À propos de l'extension d'un solveur SAT pour traiter des contraintes pseudo-booléennes

Daniel Le Berre, Anne Parrain

À propos de l'extension d'un solveur SAT pour traiter des contraintes pseudo-booléennes

Daniel Le Berre **Anne Parrain**

CRIL-CNRS FRE 2499, Université d'Artois, Lens, France
{leberre,parrain}@cril.univ-artois.fr

Abstract

The use of SAT solvers to handle practical problems has grown dramatically over the last decade. SAT solvers are now mature enough to be used in hardware or software model checkers, planners, semantic web tools, bio-informatic, ... Many researchers are working on extensions of the current framework to more general constraints : pseudo-boolean (PB) constraints, satisfiability modulo theories, etc. We review current available solutions for solving PB-constraints in CDCL solvers and we focus on the limits of our own implementation. We summarize here our own experience regarding our attempt to extend a classical SAT solver with cutting planes instead of resolution. We present first pseudo boolean constraints and their relationship with plain clauses. We review current available solutions for solving PB-constraints in Conflict Driven Clause Learning solvers, the most successful architecture of SAT solvers for SAT instances resulting from a translation of the initial problem into SAT. Then we focus on the current limits of our own implementation SAT4JPseudo that participated to the first and second PB evaluations. We conclude by pointing out some features that require attention for building in the future efficient PB solvers within the CDCL architecture.

Résumé

La résolution pratique du problème SAT rythme maintenant le fonctionnement de divers outils de vérification formelle de logiciels ou matériels, de planificateurs, d'outils pour le web sémantique, pour la gestion de configuration de logiciels, pour la bioinformatique. La principale raison de cet engouement est l'efficacité des prouveurs actuels pour la résolution de problèmes traduits en SAT (prouveurs "à la Chaff"). Une grande partie des travaux de recherche se focalise maintenant sur l'extension des techniques qui ont fait le succès de SAT à des langages plus expressifs que la logique propositionnelle : les formules booléennes quantifiées, les contraintes de cardinalité, les contraintes pseudo-booléennes, et plus récemment les contraintes issues de diverses théories (SMT). Nous présentons dans cet article notre expérience à propos de la réalisation d'un prouveur, SAT4JPseudo, dédié aux contraintes pseudo-booléennes, qui généralise l'approche SAT "à la Chaff" en remplaçant la résolution utilisée lors de l'apprentissage de clauses par les plans de coupe (*cutting planes*) pour l'apprentissage de contraintes pseudo-booléennes. SAT4JPseudo combine des techniques proposées précédemment dans les prouveurs Galena et PBChaff, ce qui permet de les comparer aux approches développées plus récemment. Nous discutons les résultats de SAT4JPseudo lors des deux évaluations internationales de prouveurs pseudo-booléens et identifions ces limites actuelles.

1 Introduction

Dans un grand nombre de domaines différents, il y a un intérêt de plus en plus grand pour le problème SAT. Cet intérêt est essentiellement du à l'efficacité de la résolution pratique des instances SAT issues problèmes réels [15, 16]. Les solveurs dédiés à ce type d'instances SAT sont basés sur une architecture "apprentissage de clauses dirigé par les conflits" (conflict driven clause learning) que nous noterons CDCL dans la suite de l'article. Parmi les raisons de la forte amélioration des solveurs SAT, l'organisation de compétitions comme les compétitions SAT[1] ou la SAT Race[2], et la disponibilité de nombreuses instances (SATLIB[3], Velev[4], IBM[5], jouent certainement un rôle important. Néanmoins, il nous semble que le facteur le plus important est que beaucoup de ces solveurs SAT ont été

[1] http://www.satcompetition.org/
[2] http://fmv.jku.at/sat-race-2006/
[3] http://www.satlib.org/
[4] http://www.ece.cmu.edu/~mvelev
[5] http://www.haifa.ibm.com/projects/verification/RB_Homepage/bmcbenchmarks.html

développés et rendus librement accessibles à la communauté, permettant ainsi de partager les améliorations algorithmiques concernant les contraintes unitaires, la propagation booléenne, l'analyse de conflit, la gestion de la mémoire, etc. . ., et donc de contribuer à une amélioration générale de l'ensemble des solveurs SAT.

Un autre domaine de recherche connexe a été beaucoup développé ces dernières années : la satisfiabilité modulo une théorie[6](satisfiability modulo theories, SMT). On peut le voir comme une généralisation du problème SAT dans lequel les contraintes appartiennent à des théories différentes. Il y a deux ans, une compétition de solveurs SMT a été organisée[7] et c'est le solveur BarcelogicTools [17] qui a gagné la compétition dans les quatre catégories auxquelles il a participé (l'année dernière, il a été vaincu par Yides [8], mais ses résultats étaient encore très bons). Le solveur BarcelogicTools a également participé à la SAT Race 2006 et a obtenu de très bonnes performances (classé 6ème sur 16) pour un solveur de type SAT paramétré. De si bons résultats peuvent suggérer que toutes les techniques utilisées dans les solveurs SAT CDCL pourraient être généralisées et utilisées efficacement dans des solveurs de type plus générique, nommés DPLL(T), où T est la théorie spécifique du moteur d'inférence.

Ce sont les mêmes idées qui ont amené à la proposition de solveurs pseudo-booléens CDCL [1], dans lesquels les contraintes sont des inégalités linéaires avec des coefficients entiers et des variables booléennes. Il n'est pas très difficile d'étendre un solveur SAT pour manipuler de telles contraintes car elles peuvent être affaiblies en clauses propositionnelles pendant la phase d'analyse de conflit. Néanmoins, il est beaucoup plus difficile de garder tous les avantages de ces contraintes en utilisant les plans de coupe (*cutting planes*) [7, 5], car cela remet en cause plusieurs algorithmes et structures de données, utilisés dans les solveurs CDCL.

La première évaluation de solveurs PB a été organisée en 2005 [21]. Il nous semble que le résultat le plus intéressant de cette évaluation a été que la traduction des contraintes pseudo-booléennes en clauses CNF (avec ensuite utilisation d'un solveur SAT classique) a été très compétitive par rapport aux autres techniques. Et la même conclusion a pu être tirée de la deuxième évaluation PB en 2006. Cela peut indiquer que les solveurs PB actuels ne savent pas tirer avantage de la représentation d'un problème en contraintes pseudo-booléennes dans les instances disponibles. Cela ouvre donc des perspectives de recherche intéressantes.

