

De la pertinence des littéraux dans les contraintes pseudo-booléennes apprises

Daniel Le Berre, Pierre Marquis, Stefan Mengel, Romain Wallon

▶ **To cite this version:**

Daniel Le Berre, Pierre Marquis, Stefan Mengel, Romain Wallon. De la pertinence des littéraux dans les contraintes pseudo-booléennes apprises. 15es Journées Francophones de Programmation par Contraintes (JFPC'19), Jun 2019, Albi, France. pp.43-52. hal-02912461

HAL Id: hal-02912461
https://hal.science/hal-02912461

Submitted on 3 Dec 2021

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.

De la pertinence des littéraux dans les contraintes pseudo-booléennes apprises

Daniel Le Berre[1,2] **Pierre Marquis**[1,2,3] **Stefan Mengel**[1] **Romain Wallon**[1,2]

[1] CRIL-CNRS UMR 8188, Lens, France
[2] Université d'Artois
[3] Institut Universitaire de France
{leberre,marquis,mengel,wallon}@cril.fr

Résumé

L'apprentissage de contraintes pseudo-booléennes (PB) dans les solveurs PB via le système de preuve des plans-coupes n'est pas aussi bien compris que celui des clauses dans les solveurs SAT utilisant l'analyse de conflit. En particulier, les approches basées sur la résolution généralisée peuvent déduire des contraintes PB comportant des littéraux non pertinents, i.e. des littéraux dont la valeur de vérité (quelle qu'elle soit) ne change jamais la valeur de vérité de la contrainte. Cela peut se produire même quand la formule PB considérée en entrée ne comporte pas de tels littéraux. Nous observons expérimentalement de telles situations sur des instances de l'évaluation pseudo-booléenne 2016. Nous montrons que la détection des littéraux non pertinents dans les contraintes PB apprises, bien que calculatoirement difficile, peut avoir un impact positif sur le temps d'exécution des solveurs (dans notre cas, Sat4j), en permettant le retrait de ces littéraux.

Abstract

Learning pseudo-Boolean (PB) constraints in PB solvers using the cutting planes proof system is a problem that is not as well understood as clause learning in conflict driven clause learning solvers. In particular, it turns out that approaches based on generalized resolution may derive PB constraints containing irrelevant literals, i.e. literals whose assigned truth values (whatever they are) never change the truth value of the constraint. This may happen even when starting from inputs without irrelevant literals. We observe such situations experimentally on benchmarks from the 2016 PB evaluation. We show that the detection and removal of irrelevant literals in learned PB constraints, though computationally expensive, can have a positive impact on the runtime of solvers (in our case, Sat4j).

1 Introduction

En dépit de l'efficacité relative des solveurs SAT modernes dans la résolution de nombreux problèmes industriels, il subsiste des instances qui restent difficiles à résoudre, même pour les solveurs de l'état de l'art. C'est en particulier le cas des formules incohérentes pour lesquelles la preuve d'incohérence nécessite un nombre exponentiel d'étapes de résolution, comme par exemple celles représentant le *principe du pigeonnier* [13]. Parmi ces formules difficiles, nombreuses sont celles requérant la capacité de détecter des symétries et de « compter » pour pouvoir générer des preuves courtes. De ce constat est né l'engouement pour le raisonnement pseudo-booléen [20], qui bénéficie à la fois de l'expressivité des *contraintes pseudo-booléennes* (des équations et inéquations linéaires en variables booléennes) et du pouvoir d'inférence du système de preuve des *plans-coupes* [3, 12, 14, 19]. Ce système de preuve est en théorie strictement plus puissant que le système de preuve à base de résolution utilisé dans les solveurs SAT. Cependant, en pratique, aucun des solveurs PB existants n'utilise pleinement le pouvoir du système de preuve des plans-coupes [22]. En effet, la plupart des solveurs actuels sont fondés sur une forme spécifique de ce système, pouvant être vue comme une généralisation de la résolution [14]. Celle-ci permet d'étendre l'inférence clausale à l'inférence pseudo-booléenne, et ainsi d'hériter de nombreuses techniques utilisées dans la résolution du problème SAT [2, 6, 8]. En particulier, l'apprentissage de clauses guidé par les conflits (*conflict-driven clause learning*) à la base de l'architecture des solveurs SAT modernes [9, 17, 18] peut être généralisé aux solveurs PB, permettant ainsi d'*apprendre* des contraintes pour effectuer des retours-arrière non chronologiques.

Lorsqu'un conflit survient durant la recherche (i.e. lorsqu'une contrainte pseudo-booléenne est falsifiée), la règle de résolution généralisée [14] est appliquée entre la contrainte conflictuelle et la raison de la propagation de l'un de ses littéraux pour produire une nouvelle contrainte. Il est important de noter que, contrairement aux solveurs utilisant la résolution, la raison doit parfois être *affaiblie* au préalable pour que la contrainte inférée soit toujours conflictuelle. Cette opération est répétée jusqu'à ce que la contrainte déduite permette de propager certains de ses littéraux. La contrainte est ensuite apprise, avant d'effectuer un retour-arrière. Au fil des ans, de nombreux solveurs pseudo-booléens implantant des variantes du système des plans-coupes ont été développés [2, 7, 15, 21]. Cependant, depuis la première évaluation pseudo-booléenne [16], le constat est décevant : ces solveurs ne parviennent pas à résoudre une gamme de problèmes aussi large que les solveurs utilisant la résolution, qui encodent les contraintes sous forme de clauses. En particulier, les bonnes performances des solveurs PB sur certaines catégories spécifiques d'instances ne se généralisent pas à toutes les catégories [10]. Ceci est en partie dû au fait que les règles du système des plans-coupes sont difficiles à implanter efficacement, et que choisir lesquelles desdites règles il faut appliquer est loin d'être trivial. En premier lieu, l'approche considérée a été de remplacer l'application des règles de la résolution par celles de la résolution généralisée pendant l'analyse de conflit. Cependant, une telle approche n'est pas satisfaisante, car elle est équivalente à la résolution lorsqu'elle est appliquée à des clauses et nécessite un traitement spécifique pour déduire des contraintes de cardinalité [1]. Plus récemment, une amélioration a été proposée et implantée dans le solveur *RoundingSat* [11]. Elle consiste en une utilisation agressive de la règle de division fournie par le système des plans-coupes (mais pas par la résolution généralisée).

