,	17.20)	643.67	(0,	-0.59)	643.67	(0.00,	-0.43)
f15-b2	**516.36**	334.07	(35.30,	-23.96)	257.93	(50.04,	30.65)	452.94	(12.28,	3.03)	369.69	(28.40,	21.81)
IBM_k75	(4035.72)	5096.73	(-26.29,	-30.56)	6324.08	(-56.70,	-57.69)	5782.49	(-43.28,	-43.89)	5534.54	(-37.13,	-37.69)
okgen	**46.43**	46.43	(0.00,	-0.02)	46.43	(0.00,	-0.01)	46.43	(0.00,	-0.01)	46.43	(0.00,	-0.01)

Heuristique $w(c) > 0$

instances	T_b	Pas de Restriction T_r	($\%_p$,	$\%_g$)	Horn T_r	($\%_p$,	$\%_g$)	Binaire T_r	($\%_p$,	$\%_g$)	Horn & Binaire T_r	($\%_p$,	$\%_g$)
gensys	(7543.42)	4742.55	(37.12,	37.12)	7434.11	(1.44,	1.44)	4615.46	(38.81,	38.80)	7610.16	(-0.88,	-0.89)
rand_net	**41.15**	27.00	(34.38,	30.33)	25.60	(37.79,	36.36)	41.15	(0,	-1.52)	41.15	(0.00,	-0.73)
f3-b25	**755.90**	745.80	(1.33,	1.30)	738.83	(2.25,	2.23)	773.19	(-2.28,	-2.31)	779.37	(-3.10,	-3.12)
ip50	(88.43)	54.28	(38.61,	-22.39)	138.28	(-56.35,	-78.33)	67.17	(24.04,	7.93)	**52.71**	(40.39,	29.97)
f2clk	**280.88**	224.82	(19.95,	17.79)	221.20	(21.24,	20.54)	299.23	(-6.53,	-6.99)	289.93	(-3.22,	-3.50)
f15-b3	(875.37)	543.86	(37.87,	11.61)	687.06	(21.51,	11.97)	751.79	(14.11,	7.75)	498.80	(43.01,	38.70)
IBM_k45	**3940.82**	**1826.6**	(53.64,	52.77)	3324.82	(15.63,	15.36)	3637.35	(7.70,	7.54)	3714.29	(5.74,	5.62)
IBM_k70	**643.67**	**31.51**	(95.10,	92.73)	518.48	(19.44,	18.61)	643.67	(0,	-0.58)	643.67	(0,	-0.43)
f15-b2	**516.36**	378.88	(26.62,	-13.95)	**155.70**	(69.84,	54.89)	483.39	(6.38,	-3.49)	371.51	(28.05,	21.59)
IBM_k75	(4035.72)	4202.16	(-4.12,	-7.22)	5782.1	(-43.27,	-44.21)	5927.94	(-46.88,	-47.53)	5655.36	(-40.13,	-40.68)
okgen	**46.43**	46.43	(0.00,	0.00)	46.43	(0.00,	-0.00)	46.43	(0.00,	-0.00)	46.43	(0.00,	-0.00)

TAB. 4 – Résultats avec Minisat

		Pas de Restriction			Horn			Binaire			Horn & Binaire		
instances	T_b	T_r	($\%_p$,	$\%_g$)	T_r	($\%_p$,	$\%_g$)	T_r	($\%_p$,	$\%_g$)	T_r	($\%_p$,	$\%_g$)
gensys	(5181.22)	4672.43	(9.81,	9.79)	7879.9	(-52.08,	-52.10)	5146.42	(0.67,	0.64)	7964.11	(-53.71,	-53.73)
rand_net	(131.01)	173.97	(-32.79,	-34.16)	110.02	(16.01,	15.54)	131.01	(0,	-0.48)	131.01	(0.00,	-0.23)
f3-b25	(1664.94)	2935.61	(-76.31,	-76.41)	1926.54	(-15.71,	-15.77)	2934.61	(-76.25,	-76.35)	1924.24	(-15.57,	-15.63)
ip50	**79.45**	110.37	(-38.91,	-120.79)	143.30	(-80.36,	-108.08)	150.33	(-89.20,	-106.97)	58.31	(26.60,	15.15)
f2clk	(323.15)	270.61	(16.25,	14.28)	291.93	(9.66,	9.01)	453.03	(-40.19,	-40.59)	518.12	(-60.33,	-60.56)
f15-b3	**833.17**	**244.23**	(70.68,	27.53)	976.24	(-17.17,	-31.18)	281.31	(66.23,	59.58)	760.59	(8.71,	4.19)
IBM_k45	(7829.02)	4111.24	(47.48,	46.80)	4382.78	(44.01,	43.88)	3457.46	(55.83,	55.75)	3314.55	(57.66,	57.59)
IBM_k70	(712.10)	225.22	(68.37,	63.22)	757.96	(-6.43,	-7.19)	712.10	(0,	-0.53)	712.10	(0.00,	-0.39)
f15-b2	(794.85)	261.70	(67.07,	28.56)	575.30	(27.62,	15.02)	556.98	(29.92,	23.91)	441.52	(44.45,	40.17)
IBM_k75	**2877.51**	3714.45	(-29.08,	-35.07)	4099.39	(-42.46,	-43.85)	5505.93	(-91.34,	-92.20)	3796.42	(-31.93,	-32.71)
okgen	(323.06)	323.06	(0.00,	-0.00)	323.06	(0.00,	-0.00)	323.06	(0.00,	-0.00)	323.06	(0.00,	-0.00)