Dans cet article, nous introduisons d'abord la notion de contrainte pseudo-booléenne et la règle d'inférence du *plan de coupe*. Nous décrivons ensuite rapidement le principe d'un algorithme CDCL et nous présentons les méthodes pour l'étendre au cas des contraintes pseudo-booléennes. Nous discutons ensuite les résultats de notre extension d'un solveur SAT, SAT4JPseudo, aux deux évaluations PB, puis nous présentons les leçons que nous pouvons en tirer quant à l'utilisation des plans de coupe à la place de la résolution dans un solveur pseudo-booléen de type CDCL.

[6] `http://www.smt-lib.org/`

[7] `http://www.csl.sri.com/users/demoura/smt-comp/2005/`

2 Contraintes pseudo-boolénnes linéaires

2.1 Définitions

Une contrainte pseudo-booléenne linéaire est définie sur un ensemble de variables entières x_i à valeurs dans $[0, 1]$. 1 représente la valeur `vrai` et 0 la valeur `faux`. La forme générale d'une contrainte pseudo-booléenne linéaire est $\sum_i a_i.x_i \triangleright k$ où a_i et k sont des constantes entières et $\triangleright$ est l'un des opérateurs classiques de comparaison $(=, >, \geq, <, \leq)$. Les opérateurs d'addition et de multiplication ont leur sémantique mathématique habituelle. La partie droite de la contrainte (k) est appelée le *degré* de la contrainte. Un problème pseudo-booléen est soit une conjonction de contraintes pseudo-booléennes (problèmes de décision), soit, le plus souvent, une conjonction de contraintes pseudo-booléennes plus une fonction d'objectif qui consiste à minimiser ou maximiser une expression $\sum_i a_i.x_i$ (problèmes d'optimisation).

Ces contraintes peuvent être transformées afin de n'utiliser que l'opérateur $\geq$: les contraintes d'égalité sont d'abord codées en deux inégalités ($\geq$ et $\leq$) ; les contraintes $<$ et $\leq$ sont multipliées par -1 pour obtenir des contraintes $>$ et $\geq$ respectivement et enfin $\sum_i a_i.x_i > k$ peut être transformée en $\sum_i a_i.x_i \geq k{+}1$. À cette étape, les contraintes pseudo-booléennes sont de la forme :

$$\sum_i a_i.x_i \geq k, \qquad a_i, k \in \mathbb{Z} \tag{1}$$

En introduisant des littéraux l_i et $\overline{l_i}$, on peut encore transformer les contraintes afin de n'obtenir plus que des coefficients entiers positifs : on pose $l_i = x_i$ et $\overline{l_i} = 1 - x_i$. Toute conjonction de contraintes pseudo-booléennes linéaires peut ainsi être transformée en une conjonction de contraintes de type $\geq$ sur des littéraux avec des coefficients strictement positifs :

$$\sum_i a_i.l_i \geq k, \qquad a_i, k \in \mathbb{N} \tag{2}$$

Par exemple, $3x_1 - 7x_2 < -2$ est normalisé en $3\overline{l_1} + 7l_2 \geq 10$.

Les clauses propositionnelles et les contraintes de cardinalités (normalisées) peuvent être considérées

comme un cas particulier de contraintes pseudo-booléennes. Une clause $l_1 \vee l_2 \vee \ldots_n$ se traduit en $l_1 + l_2 + \ldots + l_n \geq 1$, une contrainte de cardinalité $atleast(k, \{l_1, l_2, \ldots, l_n\})$ en $l_1 + l_2 + \ldots + l_n \geq k$ et $atmost(k, \{l_1, l_2, \ldots, l_n\})$ en $\overline{l_1} + \overline{l_2} + \ldots + \overline{l_n} \geq n - k$.

2.2 Règles d'inférence

En calcul propositionnel, il y a essentiellement deux règles d'inférence qui peuvent être appliquées sur une formule en forme normale conjonctive : la *résolution* et la *fusion*. Cette dernière règle est souvent sous-estimée car habituellement les clauses sont vues comme des ensembles de littéraux, et la règle de fusion est obtenue par l'union ensembliste. Néanmoins, la fusion est nécessaire pour la complétude de la procédure de réfutation. La fusion joue également un rôle central dans le calcul des prédicats où des littéraux peuvent être fusionnés par calcul du plus grand unificateur. Dans ce cadre également, la résolution et la fusion sont toutes deux nécessaires pour obtenir une procédure de réfutation complète : mais la fusion n'est plus triviale comme elle l'est en calcul propositionnel.

Avec les contraintes PB, nous avons basiquement besoin des deux mêmes règles pour obtenir une procédure de réfutation complète, mais nous pouvons également utiliser quelques autres règles d'inférence qui n'ont pas leur équivalent dans le calcul propositionnel. La règle correspondant à la résolution est appelée *plan de coupe* et elle calcule une combinaison linéaire positive de deux contraintes pseudo-booléennes.

Pour une contrainte PB en forme (1) :

$$\textit{plan de coupe:} \quad \frac{\sum_i a_i.x_i \geq k \quad \sum_i a'_i.x_i \geq k'}{\sum_i (\alpha.a_i + \alpha'.a'_i).x_i \geq \alpha.k + \alpha'.k' \text{ avec } \alpha > 0 \text{ et } \alpha' > 0}$$

En général, les coefficients pour la combinaison linéaire sont choisis de manière à éliminer au moins une variable, i.e. tels que $\exists i, \alpha.a_i + \alpha'.a'_i = 0$. Contrairement à ce qui se passe avec la résolution en calcul propositionnel, on peut former une combinaison qui n'élimine aucune variable. On peut également forcer une combinaison qui élimine plus d'une variable (ce qui est impossible en calcul propositionnel).

