Adaptation des stratégies des solveurs SAT CDCL aux solveurs PB natifs

Daniel Le Berre, Romain Wallon

▶ To cite this version:

Daniel Le Berre, Romain Wallon. Adaptation des stratégies des solveurs SAT CDCL aux solveurs PB natifs. 16es Journées Francophones de Programmation par Contraintes (JFPC'21), Jun 2021, Nice (en ligne), France. hal-03295266

HAL Id: hal-03295266
https://hal.science/hal-03295266v1

Submitted on 21 Jul 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.

Actes JFPC 2021

Adaptation des stratégies des solveurs SAT CDCL aux solveurs PB natifs

Daniel Le Berre[1] **Romain Wallon**[2*]

[1] CRIL, Univ Artois & CNRS

[2] LIX, Laboratoire d'Informatique de l'X, École Polytechnique, Chaire X-Uber

leberre@cril.fr wallon@lix.polytechnique.fr

Résumé

Les solveurs pseudo-booléens (PB) travaillant nativement sur des contraintes PB sont fondés sur l'architecture CDCL à l'origine des hautes performances des solveurs SAT modernes. En particulier, ces solveurs PB utilisent non seulement une procédure d'analyse de conflits (utilisant les plans-coupes), mais également des stratégies complémentaires qui sont cruciales pour l'efficacité du solveur, comme les heuristiques de choix de variable, de suppression des contraintes apprises et de redémarrage. Cependant, ces stratégies sont le plus souvent réutilisées par les solveurs PB sans tenir compte de la forme particulière des contraintes PB qu'ils considèrent. Dans cet article, nous présentons et évaluons différentes manières d'adapter ces stratégies pour tenir compte des spécificités des contraintes PB tout en préservant le même comportement que dans le cadre clausal. Nous avons implanté ces stratégies dans deux solveurs différents, à savoir *Sat4j* (pour lequel nous considérons trois configurations) et *RoundingSat*. Nos expérimentations montrent que ces stratégies dédiées permettent d'améliorer, parfois significativement, les performances de ces solveurs, à la fois sur des problèmes de décision et d'optimisation.

Abstract

Current implementations of pseudo-Boolean (PB) solvers working on native PB constraints are based on the CDCL architecture which empowers highly efficient modern SAT solvers. In particular, such PB solvers not only implement a (cutting-planes-based) conflict analysis procedure, but also complementary strategies for components that are crucial for the efficiency of CDCL, namely branching heuristics, learned constraint deletion and restarts. However, these strategies are mostly reused by PB solvers without considering the particular form of the PB constraints they deal with. In this paper, we present and evaluate different ways of adapting CDCL strategies to take the specificities of PB constraints into account while preserving the behavior they have in the clausal setting. We implemented these strategies in two different solvers, namely *Sat4j* (for which we consider three configurations) and *RoundingSat*. Our experiments show that these dedicated strategies allow to improve, sometimes significantly, the performance of these solvers, both on decision and optimization problems.

1 Introduction

Le succès des solveurs SAT dits *modernes* a motivé la généralisation de l'architecture CDCL (*Conflict-Driven Clause Learning*, ou apprentissage de clauses guidé par les conflits) [30, 31, 13] à la résolution de problèmes pseudo-booléens (PB) [35]. La principale motivation derrière le développement de solveurs PB est que les solveurs SAT classiques sont fondés sur le système de preuve par *résolution*, qui est relativement *faible* : les instances difficiles pour ce système (tels que ceux nécessitant de « savoir compter », comme les formules du *principe du pigeonnier*, ou *principe des tiroirs* [20]) sont difficiles pour les solveurs SAT. Le système de preuve des plans-coupes [19, 21, 32] constitue une alternative plus puissante, permettant par exemple de résoudre des formules du principe du pigeonnier en un nombre linéaire d'étapes d'inférence. Plus généralement, ce système de preuve *p-simule* la résolution : toute preuve par résolution peut être simulée par une preuve du système des plans-coupes de taille polynomiale [10]. En théorie, les solveurs PB devraient donc être capables de trouver des preuves d'incohérence plus courtes, et donc d'être plus efficaces que les solveurs SAT classiques. En pratique cependant, les solveurs PB actuels ne parviennent pas à tenir les promesses de la théorie. En particulier, la plupart des solveurs PB [12, 9, 36, 25] im-

*Une grande partie des travaux présentés dans cet article ont été réalisés lorsque cet auteur était doctorant au CRIL

plantent un sous-ensemble du système des plans-coupes appelé *résolution généralisée* [21], qui permet d'étendre l'algorithme CDCL aux contraintes PB. Lorsqu'une contrainte devient conflictuelle, la règle de résolution généralisée est appliquée entre cette contrainte et la raison de la propagation de l'un de ses littéraux pour inférer une nouvelle contrainte conflictuelle. Cette opération est répétée jusqu'à ce qu'une contrainte assertive soit finalement produite. Cependant, les solveurs implantant cette procédure n'exploitent pas toute la puissance du système des plans-coupes [37], et sont moins performants que les solveurs fondés sur la résolution dans les compétitions PB [34].