Dans cet article, nous considérons un nouveau point de vue sur les propriétés des systèmes de preuve introduits plus haut, et, en particulier, sur l'une de leurs faiblesses. En effet, il semble naturel de penser que les contraintes apprises au cours de l'analyse de conflit contiennent uniquement des *littéraux pertinents*, i.e. des littéraux dont l'affectation a une influence sur la valeur de vérité de la contrainte. Tant que les contraintes inférées sont des clauses ou des contraintes de cardinalité, la question ne se pose pas : les littéraux jouant un rôle symétrique dans ces contraintes, chacun d'eux est pertinent sauf si la contrainte est valide. Cela peut cependant ne pas être le cas pour des contraintes pseudo-booléennes. Alors que, le plus souvent, les instances pseudo-booléennes ne contiennent que des littéraux pertinents, la résolution généralisée peut produire des contraintes comportant des littéraux qui ne le sont pas. Nous observons que leur présence peut nuire à l'efficacité des solveurs PB. D'un côté, de tels littéraux rendent la contrainte plus complexe que nécessaire, à la fois en termes de nombre de littéraux et de taille de coefficients. De l'autre, ils rendent caduque le critère syntaxique permettant la détection de clauses et de contraintes de cardinalité, pour lesquelles les solveurs disposent de structures de données plus efficaces. Pire, la présence de littéraux non-pertinents peut conduire à l'inférence de contraintes plus faibles que celles qui auraient pu être déduites. Ce constat s'applique tout particulièrement lorsque des arrondis sont appliqués aux contraintes. Nous montrons que ces opérations peuvent non seulement donner plus d'importance aux littéraux non pertinents qu'ils n'en ont initialement, mais peuvent même les rendre pertinents, réduisant ainsi le pouvoir de propagation de la contrainte apprise. Ces observations conduisent naturellement à l'idée de détecter les littéraux non pertinents, afin de pouvoir les supprimer. Malheureusement, vérifier si un littéral est pertinent dans une contrainte pseudo-booléenne est NP-complet [5, Section 9.6]. D'un point de vue pratique, il semble donc irréaliste d'effectuer cette opération sur toutes les contraintes produites. Cependant, comme nous le montrons plus loin, les littéraux non pertinents présentent des propriétés intéressantes qui peuvent être exploitées pour réduire le nombre de tests à réaliser sur une contrainte pour les identifier. Grâce à cela, nous avons pu implanter diverses approches permettant de détecter une large majorité des littéraux non pertinents dans les contraintes apprises. Nos expérimentations montrent que ces approches, combinées à des bornes adéquates sur les contraintes à traiter, permettent de détecter et de supprimer ces littéraux des contraintes apprises, et peuvent même améliorer significativement le temps d'exécution des solveurs.

La suite de cet article est organisée comme suit. Dans un premier temps, nous introduisons quelques préliminaires relatifs au raisonnement pseudo-booléen. Puis, nous discutons de l'existence de littéraux non pertinents dans les contraintes pseudo-booléennes, et expliquons dans quelle mesure ils peuvent affecter les performances des solveurs. Nous présentons ensuite notre algorithme pour identifier et supprimer ces littéraux, avant de l'évaluer expérimentalement. Enfin, nous concluons en présentant quelques perspectives de futurs travaux.

2 Préliminaires

Nous considérons un cadre propositionnel défini sur un ensemble fini de variables propositionnelles V interprété classiquement. Un *littéral* l est une variable

propositionnelle $x \in V$ ou sa négation $\bar{x}$. Notons que, dans ce contexte, les valeurs booléennes sont représentées par les entiers 1 (vrai) et 0 (faux) et que $\bar{x} = 1 - x$. L'implication logique est symbolisée par $\models$ et l'équivalence logique par $\equiv$.

2.1 Définitions

Contraintes pseudo-booléennes. Une *contrainte pseudo-booléenne (PB)* est une contrainte de la forme $\sum_{i=1}^{n} a_i l_i \ \triangle \ d$, où les a_i et d sont des entiers, les l_i des littéraux et $\triangle \in \{\leq, <, =, >, \geq\}$. Chaque a_i est appelé *poids* ou *coefficient* et d est appelé *degré* de la contrainte. Toute contrainte PB peut être *normalisée* en une (conjonction de) contraintes de la forme $\sum_{i=1}^{n} a_i l_i \geq d$ dans laquelle les coefficients et le degré sont positifs. C'est pourquoi, dans la suite, nous supposons que toutes les contraintes PB sont normalisées. Une *contrainte de cardinalité* est une contrainte PB dont tous les coefficients valent 1, et une *clause* est une contrainte de cardinalité de degré 1. Cette définition coïncide avec la définition usuelle d'une clause comme une disjonction de littéraux, et illustre le fait que les contraintes de cardinalité et pseudo-booléennes généralisent les clauses.

Conditionnement. Étant données une contrainte PB χ et une conjonction de littéraux τ, $\chi|\tau$ représente le *conditionnement* de χ par τ, qui est équivalent à la contrainte χ où chaque littéral est remplacé par 1 s'il apparaît dans τ, ou par 0 si son opposé apparaît dans τ. Les constantes apparaissant dans le membre gauche sont ensuite déplacées dans le membre droit, pour obtenir une contrainte PB correspondant à la définition précédente. Par ailleurs, notons que, pour deux termes τ_1 et τ_2 conjointement cohérents, $(\chi|\tau_1)|\tau_2$ est équivalent à $\chi|(\tau_1 \wedge \tau_2)$.

2.2 Règles d'inférence et systèmes de preuve

Le système de preuve des *plans-coupes* [12] constitue la contre-partie du système de preuve de la *résolution* dans le cadre pseudo-booléen. Nous présentons ici quelques-unes de ses règles.