La seconde règle essentielle correspond à la règle de fusion qui est appelée la *saturation*. Si un coefficient a_j est plus grand que le degré k, alors il peut être remplacé par k (ce qui revient à faire une fusion de certaines occurrences de l_j). Cette règle ne peut s'appliquer que sur des contraintes PB de la forme (2) :

$$\textit{saturation:} \quad \frac{a_j.l_j + \sum_{i \neq j} a_i.l_i \geq k \text{ avec } a_j > k}{k.l_j + \sum_{i \neq j} a_i.l_i \geq k}$$

Une manière simple de comprendre cette règle est de considérer que, si l_j est vrai, la contrainte sera satisfaite même si a_j est réduit à k. Ainsi, réduire les coefficients supérieurs au degré ne change pas les solutions pour la contrainte.

En appliquant la règle de saturation sur les contraintes en forme (2), nous obtenons maintenant une forme normale pour les contraintes PB :

$$\sum_i a_i.l_i \geq k, \qquad a_i, k \in \mathbb{N}, \forall i a_i \leq k \qquad (3)$$

La règle d'*arrondi*↑ permet de diviser le degré et chaque coefficient par un entier strictement positif et d'arrondir à l'entier supérieur chaque nombre obtenu :

$$\textit{arrondi}\uparrow: \quad \frac{\sum_i a_i.l_i \geq k}{\sum_i \lceil a_i/\alpha \rceil.l_i \geq \lceil k/\alpha \rceil \text{ avec } \alpha > 0}$$

Avec cette règle, on peut donc inférer une simple clause à partir d'une contrainte pseudo-booléenne en appliquant la règle de l'arrondi↑ avec $\alpha = k$ suivie de la règle de saturation.

La dernière régle d'inférence utilisée dans notre cadre est la *réduction*. Elle consiste à supprimer un des littéraux de la contrainte et à ôter du degré son coefficient :

$$\textit{réduction:} \quad \frac{\sum_i a_i.l_i \geq k}{\sum_{i \neq j} a_i.l_i \geq k - a_j}$$

2.3 Quelques éléments historiques

Les plans de coupe ont été introduits dans le cadre de la programmation linéaire en nombres entiers par R.Gomory en 1958 [12]. La relation entre les plans de coupe et la résolution a été étudiée par J.N. Hooker [13] qui a montré que les plans de coupe sont une généralisation de la résolution. Benhamou, L. Saïs et P. Siegel [3] ont proposé un système de preuve complet pour un type particulier de contraintes de cardinalités. Ils considèrent une contrainte de cardinalité comme un couple (liste de littéraux, degré). Cela signifie qu'un littéral peut apparaître plusieurs fois dans une contrainte : ces contraintes sont donc équivalentes aux contraintes pseudo-booléennes.

P. Barth [2] a proposé la première extension de l'algorithme de Davis-Putnam-Logeman-Loveland (noté DPLL dans la suite) au cas des contraintes pseudo-booléennes. Depuis, plusieurs autres solveurs ont été proposés dans ce même cadre. Nous nous intéresserons particulièrement aux solveurs qui cherchent à conjuguer l'efficacité des solveurs SAT comme Chaff [16]

avec la puissance des règles d'inférence spécifiques aux contraintes PB. La première extension de Chaff aux contraintes PB est apparue dès 2002 avec PBS [1], mais le moteur d'inférence était encore basé sur la résolution. L'utilisation des plans de coupe lors de l'analyse des conflits a été proposée indépendamment dans les solveurs PBChaff [7] et Galena [5].

Pour plus de détails concernant les plans de coupe et le calcul propositionnel, nous invitons le lecteur à se référer, par exemple, à [4].

2.4 Solveurs avec apprentissage de contraintes

Les solveurs SAT complets explorent l'espace des interprétations booléennes en utilisant un arbre binaire sur les valeurs de vérité des variables. Les solveurs DPLL traditionnels parcourent systématiquement cet arbre de recherche. Les solveurs CDCL utilisent une procédure d'apprentissage spécifique pour couper certaines parties de l'arbre. L'idée est la suivante :

1. L'affectation d'une valeur de vérité à une variable est soit impliquée par une contrainte, et dans ce cas la contrainte est appelée la raison de la propagation et la valeur de la variable est dite *forcée*, soit la conséquence d'un choix de l'heuristique, et alors la valeur de la variable est dite *valeur de décision*. Cela est vrai pour tous les solveurs SAT complets. Le cas de base pour la propagation d'une valeur de vérité par une clause est lorsque la clause ne contient qu'un seul littéral non falsifié (on parle de *clause unitaire*) ;
2. la propagation de valeurs de vérité peut s'arrêter lorsqu'il n'y a plus de variable à affecter : la formule est alors prouvée SATisfiable ;
3. le plus souvent, la propagation s'arrête par un conflit : une contrainte a été falsifiée. Dans ce cas, un algorithme classique de retour arrière annule la dernière valeur de décision et essaie avec son opposé. Dans les solveurs CDCL, une procédure spécifique analyse la raison du conflit et produit à la fois une *clause* et un *niveau de retour arrière* tels que cette clause devienne assertive, i.e. qu'elle implique la propagation d'une valeur de vérité après simplification, au niveau de retour arrière proposé. Un solveur CDCL peut remonter à un niveau plus haut que le dernier niveau de décision, et peut ne pas inverser une valeur de décision lors du retour arrière : l'inversion de valeur impliquée par la nouvelle clause peut concerner n'importe laquelle des valeurs de vérité fixée sous le niveau de retour arrière. Ainsi, après chaque conflit, une nouvelle valeur de décision est forcée. L'inconsistence est prouvée lorsque le conflit provient uniquement de valeurs forcées.

Étendre un solveur SAT aux contraintes pseudo-booléennes consiste à remplacer la règle de résolution par les règles d'inférence décrites précédemment. Dans un solveur avec apprentissage de *clauses* dirigé par les conflits, la résolution est utilisée pour produire les nouvelles clauses nécessaires pour le retour arrière non chronologique et pour l'apprentissage durant la procédure d'analyse du conflit. Dans le cadre plus général d'un solveur avec apprentissage de *contraintes* dirigé par les conflits, la procédure d'analyse de conflit utilise toutes les règles précédentes pour produire et vérifier que la nouvelle contrainte apprise vérifie certaines propriétés (voir par exemple [5, 7] pour plus de détails).