Malgré les améliorations récemment apportées par *RoundingSat* [16] avec l'utilisation de la règle de *division* pendant l'analyse de conflit, les implantations actuelles du système des plans-coupes ont toujours un point faible majeur : elles sont équivalentes au système par résolution lorsqu'elles reçoivent une CNF en entrée. De plus, ces implantations sont plus complexes que le simple remplacement de la résolution dans l'analyse de conflit par la résolution généralisée : trouver *quelles* règles appliquer et *quand* n'est pas si évident que cela [17, 24]. En particulier, les solveurs PB doivent tenir compte de propriétés spécifiques aux contraintes PB et au système des plans-coupes pour les adapter à l'architecture CDCL. De plus, de nombreuses autres fonctionnalités des solveurs SAT CDCL sont requises pour garantir l'efficacité pratique de ces derniers (voir, par exemple, [15]). Dans cet article, nous nous concentrons sur ces fonctionnalités dans le cadre PB. A notre connaissance, peu de travaux étudient l'extension de ces composants aux solveurs PB : ils sont le plus souvent réutilisés tels que définis dans les solveurs SAT classiques, et adaptés juste assez pour fonctionner dans le solveur, sans considérer leur impact dans le contexte de la résolution de problèmes PB.

Dans la Section 3, nous introduisons de nouvelles variantes de l'*heuristique de choix de variable VSIDS* (*variable state independent decaying sum*) [31]. Ces variantes généralisent les propriétés de cette heuristique aux solveurs PB, en considérant les affectations des littéraux rencontrés. Dans la Section 4, nous proposons diverses stratégies de *suppression des contraintes apprises*, en définissant plusieurs nouvelles mesures visant à évaluer la qualité des contraintes apprises par le solveur. En particulier, nous considérons de nouvelles définitions de la mesure du *LBD* (*Literal Block Distance*) [2] qui, comme nous le montrons, n'est pas bien défini pour les contraintes PB. Dans la Section 5, nous utilisons ensuite ces nouvelles mesures pour détecter *quand* déclencher un *redémarrage*, en les utilisant dans des politiques *dynamiques* [3]. Enfin, dans la Section 6, nous évaluons empiriquement l'impact de ces différentes stratégies dans plusieurs solveurs PB.

2 Préliminaires

Nous considérons un cadre propositionnel classique défini sur un ensemble fini de variables propositionnelles $\mathcal{V}$. Un *littéral* ℓ est une variable $v \in \mathcal{V}$ ou sa négation $\bar{v}$. Les valeurs booléennes sont représentées par les entiers 1 (vrai) et 0 (faux), de sorte que $\bar{v} = 1 - v$.

Une *contrainte pseudo-booléenne (PB)* est une équation ou inéquation de la forme $\sum_{i=1}^{n} \alpha_i \ell_i \vartriangle \delta$, dans laquelle les *coefficients* α_i et le *degré* δ sont des entiers, les ℓ_i sont des littéraux et $\vartriangle \in \{<, \leq, =, \geq, >\}$. Une telle contrainte peut être *normalisée* en une conjonction de contraintes de la forme $\sum_{i=1}^{n} \alpha_i \ell_i \geq \delta$ dans laquelle les coefficients et le degré sont des entiers positifs. Dans la suite, nous supposons donc que toutes les contraintes sont normalisées. Une *contrainte de cardinalité* est une contrainte PB dont tous les coefficients sont égaux à 1 et une clause est une contrainte de cardinalité de degré 1. Cette définition montre que les contraintes PB généralisent les clauses, et que le raisonnement clausal est donc un cas particulier du raisonnement PB.