Saturation. La contrainte $al + \sum_{i=1}^{n} a_i l_i \geq d$, où $a > d$, est équivalente à $dl + \sum_{i=1}^{n} a_i l_i \geq d$. Autrement dit, toute contrainte PB peut être réécrite de manière à ce que tous ses coefficients soient au plus égaux à son degré.

Clashing addition. La conjonction des deux contraintes PB $\chi_1 : al + \sum_{i=1}^{n} a_i l_i \geq d_1$ et $\chi_2 : b\bar{l} + \sum_{i=1}^{n} b_i l_i \geq d_2$, permet de déduire la contrainte $\sum_{i=1}^{n} (ba_i + ab_i) l_i \geq (bd_1 + ad_2 - ab)$. Notons que, lorsque χ_1 et χ_2 sont des clauses, cette opération est équivalente à la résolution classique [14].

Division. Étant donné un entier positif α, la contrainte $\sum_{i=1}^{n} a_i l_i \geq d$ implique $\sum_{i=1}^{n} \lceil \frac{a_i}{\alpha} \rceil l_i \geq \lceil \frac{d}{\alpha} \rceil$. De plus, si tous les a_i sont divisibles par α, alors la première contrainte est équivalente à la seconde.

Dans la suite, nous appelons *résolution généralisée* le système de preuve utilisant les règles de saturation et de *clashing addition* et par *plans-coupes* le système de preuve autorisant toute combinaison linéaire de contraintes et la règle de division.

3 Littéraux non pertinents

Un problème spécifique aux contraintes PB, que l'on ne retrouve pas dans les clauses ou contraintes de cardinalité non valides, est l'existence de littéraux *non pertinents*.

Definition 1 (Littéral non pertinent)**.** *Un littéral l est dit* non pertinent *dans une contrainte χ lorsque $\chi|l \equiv \chi|\bar{l}$. Sinon, l est dit* pertinent *dans χ. Dans ce dernier cas, χ est aussi dite* dépendante *de l. De manière équivalente, l est* non pertinent *lorsque, pour tout modèle M de χ, inverser la valeur de l dans M ne peut en faire un contre-modèle de χ.*

Dans la suite, lorsqu'il n'y a pas d'ambiguïté sur la contrainte χ, nous omettons cette contrainte et disons simplement que l est pertinent ou non.

3.1 Impact des littéraux non pertinents

Considérons la contrainte PB χ définie comme suit :

$$\chi : 17a + 10b + 10c + \mathbf{d} + \mathbf{e} \geq 17$$

Nous observons que d et e ne sont ici pas pertinents : pour satisfaire χ, il suffit de satisfaire a ou de satisfaire b et c. Donc, d et e peuvent être retirés de la contrainte. Pour ce faire, plusieurs solutions existent, la plus simple étant de se contenter de les ôter du membre gauche. Une approche plus intéressante consiste à satisfaire ces littéraux de manière à réduire le degré de la contrainte. En appliquant cela à notre exemple, nous obtenons la contrainte suivante :

$$\chi \equiv 17a + 10b + 10c \geq \mathbf{15}$$

Remarquons que la règle de saturation peut être appliquée sur cette contrainte, nous donnant ainsi :

$$\chi \equiv \mathbf{15a} + 10b + 10c \geq 15$$

Notons que tous les coefficients de cette contrainte sont divisibles par 5. En divisant par ce nombre, nous obtenons la contrainte équivalente suivante :

$$\chi \equiv 3a + 2b + 2c \geq 3$$

Cet exemple montre qu'en identifiant et en retirant les littéraux non pertinents d'une contrainte, il est possible de diminuer le nombre de ses littéraux, mais aussi la valeur de ses coefficients et de son degré. Ces simplifications permettent d'améliorer les performances du solveur, en détectant parfois des clauses ou des contraintes de cardinalité *cachées*, pour lesquelles les solveurs disposent de structures de données optimisées. En effet, il est possible qu'après avoir retiré tous les littéraux non pertinents, les coefficients de la contrainte obtenue soient tous égaux. Par exemple, considérons la contrainte $3a+3b+3c+3d+\mathbf{e}+\mathbf{f} \geq 6$. En retirant les littéraux non pertinents e et f, la contrainte $3a + 3b + 3c + 3d \geq 4$ est produite. Il s'agit d'une contrainte de cardinalité, puisqu'en la divisant par 3, nous obtenons $a + b + c + d \geq 2$.

3.2 Littéraux inutiles dans les contraintes apprises

Comme nous venons de le montrer, les solveurs PB – contrairement aux solveurs SAT – peuvent être conduits à gérer des contraintes comportant des littéraux non pertinents. Ces contraintes peuvent soit provenir de la formule originale (même si cela ne se produit pas en pratique), soit être apprises lors de l'analyse de conflit. Rappelons que, lorsqu'un solveur rencontre un conflit, il applique successivement des règles de son système d'inférence pour déduire de nouvelles contraintes. Il se trouve qu'à partir de contraintes ne comportant que des littéraux pertinents, la résolution généralisée peut produire des contraintes dont certains littéraux ne le sont pas.

Par exemple, considérons les contraintes $4a + 2b + 2c+2y+x \geq 5$ et $3\bar{x}+d+e \geq 4$. En appliquant la règle de *clashing addition* sur ces deux contraintes, avec x comme pivot, la contrainte suivante est produite :

$$12a + 6b + 6c + 6y + \mathbf{d} + \mathbf{e} \geq 16$$

Dans cette contrainte, d et e ne sont pas pertinents. Si nous résolvons maintenant cette nouvelle contrainte avec $6\bar{y} + 5a + 4b + 4c \geq 7$ en prenant y comme pivot, nous obtenons la contrainte de notre exemple précédent :

$$17a + 10b + 10c + \mathbf{d} + \mathbf{e} \geq 17$$

Nos expérimentations (cf. section 5) montrent que ce scénario se produit en pratique assez fréquemment, et en particulier que de nombreux littéraux non pertinents peuvent apparaître dans les contraintes apprises par le solveur *Sat4j*, qui utilise la résolution généralisée.