Heuristique $w(c) > 0$

		Pas de Restriction			Horn			Binaire			Horn & Binaire		
		T_r	($\%_p$,	$\%_g$)	T_r	($\%_p$,	$\%_g$)	T_r	($\%_p$,	$\%_g$)	T_r	($\%_p$,	$\%_g$)
gensys	(5181.22)	5052.73	(2.47,	2.46)	8046.2	(-55.29,	-55.30)	5163.37	(0.34,	0.33)	8231.05	(-58.86,	-58.87)
rand_net	(131.01)	63.00	(51.91,	50.63)	130.23	(0.59,	0.14)	131.01	(0.00,	-0.47)	131.01	(0.00,	-0.23)
f3-b25	(1664.94)	1701.57	(-2.20,	-2.21)	1698.4	(-2.00,	-2.01)	1745.22	(-4.82,	-4.83)	1723.25	(-3.50,	-3.51)
ip50	**79.45**	113.38	(-42.70,	-110.61)	136.52	(-71.82,	-96.28)	146.68	(-84.61,	-102.54)	60.46	(23.90,	12.30)
f2clk	(323.15)	313.56	(2.96,	1.09)	238.38	(26.23,	25.62)	466.84	(-44.46,	-44.86)	503.75	(-55.88,	-56.13)
f15-b3	**833.17**	675.88	(18.87,	-8.70)	1276.87	(-53.25,	-63.27)	271.53	(67.40,	60.72)	733.52	(11.96,	7.43)
IBM_k45	(7829.02)	5836.14	(25.45,	25.01)	4364.69	(44.24,	44.11)	3341	(57.32,	57.24)	3311.19	(57.70,	57.64)
IBM_k70	(712.10)	406.32	(42.94,	40.79)	740.12	(-3.93,	-4.68)	712.10	(0.00,	-0.53)	712.10	(0.00,	-0.38)
f15-b2	(794.85)	316.32	(60.20,	33.83)	293.82	(63.03,	53.31)	549.23	(30.90,	24.47)	469.28	(40.95,	36.76)
IBM_k75	**2877.51**	**3121.35**	(-8.47,	-12.82)	4017.92	(-39.63,	-40.95)	5170.69	(-79.69,	-80.59)	3738.34	(-29.91,	-30.69)
okgen	(323.06)	323.06	(0.00,	0.00)	323.06	(0.00,	-0.00)	323.06	(0.00,	-0.00)	323.06	(0.00,	-0.00)

TAB. 5 – Résultats avec SatElite

traitement permettant de détecter une partie -mais pas la totalité- des clauses redondantes. Nos expérimentations montrent notamment l'efficacité d'une heuristique de pondération couplée à la PU-redondance. Un tel pré-traitement peut être considéré comme une méthode de compilation permettant des opérations plus rapides sur les instances. Il est intéressant de constater que les temps combinés de simplification et de résolution via notre approche améliorent régulièrement les résultats des solveurs seuls notamment sur des instances réputées difficiles.

Ce travail de recherche ouvre des perspectives intéressantes. Par exemple, un tel pré-traitement peut jouer un rôle utile dans la détection de formules minimalement inconsistantes[12]. En outre, nous nous sommes concentrés sur les clauses binaires ou les clauses de Horn. D'autres fragments polynomiaux tels que les clauses Reverse Horn ou encore les clauses strictement positives ou négatives pourrait être prises en compte dans de futures recherches.