Les solveurs PB ont donc été conçus pour étendre l'algorithme CDCL des solveurs SAT classiques. En particulier, pendant la phase d'exploration, les solveurs PB *affectent* des variables. Dans la suite, nous notons $\ell(V@D)$ le fait que la valeur V est affecté au littéral ℓ au niveau de décision D, et $\ell(?@?)$ le fait que ℓ n'est pas affecté. L'affectation des variables se fait soit en *prenant une décision*, soit en *propageant* une valeur de vérité pour une variable. Dans ce contexte, la forme normalisée des contraintes PB est particulièrement commode pour détecter des propagations : comme pour les clauses, les propagations sont déclenchées par la falsification de littéraux dans une contrainte. Cependant, contrairement aux clauses, une contrainte PB peut propager des littéraux même si d'autres littéraux sont satisfaits ou non-affectés, comme dans cet exemple.

Exemple 1. *La contrainte* $5a(0@3) + 5b(?@?) + c(?@?) + d(?@?) + e(0@1) + f(1@2) \geq 6$ *propage le littéral* b *à* 1 *(vrai) sous l'affectation courante. Si* b *est affecté à* 0*, la contrainte* $5a(0@3) + 5b(0@3) + c(?@?) + d(?@?) + e(0@1) + f(1@2) \geq 6$ *devient conflictuelle. Dans les deux cas, observons que* f *est satisfait et que* c *et* d *ne sont pas affectés.*

Les solveurs PB implantent par ailleurs une procédure d'analyse de conflit permettant, grâce au système des plans-coupes, d'apprendre de nouvelles contraintes et de réaliser des retours-arrière non-chronologiques. Nous omettons délibérément sa description qui sort du cadre de cet article, et renvoyons le lecteur intéressé vers [35, 16, 24] pour plus de détails sur le sujet.

3 Heuristique de choix de variable

Un composant important d'un solveur SAT est son *heuristique de choix de variable* : pour trouver efficacement une solution ou une preuve d'incohérence, le solveur doit choisir les *bonnes* variables sur lesquelles prendre des décisions. La plupart des solveurs SAT actuels utilisent VSIDS [31] ou l'une de ses variantes [7], ou encore l'heuristique plus récente LRB [27]. Nous nous concentrons ici sur VSIDS, qui est celle adoptée par les solveurs que nous étudions dans la suite.

3.1 VSIDS et ses variantes dans les solveurs SAT

Dans l'implantation originale de VSIDS, chaque variable possède un *score* qui est incrémenté chaque fois qu'une nouvelle clause contenant cette variable est apprise. De plus, ces scores sont régulièrement divisés par 2 (en pratique, tous les 256 conflits), de manière à favoriser les variables apparaissant dans les clauses apprises les plus récentes. Au moment de choisir une variable, c'est celle ayant le plus haut score qui est sélectionnée.

La variante la plus commune de VSIDS est *exponential VSIDS* (EVSIDS), introduite par *MiniSat* [13]. Dans cette heuristique, une valeur g est choisie entre 1.01 et 1.2 au début de l'exécution du solveur. Quand une variable est rencontrée pendant l'analyse du i-ème conflit (soit dans la clause apprise, soit dans celles ayant servi à la produire), son score est mis à jour en lui ajoutant g^i. Cette mise à jour est souvent appelée *bumping*, ou *incrémentation*. Elle préserve le fait que les variables favorisées sont celles *impliquées* dans les conflits récents, tout en évitant le coût d'une division fréquente. Notons toutefois qu'une division régulière reste nécessaire pour éviter des dépassements de capacité, mais cette division reste moins fréquente que celle utilisée dans la version originale de VSIDS.

3.2 VSIDS dans les solveurs PB

Les solveurs PB actuels utilisent l'heuristique VSIDS (ou l'une de ses variantes) pour décider quelle sera la prochaine variable à affecter. En pratique, cette heuristique peut être utilisée en l'état par les solveurs PB, mais cela ne permet pas de prendre en compte toutes les informations contenues dans la contrainte, comme cela a été observé dans [9] (qui, cependant, ne propose pas d'heuristique plus adaptée). C'est pourquoi différentes variantes de cette heuristique ont été proposée.

Dans [11, Section 4.5], il est ainsi proposé d'ajouter, pour chaque variable apparaissant dans les *contraintes de cardinalité du problème original*, le degré de la contrainte au score *initial* de ces variables. Cette approche permet de compter le nombre d'occurrences des variables dans les clauses représentées par cette contrainte de cardinalité, qui correspond exactement à la valeur de son degré. Le score des variables des contraintes apprises n'est cependant incrémenté que de 1 (une seule des clauses sous-jacentes étant en général responsable du conflit analysé).