3.3 Littéraux non pertinents et division

Récemment, *RoundingSat* [11] a proposé une amélioration dans l'implantation de solveurs PB visant à étendre la gamme des instances résolues efficacement par ces solveurs. Empiriquement, nous avons constaté que les contraintes produites par ce solveur peuvent également contenir des littéraux non pertinents, mais dans une mesure bien moindre que *Sat4j*. C'est un pas en avant dans la conception d'un système de preuve garantissant que les littéraux des contraintes inférées sont tous pertinents si c'est le cas des contraintes originales. Cependant, éviter la présence de ces littéraux n'est pas suffisant, puisqu'il est possible d'affaiblir une contrainte au point de rendre pertinents des littéraux qui ne le sont pas initialement. Malheureusement, cela peut se produire dans *RoundingSat*, comme nous le montrons ci-après. Retirer les littéraux non pertinents des contraintes produites par *RoundingSat* présente donc également un intérêt.

Impact de la division. Avant d'appliquer la règle de *clashing addition* au cours de l'analyse de conflit, *RoundingSat* applique sur les deux contraintes la division par le coefficient c du littéral servant de pivot, afin de s'assurer qu'il devienne égal à 1. Les littéraux non falsifiés dont le coefficient n'est pas divisible par c sont affaiblis, mais les coefficients des littéraux falsifiés sont divisés et arrondis par excès. Cette opération peut alors rendre *artificiellement* pertinents des littéraux qui ne le sont pas.

Considérons par exemple $6x + 3y + 3z + \mathbf{t} \geq 9$ où t n'est pas pertinent. Supposons que t soit falsifié, et que *RoundingSat* utilise y comme pivot pour la *clashing addition*. En divisant la contrainte par 3, nous obtenons $2x+y+z+t \geq 3$, où t est pertinent : il joue maintenant un rôle symétrique à celui de y et z.

Notons que, si t avait été retiré avant d'appliquer la division, nous aurions obtenu la contrainte plus forte $2x + y + z \geq 3$.

Impact de la réduction en contrainte de cardinalité. Lors de l'analyse de conflit, si *RoundingSat* détecte que les coefficients deviennent « trop grands », la contrainte apprise est réduite en une contrainte de cardinalité. Par exemple, $6x + 3y + 3z + \mathbf{t} \geq 9$ devient $x + y + z + t \geq 2$. Cette fois encore, une contrainte plus forte aurait pu être apprise en retirant le littéral non pertinent t, en l'occurrence $x + y + z \geq 2$.

4 Identification et suppression

Comme nous l'avons vu plus haut, supprimer les littéraux non pertinents des contraintes PB peut permettre de simplifier ces dernières. Malheureusement, il est connu que vérifier si un littéral est pertinent dans une contrainte PB donnée est NP-complet [5, Section 9.6]. En conséquence, vérifier la pertinence de chaque littéral de chacune des contraintes produites semble hors de portée d'un point de vue pratique. Cependant, comme nous le montrons ici, certaines propriétés des littéraux non pertinents peuvent être exploitées pour définir un algorithme détectant « efficacement » les littéraux non pertinents afin de les supprimer. Afin d'illustrer ces propriétés, nous considérons la pertinence du littéral l dans la contrainte suivante, utilisée comme exemple récurrent :

$$\chi : al + \sum_{i=1}^{n} a_i l_i \geq d$$

4.1 Identifier un littéral non pertinent

Rappelons que l n'est pas pertinent si, et seulement si, $\chi|l \equiv \chi|\bar{l}$, ou, autrement dit :

$$\sum_{i=1}^{n} a_i l_i \geq d - a \equiv \sum_{i=1}^{n} a_i l_i \geq d$$

Notons que, comme $d > d - a$, la seconde contrainte implique trivialement la première, donc la seule chose à vérifier est l'implication suivante :

$$\sum_{i=1}^{n} a_i l_i \geq d - a \models \sum_{i=1}^{n} a_i l_i \geq d$$

En conséquence, une première approche pour déterminer si l est pertinent ou pas consiste à vérifier la cohérence de la conjonction des contraintes :

$$\sum_{i=1}^{n} a_i l_i \geq d - a \wedge \sum_{i=1}^{n} a_i l_i < d$$

Ceci revient, après normalisation, à vérifier la cohérence des deux contraintes PB :

$$\sum_{i=1}^{n} a_i l_i \geq d - a \wedge \sum_{i=1}^{n} a_i \bar{l_i} \geq \sum_{i=1}^{n} a_i - d$$

Cette formule est cohérente si, et seulement si, le littéral l est pertinent. Ainsi, il suffit d'utiliser son solveur PB préféré pour tester la pertinence d'un littéral.

Notons que le fait qu'il n'y ait que deux contraintes ne signifie pas que ce problème est facile à résoudre (le problème NP-complet *subset-sum* se réduit à celui de la cohérence de deux contraintes PB). Ceci est d'autant plus vrai que le nombre de littéraux dans les contraintes est grand.

Une autre approche pour tester la pertinence de l consiste à utiliser la *programmation dynamique*. En effet, rappelons l'implication que nous souhaitons vérifier :

$$\sum_{i=1}^{n} a_i l_i \geq d - a \models \sum_{i=1}^{n} a_i l_i \geq d$$

Observons que cette implication est vraie si, et seulement si, il n'existe aucune interprétation de $\sum_{i=1}^{n} a_i l_i$ égale à un nombre compris entre $d - a$ et $d - 1$. Pour vérifier la non-pertinence de l, il suffit de vérifier qu'il n'existe aucun sous-ensemble de $a_1, ..., a_n$ dont la somme vaut l'un de ces nombres, et donc, de résoudre *subset-sum* pour chacune de ces entrées.

Il est bien connu que cela peut se faire en temps $O(nd)$ avec un algorithme de programmation dynamique [4, Chapter 34.5], et donc en temps pseudo-polynomial dans la taille de la contrainte et de son degré. Dans cette approche, un tableau S de valeurs booléennes est construit, et S_j est vrai si, et seulement si, il existe un sous-ensemble des éléments dont la somme vaut j. Dans notre cas, pour savoir si l n'est pas pertinent, il suffit de vérifier qu'une fois le tableau rempli, S_j est faux pour tout j entre $d - a$ et $d - 1$.