D. Chai and A. Kuehlmann présentent dans [5] une généralisation de la structure d'un solveur CDCL [15] qui prend en entrée des clauses, des contraintes de cardinalité et des contraintes pseudo-booléennes et apprend une contrainte de n'importe laquelle de ces trois catégories. L'objectif de la suite de la section 2 est de décrire les différences dans les procédures de propagation et d'analyse de conflit lorsqu'on manipule des clauses propositionnelles ou des contraintes pseudo-booléennes. Nous utilisons les algorithmes proposés dans le solveur CDCL de Chai *et al* pour le cas des contraintes pseudo-booléennes.

2.5 Contraintes déterminantes et fixantes

Dans la suite de l'article, les contraintes pseudo-booléennes sont en forme normale (3).

La définition des clauses unitaires peut être étendue au cas des contraintes pseudo-booléennes. Néanmoins, le terme *unitaire* est employé en référence à la forme syntaxique de ces clauses. Cela n'a plus de sens pour une contrainte pseudo-booléenne, et nous nommerons la notion équivalente par *contrainte déterminante*. Une clause unitaire implique exactement un littéral, ce qui n'est plus vrai dans le cas pseudo-booléen. Une *contrainte déterminante* est une contrainte qui implique *au moins* un littéral. Plus formellement, une contrainte C est déterminante si et seulement si $\exists a_i \in C$ tel que $a_i > \sum a_j - k$. Par exemple, pour x_1, x_2, x_3 littéraux non affectés, la contrainte pseudo-booléenne $3x_1 + 2x_2 + x_3 \geq 5$ implique que x_1 et x_2 soient satisfaits (i.e. soient affectés à 1).

Dans les solveurs CDCL classiques, le but de la procédure d'analyse de conflit est d'apprendre une clause qui deviendra unitaire lors du retour arrière. Ces clauses sont appelées des *clauses fixantes* (communément appelées "assertives clauses" en anglais). Cette définition est simplement étendue au cas pseudo-booléen : une *contrainte fixante* est une contrainte qui deviendra déterminante lors du retour arrière.

2.6 Détecter les contraintes déterminantes

Une contrainte déterminante ne pouvant plus être reperée par sa forme syntaxique, il nous faut maintenant une procédure un peu plus complexe pour détecter si une contrainte est déterminante.

Dixon *et al* [11] nomment *poss* la différence entre la somme des coefficients des littéraux qui ne sont pas falsifiés et le degré de la contrainte. La même valeur est nommée *slack*, c'est-à-dire la marge, par Chai *et al.* Cette valeur permet de détecter facilement certaines situations :

- une contrainte est falsifiée si et seulement si sa valeur *poss* est négative ;
- une contrainte est déterminante si et seulement si il existe un littéral l_i avec un coefficient c_i tel que $poss - c_i$ est négatif.

Il suffit donc de maintenir la valeur de *poss* pendant la recherche pour détecter le moment où une contrainte devient soit falsifiée soit déterminante.

La structure de données paresseuse (*watched literals*) proposée par [16] peut aussi être utilisé pour détecter ces différents états, comme indiqué dans [5]. Mais s'il est suffisant d'observer deux littéraux non falsifiés pour une clause, l'ensemble des littéraux observés pour une contrainte PB doit contenir des littéraux satisfaits ou non affectés tels que la somme de leur coefficient est supérieur à la somme du degré de la contrainte et du plus grand coefficient des littéraux non affectés. Et lorsqu'un littéral de cet ensemble est falsifié, il doit alors être remplacé par le plus petit nombre de littéraux nécessaires pour préserver cette condition. La taille de cet ensemble va donc varier pendant la recherche, ce qui n'est pas le cas pour les clauses. Dans le cas des contraintes de cardinalités, on peut également calculer la taille de l'ensemble des littéraux observés : il est d'exactement $k + 1$ littéraux, puisque le plus grand coefficient d'un littéral non affecté est toujours 1. Mais cela signifie également que la taille de l'ensemble augmente avec la valeur du degré.... Les clauses sont le meilleur cas de figure pour l'approche des littéraux observés.

2.7 Détecter les contraintes fixantes

Les solveurs CDCL détectent les clauses fixantes grâce à un test syntaxique basé sur la définition du point d'implication unique (Unique Implication Point, abrégé en UIP) : les clauses fixantes ne contiennent qu'un seul littéral du niveau de décision courant. On peut obtenir une telle clause soit en effectuant une coupe dans le graphe d'implication, soit en itérant une résolution entre la contrainte conflictuelle et la raison des affectations précédentes, jusqu'à obtenir une clause fixante.

Pour cela, l'algorithme de recherche bénéficie d'une bonne propriété des clauses unitaires : une clause est unitaire au plus une fois dans une branche de l'arbre de résolution. Mais malheureusement cette propriété n'est plus vérifiée dans le cas des contraintes pseudo-booléennes. Par exemple, soient x_1, x_2, x_3 des littéraux non affectés, la contrainte pseudo-booléenne $3x_1 + 2x_2 + x_3 \geq 4$ implique que x_1 est satisfaite. Si x_2 et x_3 sont non affectés, il est aussi facile de voir que, dès que x_2 sera falsifié, la contrainte impliquera que x_3 doit être satisfait, and vice-versa. Ainsi, une même contrainte peut être un conflit et la raison d'une affectation précédente.

De plus, le plan de coupe entre la contrainte falsifiée et la raison de l'affectation conflictuelle ne produit pas toujours une contrainte falsifiée, à l'inverse de ce qui se passe pour les clauses. Le problème vient du fait qu'une affectation forcée par une contrainte déterminante peut sur-satisfaire cette contrainte. La valeur *poss* représente la marge qu'a une contrainte pour ne pas être falsifiée. Si on veut obtenir des contraintes conflictuelles en appliquant les plans de coupe, la somme des valeurs *poss* des deux contraintes doit être strictement négative. Si tel n'est pas le cas, on peut alors appliquer une règle de réduction par suppression de littéraux non affectés ou satisfaits pour diminuer la valeur *poss* d'une contrainte. Cela revient à concentrer l'analyse du conflit sur les littéraux (falsifiés) responsables du conflit.