Exemple 2 (Tiré de [11, Section 4.5])**.** *Si la contrainte de cardinalité $a+b+c \geq 2$ est présente dans le problème original, le score de ses variables est incrémenté de 2. En effet, cette contrainte est équivalente à la conjonction des clauses $a+b \geq 1$, $a+c \geq 1$ et $b+c \geq 1$. Si cette contrainte est apprise, les scores ne sont incrémentés que de 1.*

Bien que cette heuristique soit plus spécifique que la version originale de VSIDS pour les problèmes PB, elle n'est cependant pas satisfaisante, car elle ne s'adapte pas bien aux implantations modernes de VSIDS (et en particulier, EVSIDS). Tout d'abord, comme seules les contraintes originales sont concernées, l'heuristique n'a aucune influence sur l'incrémentation du score des variables impliquées dans les conflits récents. Par ailleurs, la forme particulière des contraintes PB générales n'est pas prise en compte par cette heuristique. La principale raison du choix de ne considérer que les contraintes de cardinalité est ici que déterminer le nombre de clauses dans lesquelles un littéral d'une contrainte PB apparaît est difficile en général.

Une autre alternative, implantée dans *Pueblo* [36], est d'estimer l'importance relative d'un littéral dans la contrainte, en calculant le rapport de son coefficient par le degré de la contrainte. Cette valeur est ensuite ajouté au score de la variable.

Exemple 3. *Si le score de a est incrémenté dans la contrainte $5a + 5b + c + d + e + f \geq 6$, l'incrément est multiplié par $5/6$.*

Concernant *Sat4j* [25] et *RoundingSat* [16], ces deux solveurs implantent une approche plus classique d'EVSIDS, en incrémentant le score des variables rencontrées pendant l'analyse de conflit. Cependant, certains détails d'implantation diffèrent entre ces deux solveurs. En particulier, dans *Sat4j* le score d'une variable est incrémenté *chaque fois* qu'elle apparaît dans une clause rencontrée pendant l'analyse de conflit, tandis que *RoundingSat* ne l'incrémente qu'une seule fois (comme dans *MiniSat* [13]), et deux fois dans le cas des variables *éliminées* au cours de l'analyse de conflit.

3.3 Vers un meilleur VSIDS pour les solveurs PB

Comme mentionné plus haut, les implantations actuelles de VSIDS, et en particulier d'EVSIDS, sont conçues pour favoriser la sélection de variables impliquées dans les conflits récents. Lorsque seules des

clauses sont considérées, identifier ces littéraux est évident : les littéraux impliqués dans le conflit sont exactement ceux apparaissant dans la clause. Cependant, ce n'est plus le cas lorsque des contraintes PB sont considérés. En effet, étant donnée une contrainte PB, ses littéraux ne jouent pas le même rôle dans la contrainte de par la présence de coefficients, et n'ont donc pas nécessairement le même impact sur le conflit.

Une observation cruciale pour détecter les littéraux qui sont réellement *impliqués* dans un conflit est l'impact de l'affectation courante. En effet, dans les solveurs SAT classiques, tous les littéraux apparaissant dans les clauses rencontrées pendant l'analyse de conflit sont toujours *affectés*, et tous sauf un sont même *falsifiés*. Cependant, dans les contraintes PB, ce n'est pas toujours le cas (voir Exemple 1), et même des littéraux falsifiés peuvent être *ineffectifs* [24, Section 3.1].

Définition 1 (Littéral effectif)**.** *Étant donnée une contrainte PB conflictuelle (resp. assertive) χ, un littéral ℓ de χ est dit* effectif *dans χ s'il est falsifié et si le satisfaire ne préserve pas le conflit (resp. la propagation). ℓ est dit* ineffectif *s'il n'est pas effectif.*

Même si de tels littéraux apparaissent pendant l'analyse de conflit, ils ne jouent aucun rôle dans celui-ci, tout comme les variables associées. Nous proposons donc de prendre en compte l'affectation courante lors de la mise à jour des scores des variables à l'aide de trois nouvelles stratégies, à savoir *bump-assigned*, qui incrémente uniquement le score des variables affectées rencontrées pendant l'analyse de conflit, *bump-falsified*, qui incrémente uniquement le score des variables pour lesquelles un littéral apparaît falsifié pendant l'analyse de conflit, et *bump-effective*, qui incrémente uniquement le score des variables pour lesquelles un littéral apparaît effectif pendant l'analyse de conflit.