4.2 Réduire le nombre de tests

Vérifier la pertinence d'un littéral étant calculatoirement difficile, il n'est pas possible en pratique de faire le test pour *tous* les littéraux d'une contrainte. Heureusement, cela n'est pas nécessaire. Observons dans un premier temps que, si l n'est pas pertinent, alors tous les littéraux ayant le même coefficient ne sont pas pertinents non plus, puisqu'ils sont tous symétriques dans la contrainte. Par ailleurs, la propriété suivante permet d'optimiser le nombre de tests à effectuer :

Proposition 1. *S'il existe i_0 tel que le littéral l_{i_0}, de coefficient $a_{i_0} < a$, est pertinent, alors l est aussi pertinent.*

Démonstration. Par l'absurde, supposons que l ne soit pas pertinent dans ce cas. l_{i_0} étant supposé pertinent, il existe un modèle M de χ tel que M satisfait l_{i_0}, et falsifier ce littéral fait de M un contre-modèle de χ. Notons M' le contre-modèle ainsi obtenu. Quelle que soit la valeur donnée au littéral non pertinent l, M est un modèle de χ et M' en est un contre-modèle, donc supposons que l soit falsifié.

L'interprétation M satisfait la contrainte suivante :

$$\chi|(\bar{l} \wedge l_{i_0}) \equiv \sum_{i=1, i \neq i_0}^{n} a_i l_i \geq d - a_{i_0} \qquad (1)$$

Observons que c'est aussi le cas de M' puisque cette contrainte contient seulement des littéraux sur lesquels les deux interprétations coïncident.

Rappelons maintenant que l est supposé non pertinent, donc changer sa valeur ne permet pas de faire de M' un modèle de χ. En particulier, la contrainte suivante ne peut donc pas être satisfaite par M'.

$$\chi|(l \wedge \bar{l_{i_0}}) \equiv \sum_{i=1, i\neq i_0}^{n} a_i l_i \geq d - a \qquad (2)$$

Cependant, comme $a_{i_0} < a$, alors $d - a_{i_0} > d - a$, donc $(1) \models (2)$, et M' est un modèle de (2). Comme cela n'est pas possible, l est nécessairement pertinent. □

Pour résumer nos observations, nous avons que :
— si plusieurs littéraux ont le même coefficient et que l'un d'entre eux n'est pas pertinent, alors aucun de ces littéraux ne l'est ;
— si une contrainte dépend d'un littéral, alors elle dépend de tous les littéraux ayant un coefficient plus grand que celui de ce littéral.

Grâce à ces résultats, nous pouvons réduire le nombre de tests de pertinence en ordonnant les littéraux par coefficients croissants. Un seul test par coefficient est nécessaire, et dès qu'un littéral pertinent est identifié, tous les autres littéraux sont pertinents. Cette approche est résumée par l'algorithme 1, dans lequel la fonction *dépendDe* peut être implantée soit avec un solveur PB, soit en utilisant la programmation dynamique

Algorithme 1 : ôter-littéraux-non-pertinents

Entrée : Une contrainte PB χ non valide
Sortie : χ sans ses littéraux non pertinents

Trier les coefficients de χ par ordre croissant
pour chaque *c coefficient dans* χ **faire**
  Choisir un littéral l ayant c pour coefficient
  si *dépendDe*(l, χ) **alors**
    retourner χ
  fin
  Retirer tous les littéraux de χ de coefficient c
  Adapter le degré de χ
  Saturer χ
fin

Proposition 2. *L'algorithme 1 est correct et se termine.*

Démonstration. D'une part, l'algorithme est trivialement correct de par nos résultats précédents. D'autre part, la contrainte χ contient un nombre fini de coefficients, et puisqu'elle n'est pas valide, elle dépend nécessairement d'au moins l'un de ses littéraux. □

Considérons par exemple la contrainte suivante (l'un de nos précédents exemples) comme entrée de notre algorithme :

$$17a + 10b + 10c + d + e \geq 17$$

Le premier coefficient est 1, et soit d soit e est choisi. Le test de pertinence révèle que le littéral choisi ne l'est pas, et donc d et e sont retirés tous les deux. Le degré est mis à jour, et la contrainte saturée :

$$15a + 10b + 10c \geq 15$$

Le coefficient considéré ensuite est 10, et soit b soit c est choisi. Le littéral est pertinent, donc la contrainte ne peut plus être modifiée. Elle est donc apprise sous cette forme (la détection du diviseur commun est laissée au solveur).

5 Résultats expérimentaux

Cette section présente des résultats expérimentaux illustrant la présence de littéraux non pertinents dans les contraintes apprises, ainsi que leur impact sur les performances des solveurs PB. Toutes les expérimentations présentées dans cette section ont été réalisées sur un cluster de calcul équipé de bi-processeurs quadri-cœur Intel XEON E5-5637 v4 (3.5 GHz) et de 128 Go de mémoire. Les instances considérées sont les 777 instances de décision soumises à la dernière évaluation pseudo-booléenne (`http://www.cril.univ-artois.fr/PB16/`). Nous avons implanté trois approches pour détecter les littéraux non pertinents, utilisant soit la résolution généralisée avec le solveur *Sat4j-CP*, soit la résolution avec le solveur *Sat4j-Res*, soit un algorithme de programmation dynamique résolvant *subset-sum*. Les diverses implantations sont disponibles à l'adresse `https://gitlab.ow2.org/sat4j/sat4j/tree/sat19`. Pour rester efficace, seules les contraintes comportant moins de 1000 littéraux et ayant un degré inférieur à 20000 sont traitées. Les autres contraintes ne sont pas modifiées. Par ailleurs, un délai de garde de 5 secondes est fixé aux deux approches utilisant un solveur : lorsqu'il est atteint, le littéral est supposé pertinent, rendant ces approches incomplètes mais correctes. L'approche par programmation dynamique ne nécessite pas de tel délai, les bornes choisies fournissant suffisamment de garanties quant à sa complexité temporelle.