Enfin, le niveau de retour arrière est le plus haut niveau de décision pour lequel la contrainte est toujours déterminante. Dans le cadre de clauses, il suffit de regarder la clause pour le déterminer. Dans le cadre de PB, il faut tester si la contrainte est déterminante pour chaque niveau entre la racine et le niveau de décision du conflit, et s'arrêter au plus haut niveau.

2.8 Exemple

Nous allons illustrer notre discussion par l'exemple suivant.

$$\begin{cases} (a) & 5x_1 + 3x_2 + 2x_3 + 2x_4 + x_5 = 8 \\ (b) & x_1 + x_3 + x_4 \geq 2 \end{cases}$$

En utilisant les transformations présentées précédemment, la formule est amenée en forme normale :

$$\begin{cases} (a_1) & 5x_1 + 3x_2 + 2x_3 + 2x_4 + x_5 \geq 8 \\ (a_2) & 5\overline{x_1} + 3\overline{x_2} + 2\overline{x_3} + 2\overline{x_4} + \overline{x_5} \geq 5 \\ (b) & x_1 + x_3 + x_4 \geq 2 \end{cases}$$

Au niveau de décision le plus haut (`DL = 0`), x_5 est affecté à 0. (a_1) devient déterminante et x_1 est affecté à 1.

Au niveau de décision suivant (`DL = 1`), x_4 est affecté à 0. (b) devient déterminante et x_3 est affecté à 1. (a_1) devient également déterminante et x_2 est affecté à 1, ce qui amène à un conflit avec (a_2).

Un plan de coupe devrait être effectué entre (a_1) et (a_2) par rapport à x_2. Mais au niveau de décision 1, la valeur *poss* de (a_1) est $+2$, de (a_2) est -2. Leur somme est égale à 0, donc (a_1) doit être réduite. La réduction peut être obtenue en supprimant x_1, puis x_3. La contrainte obtenue est maintenant

$$(a_3) \quad x_2 + x_4 + x_5 \geq 1$$

qui a une valeur *poss* à 0. Le plan de coupe entre (a_3) et (a_2) retourne la contrainte (valeur et niveau de décision sont notés `valeur@niveau`)

$$(c) \quad 2\overline{x_1}(0@0) + 2\overline{x_3}(0@1) + x_4(0@1) + 2x_5(0@0) \geq 2$$

qui est conflictuelle, et fixante au niveau de décision 0 (la valeur 1 est impliquée pour x_3). La contrainte contient deux variables du niveau courant de décision, ce qui n'aurait pas été le cas dans un processus d'apprentissage de clause. De plus, la contrainte apprise est plus forte que la clause

$$x_4 + x_5 \geq 1$$

qui aurait pu être apprise à la place.

3 Résolution des problèmes pseudo-booléens pendant les évaluations PB

Nous rapportons dans les tableaux fig. 1 et fig. 2 une partie des résultats des deux évaluations pseudo-booléennes qui ont été organisées en 2005 [21] et 2006. Nous restreignons notre étude aux solveurs qui ont participé aux deux évaluations, et qui peuvente être considérés comme des extensions de solveurs SAT. Les résultats, dans chaque cellule, indiquent le nombre d'instances résolues soit par des réponses UNSAT/SAT pour les problèmes de décision dans la première ligne, soit par des réponses UNSAT/OPT/SAT pour les problèmes d'optimisation dans les autres lignes. Les nombres donnés entre parenthèses correspondent à des instances pour lesquelles un certificat pour la satisfiabilité n'a pas été donné (en fait, à cause d'une simple erreur dans le format de sortie pour MiniSat+ et d'un problème de temps entre l'interruption du solveur et la sortie du certificat pour SAT4J) : ils représentent la somme des réponses SAT, SAT-TIMEOUT et NO-CERT qui apparaissent sur le site web des évaluations. Les catégories indiquées sont relatives à la taille des entiers présents dans les contraintes : les catégories MEDIUM et BIG nécessitent l'utilisation d'entiers en précision arbitraire pour éviter les problèmes de débordement de capacité. Pour plus de détails, nous invitons le lecteur à se reporter à [21] ou au site web des évaluations[8].

3.1 Les solveurs

Bsolo [14] est un solveur PB de type branch'n bound qui hérite de la plupart des caractéristiques des solveurs CDCL tout en ajoutant des techniques d'estimation de borne et des heuristiques issues de la programmation linéaire sur les entiers. C'est le seul solveur réellement spécialisé pour les problèmes d'optimisation, et effectivement il obtient en général de meilleures performances sur les problèmes d'optimisation que sur les problèmes de décision. Bsolo a aussi une heuristique qui lui permet de s'adapter aux caractéristiques de la catégorie : il annule certains calculs sophistiqués pour les instances en MEDIUM et en BIG. Ainsi, seule la catégorie SMALL bénéficie complètement de la puissance de calcul de Bsolo ;

MiniSat+ [20] utilise une traduction sophistiquée des contraintes pseudo-booléennes vers le format CNF. Les formules CNF sont ensuite données en entrée au solveur MiniSat [9]. La résolution du problème se fait donc simplement à l'aide d'un solveur SAT ;

Pueblo [19] est un solveur CDCL étendu qui utilise à la fois la résolution et les plans de coupe pour l'analyse de conflit. Il ne bénéficie pas de toute la puissance des plans de coupe, car l'analyse s'arrête dès que l'un ou l'autre des moteurs d'inférence lui propose une contrainte fixante. Les contraintes pseudo-booléennes apprises durant la recherche sont périodiquement supprimées ;

PBS [1] était l'une des premières extensions d'un solveur CDCL à prendre en compte des contraintes PB. L'analyse de conflit est basée sur la résolution. La version de PBS soumise à l'évaluation de 2006 utilise également des techniques additionnelles basées sur les plans de coupe (non publiées). PBS n'a pas été capable de répondre SAT pour les problèmes d'optimisation, à cause d'un souci d'interaction entre le solveur et le cadre de l'évaluation ;

SAT4JPseudo Notre propre implémentation d'une extension d'un solveur CDCL dans lequel les plans de coupe remplacent la résolution durant l'analyse de conflit, dans l'esprit de PBChaff [7] et Galena [5]. Le code source de SAT4JPseudo est disponible à partir de `http://www.sat4j.org`.