Remerciements

Ce travail à été financé par la Communauté Européenne via les fonds du Feder et par la Région Nord/Pas-de-Calais.

Références

[1] Y. Boufkhad and O. Roussel. Redundancy in random SAT formulas. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI'00)*, pages 273–278, 2000.

[2] S. A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, pages 151–158, New York (USA), 1971. Association for Computing Machinery.

[3] S. Darras, G. Dequen, L. Devendeville, B. Mazure, R. Ostrowski, and L. Sais. Using Boolean constraint propagation for sub-clauses deduction. In *Proceedings of the Eleventh International Conference on Principles and Practice of Constraint Programming (CP'05)*, pages 757–761, 2005.

[4] M. Davis, G. Logemann, and D. W. Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5(7) :394–397, 1962.

[5] A. del Val. Tractable databases : How to make propositional unit resolution complete through compilation. In *Proceedings of the 4th International Conference on Principles of Knowledge Representation and Reasoning (KR'94)*, pages 551–561, 1994.

[6] W. H. Dowling and J. H. Gallier. Linear-time algorithms for testing satisfiability of propositional horn formulae. *Journal of Logic Programming*, pages 267–284, 1984.

[7] O. Dubois, P. André, Y. Boufkhad, and J. Carlier. *Second DIMACS implementation challenge : cliques, coloring and satisfiability*, volume 26 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, chapter SAT vs. UNSAT, pages 415–436. American Mathematical Society, 1996.

[8] O. Dubois and G. 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)*, pages 248–253, 2001.

[9] S. Even, A. Itai, and A. Shamir. On the complexity of timetable and multicommodity flow problems. *SIAM J. Comput.*, 5 :691–703, 1976.

[10] N. Eén and A. 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.

[11] N. Eén and N. 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.

[12] É. Grégoire, B. Mazure, and C. Piette. Extracting MUSes. In *Proceedings of the 17th European Conference on Artificial Intelligence (ECAI'06)*, pages 387–391, Trento, Italy, 2006.

[13] É. Grégoire, R. Ostrowski, B. Mazure, and L. 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.

[14] D. Le Berre. Exploiting the real power of unit propagation lookahead. In *Proceedings of the Workshop on Theory and Applications of Satisfiability Testing (SAT'01)*, Boston University, Massachusetts, USA, June 2001.

[15] 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, 1997.

[16] P. Liberatore. The complexity of checking redundancy of CNF propositional formulae. In *Proceedings of the 15th European Conference on Artificial Intelligence (ECAI'02)*, pages 262–266, 2002.

[17] P. Liberatore. Redundancy in logic i : CNF propositional formulae. *Artificial Intelligence*, 163(2) :203–232, 2005.

[18] P. Marquis. Knowledge compilation using theory prime implicates. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI'95)*, pages 837–843, Montréal, Canada, 1995.

[19] B. Mazure and P. Marquis. Theory reasoning within implicant cover compilations. In *Proceedings of the ECAI-96 Workshop on Advances in Propositional Deduction*, pages 65–69, Budapest, Hungary, August 1996.

[20] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff : Engineering an efficient SAT solver. In *Proceedings of the 38th Design Automation Conference (DAC'01)*, pages 530–535, 2001.

[21] R. Ostrowski, B. Mazure, L. Saïs, and É. Grégoire. 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.

[22] Sathiamoorthy S. and Dhiraj K. P. NiVER : Non-increasing variable elimination resolution for preprocessing SAT instances. In *Proceedings of the Seventh International Conference on Theory and Applications of Satisfiability Testing (SAT'04)*, pages 276–291, 2004.

[23] B. Selman and H. A. Kautz. Knowledge compilation using horn approximations. In *Proceedings of the Ninth National Conference on Artificial Intelligence (AAAI'91)*, pages 904–909, 1991.

[24] B. Selman, H. J. Levesque, and D. G. Mitchell. A new method for solving hard satisfiability problems. In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI'92)*, pages 440–446, 1992.

[25] R. E. Tarjan. Depth first search and linear graph algorithms. *SIAM J. Comput.*, 1 :146–160, 1972.

[26] R. Williams, C. P. Gomes, and B. Selman. Backdoors to typical case complexity. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI'03)*, pages 1173–1178, 2003.

[27] W. Zhang. Configuration landscape analysis and backbone guided local search : Part i : Satisfiability and maximum satisfiability. *Artificial Intelligence*, 158(1) :1–26, 2004.