5.1 Expérimentations préliminaires

Une étude préalable des instances considérées a révélé que les littéraux de leurs contraintes sont tous pertinents. Cela semble naturel, puisqu'elles modélisent des

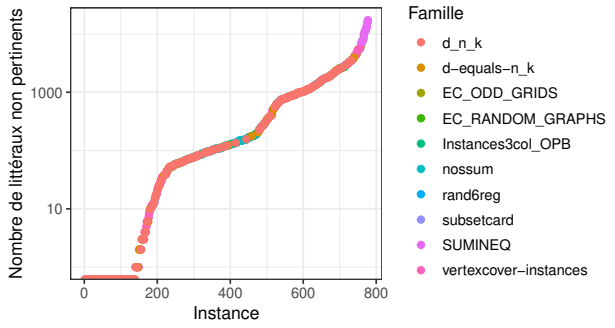


FIGURE 1 – Nombre de littéraux non pertinents par instance. Chaque point est une instance, et son ordonnée est le nombre de littéraux non pertinents détectés dans les contraintes apprises par *Sat4j-CP* sur cette instance (en échelle logarithmique).

problèmes réels et que la phase de modélisation n'introduit normalement pas de littéraux non pertinents.

Afin d'évaluer dans quelle mesure la résolution généralisée produit des littéraux non pertinents sur ces instances, nous avons exécuté *Sat4j-CP* [15] sur ces instances et récupéré pour chacune d'elles les 5000 premières contraintes apprises. La figure 1 montre le nombre de littéraux non pertinents qui y sont détectés.

Pour ces expérimentations, nous avons utilisé les trois approches proposées. Même si celles-ci détectent un nombre équivalent de littéraux non pertinents, la figure 2 montre que la plus efficace est celle utilisant la programmation dynamique.

Nous avons réalisé les mêmes expérimentations avec *RoundingSat* [11]. Celles-ci ont révélé que ses contraintes apprises ne comportent que très peu de littéraux non pertinents : aucun pour une grande majorité d'instances, et jusqu'à 12 pour 15 instances parmi les 24 de la famille `SUMINEQ`. Comme nous l'avons vu plus haut, cela est dû au fait que la division appliquée par *RoundingSat* rend pertinents des littéraux qui ne le sont pas dans ses contraintes apprises.

5.2 Impact du retrait

Comme nous l'avons montré dans une précédente section, le retrait des littéraux non pertinents peut, d'un point de vue théorique, simplifier les contraintes et ainsi améliorer les performances du solveur. Pour évaluer son impact pratique, nous l'avons implanté dans *Sat4j-CP*. Plus précisément, à l'issue de chaque analyse de conflit, la contrainte apprise est traitée de manière à en retirer les littéraux non pertinents. Pour des raisons d'efficacité, seule cette contrainte est considérée ; les contraintes intermédiaires ne sont pas



FIGURE 2 – Comparaison des trois approches en termes de temps d'exécution (en échelle logarithmique). L'abscisse mesure le temps de l'approche par programmation dynamique, et l'ordonnée celui des approches utilisant *Sat4j* : *Sat4j-CP* (en haut) et *Sat4j-Res* (en bas). Les couleurs utilisées sont les mêmes qu'à la figure 1.

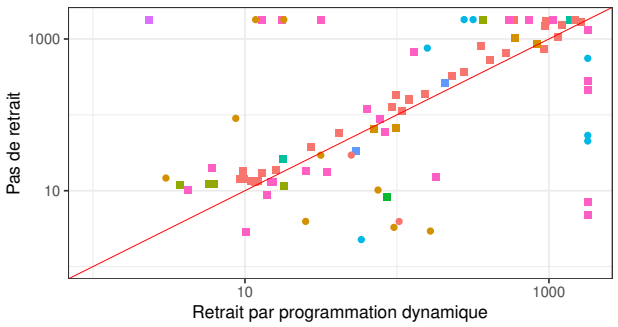


FIGURE 3 – Comparaison des temps d'exécution de *Sat4j-CP* en utilisant le retrait des littéraux non pertinents par programmation dynamique et sans ce retrait (en échelle logarithmique). Les cercles et carrés représentent les instances cohérentes et contradictoires, respectivement. Les couleurs représentent les mêmes familles qu'à la figure 1.

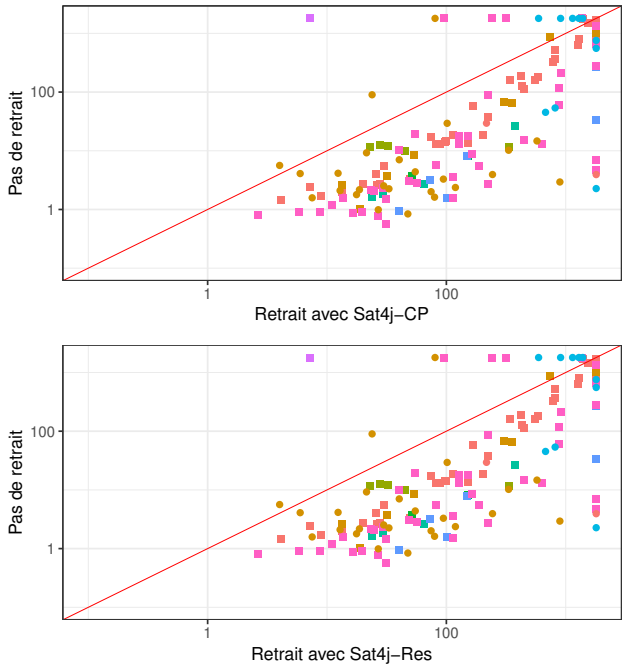


FIGURE 4 – Comparaison des temps d'exécution de *Sat4j-CP* en utilisant le retrait des littéraux non pertinents avec *Sat4j-CP* (en haut) et *Sat4j-Res* (en bas) et sans ce retrait (en échelle logarithmique). Les cercles et carrés représentent les instances cohérentes et contradictoires, respectivement. Les couleurs représentent les mêmes familles qu'à la figure 1.

modifiées. La figure 3 compare les temps d'exécution du solveur lorsque le retrait des littéraux non pertinents est réalisé par programmation dynamique avec ceux obtenus par le solveur sans ce retrait. Pour plus de clarté, les instances résolues en moins de 10 secondes par chacune des deux approches sont omises. Il en va de même pour celles n'étant résolues dans aucun cas.