[8] `http://www.cril.univ-artois.fr/PB06/`

PBChaff n'a pas été soumis à l'évaluation PB car il n'est plus maintenu par ses auteurs. Galena a été soumis à la première évaluation, mais s'est révélé incorrect et n'a pas été corrigé depuis. Sat4jPseudo est donc le seul solveur de l'évaluation avec inférence par plans de coupe.

3.2 Quelques résultats des évaluations PB

Quelques résultats de la première évaluation apparaissent sur la fig. 1. On peut noter que le solveur Minisat+ a été très compétitif par rapport aux solveurs SAT étendus. Pueblo et PBS ont également eu de très bons résultats sur les problèmes de décision. Notre solveur a globalement fourni un grand nombre de réponses, même si beaucoup ne sont pas optimales, i.e. sur un certain nombre d'instances non résolues par les autres solveurs, SAT4JPseudo a retourné un modèle. Pour les réponses optimales, nos résultats sont loin derrière ceux des autres solveurs.

Durant la deuxième évaluation PB (fig. 2), sur un ensemble d'instances plus grand, MiniSat+ a obtenu à nouveau les meilleurs résultats dans la catégorie SMALL, et Pueblo était à nouveau le meilleur solveur PB pour les problèmes de décision. Notre solveur a généralement obtenu de mauvais résultats, mais a quand même été capable d'obtenir un bon nombre de résultats UNSAT dans la catégorie OPTSMALLINT grâce à son moteur d'inférence (voir section suivante). Nous avons ajouté aux résultats officiels de l'évaluation PB une colonne avec les résultats de notre solveur lorsqu'il utilise la résolution à la place des plans de coupe. On se retrouve dans ce cas avec un solveur dont les caractéristiques sont proches de PBS. Ces résultats ont été obtenus dans les mêmes conditions que pour l'évaluation PB. Les résultats obtenus sont généralement meilleurs, en particulier dans la catégorie BIG où il obtient les meilleurs résultats. Nous détaillons dans la section suivante les résultats de ces solveurs sur chaque catégorie spécifique d'instances.

3.3 Le rapport entre rapidité et puissance dans l'évaluation PB 2006

Nous faisons apparaître dans fig. 3 le nombre d'instances résolues (i.e. UNSAT+SAT ou UNSAT+OPT) par chaque solveur pour des familles d'instances bien particulières. Nous nous intéressons aux familles avec une forme particulière (comme les pigeons ou les instances Ardal), ou pour lesquelles la dimension était assez importante pour essayer de tirer des conclusions.

Le problème des pigeons est utilisé pour discriminer les solveurs basés sur la résolution et les solveurs qui utilisent des moteurs d'inférence plus puissants. Le problème des pigeons a une preuve de taille exponentielle en calcul propositionnel, et une preuve de taille polynomiale en utilisant des contraintes de cardinalités ([6] puis [10]). Dans ce cas, SAT4J et PBS distancent les autres solveurs. Pueblo, bien qu'utilisant les plans de coupe durant l'analyse de conflit, ne peut pas tous les résoudre. Mais une approche purement programmation linéaire comme glpPB[9] arrive également à tous les résoudre.

Les problèmes mps-reduced (traduction d'instances originellement au format mps, et réduites pour entrer dans la catégorie des "petits" entiers) contiennent principalement des contraintes pseudo-booléennes. Sur ces instances, notre solveur répond mieux que les autres, et d'une manière générale les solveurs utilisant des plans de coupe ont de meilleurs résultats. Les bons résultats de SAT4J dans cette famille d'instances fournissent 34 résultats UNSAT sur les 54 instances insatisfiables de la catégorie OPTSMALLINT.

D'un autre côté, les problèmes minprime contiennent uniquement des clauses et une fonction d'optimisation. Pour ces instances, c'est la traduction en CNF qui est la meilleure solution, mais Pueblo obtient quand même des résultats assez proches.

À la lumière de ces premiers résultats, on pourrait s'attendre à ce que les solveurs utilisant les plans de coupe réussissent mieux sur les instances contenant principalement des contraintes pseudo-booléennes.

Pourtant certains résultats très surprenants peuvent être trouvés dans les problèmes Ardal fournis par Vasco Manquinho. Ils expriment des instances de problème de sommes de sous-ensembles (*subset sum problem*) avec une seule contrainte d'égalité et peu de variables (entre 60 et 100). Cette contrainte est exprimée par deux inégalités dans le format PB06. MiniSat+ obtient les meilleurs résultats sur ces instances avec 10 réponses. PBS en résoud seulement 3, SAT4J avec plans de coupe, 2, SAT4J avec la résolution, 3 (les instances résolues par PBS et SAT4J ne sont pas les mêmes). La différence d'efficacité entre raisonnement basé sur les plans de coupe et celui basé sur la résolution dans SAT4J peut être illustré sur une instance de cette famille (prob5) : la vitesse du solveur basé sur la résolution est de 1832 décisions par seconde, contre 22 décisions par seconde dans la version plans de coupe, pour un délai de 1800 secondes (avant interruption du solveur) ! L'approche purement programmation linéaire (solveur glpPB) ne résoud aucune de ces instances. Ce genre de résultats explique clairement l'engouement pour les solveurs SAT et la traduction de problèmes en SAT.

Sur les problèmes FPGA, SAT4J a de mauvais résul-

[9] `http://www.eecs.umich.edu/\~\hsheini/pueblo/`

Type		MiniSat+	SAT4JPseudo	Pueblo	PBS	Bsolo
Décision	UNSAT/SAT	43/(35)	52/17	**61/42**	**61**/28	36/8
Optimisation Unsat/Opt/Sat	SMALL	10/**176**/(120)	10/120/**(226)**	10/160/182	10/133/0	10/159/180
	MEDIUM	0/24/(67)	**2**/19/**107**	0/**34**/74	0/33/0	0/28/82
	BIG	**103/26**/(64)	85/3/**(171)**	-	-	90/9/83

FIG. 1 – Quelques résultats de l'évaluation PB05. Voir `http ://www.cril.univ-artois.fr/PB05/results/` pour les résultats exhaustifs.

	PB06					Additional
	MiniSat+	SAT4J Heuristics	Pueblo 1.4	PBS 4.1L	Bsolo	SAT4JRes
UNSAT/SAT	172/148	79/92	**204/153**	199/144	111/118	165/121
SMALL	43/405/250	**54**/357/303	37/385/**323**	29/352/0	40/**409**/280	35/367/(267)
MEDIUM	0/3/9	0/4/9	0/4/**15**	0/5/0	0/**6**/7	0/5/(9)
BIG	38/33/52	37/57/77	-	-	30/14/69	**40/72/96**

FIG. 2 – Quelques résultats de l'évaluation PB06 plus un résultat supplémentaire obtenu dans les mêmes conditions. Voir `http ://www.cril.univ-artois.fr/PB06/results/` pour les résultats exhaustifs.

tats sur les instance satisfiables, mais plutôt bons sur les instances insatisfiables. D'une manière générale, les solveurs utilisant les plans de coupe – glpPB, PBS4.1L, Pueblo, SAT4J – sont les seuls à résoudre toutes les instances insatisfiables de cette catégorie. Les problèmes Uclid sont tous insatisfiables, mais SAT4J n'y est pas très robuste, tandis que MiniSat+ fournit les meilleurs résultats.

Il n'y a que quelques cas où notre solveur a vraiment obtenu de très mauvais résultats comparé aux autres solveurs, notamment sur les problèmes du voyageur de commerce (tsp-problems) et sur les problèmes des reines pondérées[10] (*weighted queens*) fournis by Gayathri Namasivayam. Ces instances ont plus de clauses que de contraintes de cardinalités ou de contraintes PB.

Pour illustrer le noeud du problème, voici le détail du comportement de nos deux solveurs sur une instance des reines : il a été résolu par la version résolution de SAT4J en 35 secondes après 16 redémarrages et 95829 conflits à 3320 décisions par seconde. La version plan de coupe a été interrompue au bout de 1800 secondes après seulement 7 redémarrages et 2338 conflits à 7 décisions par seconde. Donc bien que le raisonnement basé sur les plans de coupe soit puissant (problèmes de pigeons, reduced...) nous n'avons actuellement pas de technique pour l'implémenter efficacement. La section suivante résume les problèmes théoriques ou algorithmiques rencontrés lorsqu'on utilise les plans de coupe comme règle d'inférence.

[10] `http://www.csr.uky.edu/~gayathri/pbsmodels.htm\ #Benchmarks`

3.4 Les limites de SAT4JPseudo basé sur les plans de coupe

Dans la version soumise à l'évaluation PB06, chaque type de contrainte était représenté de la manière la plus simple possible : clause, contrainte de cardinalité, ou contrainte pseudo-booléenne. En conséquence, le coût de la propagation unitaire est maintenant le plus bas possible pour chaque type de contrainte, et la rapidité de ce nouveau solveur a généralement été augmentée. Néanmoins, après avoir analysé les résultats de l'évaluation PB06, nous avons cerné le problème central de notre implémentation : l'analyse de conflit avec les plans de coupe est actuellement très lente. Nous résumons ici les principales raisons :

manipuler des littéraux coefficientés La manipulation de coefficients pendant les opérations de plans de coupe est beaucoup plus coûteuse que la résolution. C'est bien sûr pire encore lorsqu'on effectue les opérations de combinaison linéaire sur des entiers en précision arbitraire. Les autres solveurs habituellement réduisent le coût des opérations en réduisant les contraintes pseudo-booléennes en contraintes de cardinalités, voire en simple clauses, lorsqu'ils risquent le dépassement de capacité.

maintenir une contrainte conflictuelle Lors de l'utilisation des plans de coupe dans la phase d'analyse de conflit, un processus supplémentaire doit être effectué pour garantir que la contrainte dérivée sera conflictuelle [5], ce qui est obligatoire pour rester dans une architecture CDCL. Nous limitons le coût de cette opération grâce aux propriétés démontrées dans [7] qui exhibent des cas particuliers d'application des plans de coupe

	MiniSat+	SAT4J	Pueblo	PBS	Bsolo	glpPB
SAT/UNSAT						
pigeon hole /20	2	**20**	13	**20**	2	**20**
queens /100	**100**	18	99	**100**	**100**	**100**
tsp / 100	91	20	**100**	85	40	42
fpga /57	35	43	**57**	47	9	26
uclid /50	**47**	30	42	44	38	10
Opt Smallint						
minprime /156	**124**	104	118	103	106	52
reduced-mps /273	46	**70**	63	27	54	58
Ardal problems (une contrainte d'égal.)						
Ardal_1 /12	**10**	2	0	3	2	0

Fig. 3 – Quelques résultats de l'évaluation PB06 regroupés par familles d'instances. Voir `http ://www.cril.univ-artois.fr/PB06/results/` pour les résultats exhaustifs.

pour lesquelles la contrainte produite est toujours conflictuelle.

détecter efficacement une contrainte fixante
L'analyse de conflit se termine lorsque la contrainte dérivée est fixante. S'il est facile de détecter une contrainte fixante en utilisant les niveaux de décisions (UIP [15]), pour les contraintes PB, nous n'avons pas actuellement un algorithme incrémental efficace pour résoudre ce problème. Ainsi, notre solveur passe 50% de son temps d'exécution à vérifier si la contrainte dérivée est fixante ou non !

Une approche radicale pour éviter ces problèmes est de reconsidérer l'implémentation d'un solveur pseudo-booléen dans un cadre DPLL(T). L'objectif est de sortir du cadre CDCL qui impose les propriétés des contraintes conflictuelles et fixantes. Ce travail est quasiment réalisé dans BarcelogicTools [18].

4 Conclusion

Nous avons résumé les résultats des solveurs de type CDCL pour les contraintes pseudo-booléennes lors des deux évaluations PB de 2005 et 2006. Nous avons détaillé en particulier le comportement de notre solveur SAT4J sur certaines familles d'instances à partir des résultats de l'évaluation PB06, et nous avons essayé de faire une rapide comparaison entre solveurs basée sur les caractéristiques implémentées dans chacun. Nous espérons que cette compétition va continuer les prochaines années avec plus encore d'instances, afin de mieux comprendre comment traiter ce genre de contraintes.