Le diagramme de dispersion obtenu ne permet pas de tirer des conclusions aussi claires que celles résultant du diagramme en figure 2 : *Sat4j-CP* est assez sensible aux changements dans ses contraintes, donc modifier ses contraintes apprises peut modifier son espace de recherche, ce qui peut expliquer les différences de temps observées. Cependant, il montre que, en dépit de la complexité de la détection des littéraux non pertinents, activer leur retrait permet une amélioration des performances du solveur. En effet, son temps d'exécution est par exemple diminué pour chacune des instances de la famille `d_n_k`.

En revanche, comme le montre la figure 4, les approches utilisant un solveur pour détecter les littéraux non pertinents ne sont pas assez efficaces pour contrebalancer le coût de la détection des littéraux non pertinents.

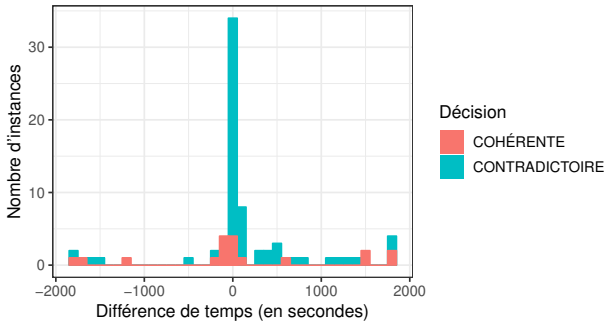


FIGURE 5 – Histogramme des différences de temps d'exécution entre *Sat4j-CP* où le retrait des littéraux non pertinents est activé ou pas. Les barres à gauche de 0 sont mieux résolues sans le retrait, et celles à droite de 0 sont mieux résolues avec.

5.3 Effets concrets sur le solveur

Puisque l'approche par programmation dynamique produit les meilleurs résultats, nous étudions ici ses effets pratiques sur le solveur. D'une part, comme le suggère la figure 3, les instances *faciles*, résolues en moins de 100 secondes, ne sont pas résolues aussi efficacement lorsque la détection est activée. C'est le prix à payer pour résoudre un problème calculatoirement difficile pendant l'analyse de conflit : comme les instances sont résolues rapidement, il n'y a pas assez de temps pour amortir son coût. D'autre part, les instances difficiles, et plus particulièrement celles étant incohérentes (il est bien connu que les instances cohérentes sont résolues par « chance »), sont résolues plus rapidement lorsque la détection est appliquée. Ce gain est illustré à la figure 5.

Notons que, bien que 7 instances n'ont pas pu être résolues avec le retrait activé, 15 de plus sont résolues avec cette fonctionnalité. Des statistiques sur ces instances sont données dans la table 1. Notons que l'instance `vertexcover-grid10x25` est omise car le solveur n'a pas fait d'affichage dans le temps imparti.

La table 1 révèle que la vitesse (i.e. le nombre d'affectations réalisées par le solveur chaque seconde) augmente pour les instances gagnées, et diminue pour les instances perdues. Dans les instances perdues, peu de littéraux sont détectés comme non pertinents, surtout lorsque l'on compare leur nombre à celui des instances gagnées, où l'on en détecte un grand nombre. Nous pouvons également remarquer que, pour ces instances, de nombreuses clauses sont identifiées, permettant ainsi au solveur d'optimiser ses structures de données internes. Les nombreuses saturations appliquées permettent également de réduire les coefficients, contribuant à améliorer l'efficacité de la résolution généralisée.

Instance	Contraintes inchangées	Contraintes modifiées	Littéraux retirés	Saturations déclenchées	Clauses détectées	Vitesse avec retrait (affect./s)	Vitesse sans retrait (affect./s)
3col-left3reg-l050-r049-n1	16286	10342	18030	10325	5544	**10561**	6595
5_6_19	31210	3959	10337	3957	3201	**2953**	2311
8_6_18	23065	3589	11622	3586	2904	**5023**	2249
compression16_11	2592	64	472	60	42	**22572**	1161
compression16_12	2596	54	336	53	36	**15992**	1199
ECgrid5x20split	35514	27841	72576	26734	24811	**1760**	597
sha1-size80-round21-1	6151	59	448	53	30	**6085**	1995
sha1-size80-round21-5	5817	41	193	35	20	**5712**	1886
sumineqArity3pyramidP0016	391	126	1270	122	120	**7334**	305
vertexcover-grid10x19-hard	7693	652	14025	629	619	**834**	236
vertexcover-grid10x19	1237	50	1255	47	45	**3056**	37
vertexcover-grid10x21	1615	50	1111	49	47	**4513**	50
vertexcover-grid10x29	4514	204	9309	191	181	**592**	291
vertexcover-grid8x17-hard	1641	85	1539	79	76	**5320**	105
sha1-size80-round21-4	8269	44	188	33	13	1973	**19504**
sha1-size96-round21-5	8066	59	262	42	22	2102	**4302**
sha1-size96-round21-6	8496	54	199	33	59	67	**10067**
vertexcover-grid10x13-hard	1075	40	1077	40	40	36	**234**
vertexcover-grid10x17	1598	46	977	4	33	89	**7261**
vertexcover-grid10x23	1231	24	547	19	15	72	**493**

TABLE 1 – Détails sur les instances **gagnées** et *perdues*

6 Conclusion et perspectives

Dans cet article, nous avons montré que les contraintes pseudo-booléennes, contrairement aux clauses et aux contraintes de cardinalité, peuvent contenir des littéraux non pertinents. En particulier, de tels littéraux peuvent être introduits dans les contraintes apprises par l'application de la résolution généralisée, même si la formule considérée en entrée n'en contient pas. Nous avons proposé plusieurs approches permettant de détecter ces littéraux, fondées soit sur un solveur PB, soit sur un algorithme de programmation dynamique résolvant *subset-sum*. Nos résultats expérimentaux montrent que l'approche par programmation dynamique est la technique la plus efficace parmi celles considérées. En dépit de sa complexité élevée, cette détection réduit le temps d'exécution de *Sat4j* sur de nombreuses instances de l'évaluation pseudo-booléenne 2016. Un tel gain peut s'expliquer par le fait que le retrait de ces littéraux conduit à une réduction de la taille des contraintes, à la fois en termes de nombre de littéraux et de taille de coefficients. Ce retrait peut également révéler, dans certains cas, des clauses ou des contraintes de cardinalité cachées. Combinées, ces opérations permettent d'obtenir des contraintes pour lesquelles la détection des propagations et l'application de la résolution généralisée sont plus efficaces. La présence de littéraux non pertinents indique également que les contraintes produites par *Sat4j* ne sont pas aussi fortes que ce qu'elles pourraient être. *RoundingSat*, quant à lui, produit moins de littéraux non pertinents. Cependant, des littéraux initialement non pertinents peuvent devenir *artificiellement* pertinents en appliquant un arrondi lors de la division de la contrainte. De ce fait, le nombre de littéraux non pertinents ne peut pas être considéré comme une mesure de la qualité des contraintes apprises. Des travaux supplémentaires sont nécessaires pour étudier la présence de littéraux non pertinents dans les contraintes intervenant dans l'analyse de conflit.

Notre but ultime serait de définir un système de preuve assurant de ne produire que des littéraux pertinents à partir de contraintes qui ne comportent que de tels littéraux. La principale difficulté est de trouver des règles qui n'affectent pas la pertinence des littéraux, contrairement à la règle de division. Notons que la restriction de cette règle à la division de contraintes dans lesquelles tous les coefficients sont divisibles par

le diviseur (et donc pour lesquelles seul le degré est arrondi) n'affecte pas la pertinence des littéraux.

D'un point de vue plus pragmatique, notre algorithme de détection pourrait être amélioré, par exemple en approximant *subset-sum* plutôt qu'en le résolvant de manière exacte. Cela pourrait permettre de supprimer systématiquement tous les littéraux non pertinents des contraintes produites par *Sat4j*. Rappelons qu'actuellement, notre algorithme de détection s'applique uniquement à la contrainte apprise à l'issue de l'analyse de conflit. Il serait alors possible de s'assurer qu'aucune des contraintes produites ne contienne de littéraux non pertinents.

Références

[1] Armin Biere, Daniel Le Berre, Emmanuel Lonca et Norbert Manthey : Detecting cardinality constraints in CNF. *In Theory and Applications of Satisfiability Testing*, pages 285–301, 2014.

[2] Donald Chai et Andreas Kuehlmann : A fast pseudo-Boolean constraint solver. *IEEE Trans. on CAD of Integrated Circuits and Systems*, pages 305–317, 2005.

[3] William Cook, Collette R. Coullard et György Turán : On the Complexity of Cutting-plane Proofs. *Discrete Appl. Math.*, pages 25–38, 1987.

[4] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest et Clifford Stein : *Introduction to Algorithms, Third Edition*. 2009.

[5] Yves Crama et Peter L. Hammer : *Boolean Functions : Theory, Algorithms, and Applications*. 2011.

[6] Heidi Dixon : *Automating Pseudo-Boolean Inference Within a DPLL Framework*. Thèse de doctorat, University of Oregon, 2004.

[7] Heidi E. Dixon et Matthew L. Ginsberg : Inference methods for a pseudo-boolean satisfiability solver. *In AAAI'02*, pages 635–640, 2002.

[8] Heidi E. Dixon, Matthew L. Ginsberg et Andrew J. Parkes : Generalizing boolean satisfiability I : background and survey of existing work. *Journal of Artificial Intelligence Research*, pages 193–243, 2004.

[9] Niklas Eén et Niklas Sörensson : An extensible sat-solver. *In Theory and Applications of Satisfiability Testing*, pages 502–518, 2004.

[10] Jan Elffers, Jesús Giráldez-Crú, Jakob Nordström et Marc Vinyals : Using combinatorial benchmarks to probe the reasoning power of pseudo-boolean solvers. *In Theory and Applications of Satisfiability Testing*, pages 75–93, 2018.

[11] Jan Elffers et Jakob Nordström : Divide and conquer : Towards faster pseudo-boolean solving. *In Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, pages 1291–1299, 2018.

[12] Ralph E. Gomory : Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Mathematical Society*, pages 275–278, 1958.

[13] Armin Haken : The intractability of resolution. *Theoretical Computer Science*, pages 297–308, 1985.

[14] John N. Hooker : Generalized resolution and cutting planes. *Annals of Operations Research*, pages 217–239, 1988.

[15] Daniel Le Berre et Anne Parrain : The SAT4J library, Release 2.2, System Description. *Journal on Satisfiability, Boolean Modeling and Computation*, pages 59–64, 2010.

[16] Vasco Manquinho et Olivier Roussel : The first evaluation of pseudo-boolean solvers (pb'05). *JSAT*, pages 103–143, 2006.

[17] Joao Marques-Silva et Karem A. Sakallah : Grasp : A search algorithm for propositional satisfiability. *IEEE Trans. Computers*, pages 220–227, 1999.

[18] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang et Sharad Malik : Chaff : Engineering an efficient sat solver. *In Proceedings of the 38th Annual Design Automation Conference*, pages 530–535, 2001.

[19] Jakob Nordström : On the Interplay Between Proof Complexity and SAT Solving. *ACM SIGLOG News*, pages 19–44, 2015.

[20] Olivier Roussel et Vasco M. Manquinho : Pseudo-Boolean and Cardinality Constraints. *In Handbook of Satisfiability*, chapitre 22, pages 695––733. 2009.

[21] Hossein M. Sheini et Karem A. Sakallah : Pueblo : A Hybrid Pseudo-Boolean SAT Solver. *JSAT*, pages 165–189, 2006.

[22] Marc Vinyals, Jan Elffers, Jesús Giráldez-Crú, Stephan Gocht et Jakob Nordström : In between resolution and cutting planes : A study of proof systems for pseudo-boolean SAT solving. *In Theory and Applications of Satisfiability Testing*, pages 292–310, 2018.