Notre solveur n'a pas donné de bons résultats en général, mais nous avons identifié le point de blocage de notre implémentation. Il s'agit du module d'analyse de conflit.

Néanmoins, le cadre CDCL nous semble intéressant car il permet par changement du module d'analyse de conflit d'adapter la puissance d'inférence du solveur à de nouvelles contraintes. Cette approche est un compromis entre l'efficacité du solveur SAT et la flexibilité des solveurs DPLL(T). Elle se fait au prix de l'expression des contraintes dans un seul formalisme. Les résultats actuels ne sont pas à la hauteur de nos espérances. Nous dessinons deux perspectives possibles pour améliorer notre solveur :

- concevoir et réaliser un algorithme incrémental efficace pour la détection de contrainte PB fixante, afin de conserver la puissance d'expression des contraintes pseudo-booléennes à toutes les étapes de la recherche ;
- mieux caractériser les contraintes PB à partir desquelles l'application des plans de coupe produit systématiquement une contrainte conflictuelle afin d'alléger le coût de cette opération dans notre prouveur.

Remerciements

Nous voudrions remercier Olivier Roussel pour toutes les discussions que nous avons pu avoir sur le sujet, et aussi pour nous avoir permis de réaliser des tests additionnels sur la plateforme de l'évaluation PB. Nous voudrions également remercier les relecteurs pour leurs commentaires.

Ce travail a été financé en partie par la région Nord-Pas-de-Calais dans le cadre du projet Cocoa.

Références

[1] F. Aloul, A. Ramani, I. Markov, and K. Sakallah. Generic ILP versus Specialized 0-1 ILP : an

update. In *Proceedings of ICCAD'02*, pages 450–457, 2002.

[2] Peter Barth. A Davis-Putnam based enumeration algorithm for linear pseudo-Boolean optimization. Technical Report MPI–I–95–2–003, Saarbrücken, 1995.

[3] B. Benhamou, L. Sais, and P. Siegel. Two proof procedures for a cardinality based language in propositional calculus. In *11th Annual Symposium on Theoretical Aspects of Computer Science (STACS), volume 775 LNCS*, pages 71–82, Caen, France, 1994.

[4] Alexander Bockmayr and Friedrich Eisenbrand. Combining logic and optimization in cutting plane theory. In Hélène Kirchner and Christophe Ringeissen, editors, *FroCos*, volume 1794 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 2000.

[5] Donald Chai and Andreas Kuehlmann. A fast pseudo-boolean constraint solver. In *ACM/IEEE Design Automation Conference (DAC'03)*, pages 830–835, Anaheim, CA, 2003.

[6] W. Cook, C. Coullard, and G. Thuran. On the complexity of cutting planes proofs. *Discrete Applied Mathematics*, 18 :25–38, 1987.

[7] Heidi Dixon. *Automated Pseudo-Boolean Inference within the DPLL framework*. PhD thesis, University of Oregon, 2004.

[8] Bruno Dutertre and Leonardo Mendonça de Moura. A fast linear-arithmetic solver for dpll(t). In Thomas Ball and Robert B. Jones, editors, *CAV*, volume 4144 of *Lecture Notes in Computer Science*, pages 81–94. Springer, 2006.

[9] Niklas Eén Niklas Sörensson. An extensible sat-solver. In *Proceedings of the Sixth International Conference on Theory and Applications of Satisfiability Testing, LNCS 2919*, pages 502–518, 2003.

[10] Heidi E. Dixon Matthew L. Ginsberg. Combining satisfiability techniques from AI and OR. In *The Knowledge Engineering Review 15*, page 53, 2000.

[11] Heidi E. Dixon Matthew L. Ginsberg. Inference methods for a pseudo-boolean satisfiability solver. In *Proceedings of The Eighteenth National Conference on Artificial Intelligence (AAAI-2002)*, pages 635–640, 2002.

[12] R. Gomory. Outline of an algorithm for integer solutions to linar programs. 64 :275–278, 1958.

[13] J. N. Hooker. Generalized resolution and cutting planes. *Ann. Oper. Res.*, 12(1-4) :217–239, 1988.

[14] V. M. Manquinho and J. Marques-Silva. On using cutting planes in pseudo-boolean optimization. *Journal on Satisfiability, Boolean Modeling and Computation (JSAT)*, 2 :209–219, 2006.

[15] Joao P. Marques-Silva and Karem A. Sakallah. GRASP - A New Search Algorithm for Satisfiability. In *Proceedings of IEEE/ACM International Conference on Computer-Aided Design*, pages 220–227, November 1996.

[16] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff : Engineering an Efficient SAT Solver. In *Proceedings of the 38th Design Automation Conference (DAC'01)*, pages 530–535, 2001.

[17] Robert Nieuwenhuis and Albert Oliveras. Decision procedures for sat, sat modulo theories and beyond. the barcelogictools. In Geoff Sutcliffe and Andrei Voronkov, editors, *LPAR*, volume 3835 of *Lecture Notes in Computer Science*, pages 23–46. Springer, 2005.

[18] Robert Nieuwenhuis and Albert Oliveras. On sat modulo theories and optimization problems. In Armin Biere and Carla P. Gomes, editors, *SAT*, volume 4121 of *Lecture Notes in Computer Science*, pages 156–169. Springer, 2006.

[19] Hossein M. Sheini and Karem A. Sakallah. Pueblo : A Hybrid Pseudo-Boolean SAT Solver. *Journal on Satisfiability, Boolean Modeling and Computation (JSAT)*, 2 :165–182, 2006.

[20] Niklas Eén Niklas Sörensson. Translating pseudo-boolean constraints into sat. *Journal on Satisfiability, Boolean Modeling and Computation (JSAT)*, 2 :1–26, 2006.

[21] Manquinho V. and Roussel O. The first evaluation of pseudo-boolean solvers (pb'05). *Journal on Satisfiability, Boolean Modeling and Computation (JSAT)*, 2 :103–143, 2006.