Exemple 4. *Lors de la mise à jour du score des variables de la contrainte* $5a(0@3) + 5b(1@3) + c(?@?) + d(?@?) + e(0@1) + f(1@2) \geq 6$*, la stratégie* bump-assigned *incrémente le score de a, b, e et f, la stratégie* bump-falsified *incrémente le score de a et e et la stratégie* bump-effective *incrémente le score de a.*

4 Suppression des contraintes apprises

Les solveurs PB, tout comme les solveurs SAT, doivent régulièrement *supprimer* les contraintes apprises pendant leur exécution. En effet, stocker ces contraintes peut non seulement accroître l'espace mémoire requis par le solveur, mais également ralentir la propagation unitaire. Il est alors crucial de choisir judicieusement *quelles* contraintes sont à supprimer.

Cette fonctionnalité est le plus souvent directement héritée des solveurs SAT dans les solveurs PB. Par exemple, *Pueblo* [36] utilise l'approche de *MiniSat* [13] fondée sur l'activité des contraintes apprises (les moins actives sont supprimées les premières), *Sat4j* [25] utilise également uns stratégie fondée sur l'activité, mais plus agressive à l'instar de *Glucose* [2], et *RoundingSat* [16] utilise une approche hybride qui lui est propre, fondée à la fois sur le *LBD* et l'activité (le second est utilisé pour départager des contraintes lorsqu'elles ont le même *LBD*). Dans d'autres solveurs PB tels que *pbChaff* [12] et *Galena* [9], la suppression (éventuelle) des contraintes apprises n'est pas documentée. Dans [9], une perspective d'affaiblissement des contraintes apprises est toutefois suggérée comme une alternative à leur suppression.

Cependant, bien que des mesures comme celle de l'activité peuvent être réutilisées telles quelles par les solveurs PB (elles ne tiennent pas compte de la représentation ni de la sémantique des contraintes qu'elles évaluent), pour d'autres mesures, il peut être pertinent de prêter attention à la forme particulière des contraintes PB. Cette section étudie deux principales approches dans cette direction.

4.1 Mesures fondées sur la taille

Dans les solveurs SAT classiques, les mesures fondées sur la taille suppriment les clauses les plus longues, c'est-à-dire, celles contenant de nombreux littéraux. Ces clauses sont faibles, notamment du point de vue des propagations : une propagation ne peut être déclenchée qu'après que de nombreux littéraux ont été falsifiés. Lorsque l'on considère des contraintes PB, ce n'est plus le cas. En effet, rappelons qu'une contrainte PB peut propager des littéraux alors que d'autres littéraux ne sont pas encore affectés, et que le nombre de littéraux d'une telle contrainte ne présume en rien de sa force.

Une autre raison qui a motivé l'utilisation de mesures fondées sur la taille est que les longues clauses sont coûteuses à maintenir, ce qui est aussi vrai dans le cas des contraintes PB. En particulier, la taille de ces contraintes prend aussi en compte la taille des coefficients, qui n'est pas négligeable : comme les coefficients peuvent devenir très grands pendant l'analyse de conflit, l'utilisation de la précision arbitraire est requise pour les représenter. Cette représentation peut ralentir les opérations arithmétiques, et donc l'analyse de conflit réalisée par le solveur. Différentes approches ont été étudiées pour limiter l'accroissement des coefficients, comme celles fondées sur les règles de division [16] ou d'affaiblissement [24]. Cependant, ces approches conduisent à l'inférence de contraintes plus faibles. En utilisant des mesures de qualité prenant en compte la taille des coefficients, il est possible de favoriser l'apprentissage de contraintes ayant des « petits » coefficients. Dans ce but, nous définissons ci-dessous des

mesures fondées sur le degré des contraintes apprises : *degree*, qui évalue la qualité d'une contrainte apprise par la *valeur* de son degré, et *degree-bits*, qui évalue la qualité d'une contrainte apprise par la *taille* de son degré, mesurée par le nombre minimum de bits nécessaires pour le représenter. L'intérêt de cette dernière mesure est d'évaluer plus finement l'espace mémoire utilisé pour la représentation du degré (qui permet d'estimer le coût des opérations arithmétiques), mais également de pouvoir représenter les valeurs fournies par cette mesure en utilisant une précision fixe.

Dans les deux cas, plus le degré est petit, meilleure est la contrainte. En effet, il est bien connu que le degré d'une contrainte PB est une borne supérieure des coefficients de cette contrainte (par la règle dite de *saturation*), de sorte que ne considérer que le degré est suffisant pour cette mesure, qui vise à privilégier des contraintes ayant des petits coefficients pour préserver l'efficacité des opérations arithmétiques réalisées.

Exemple 5. *Les mesures de qualités fondées sur le degré de la contrainte* $5a + 5b + c + d + e + f \geq 6$ *sont* 6 *pour la mesure* degree*, et* 3 *pour la mesure* degree-bits *(comme la représentation binaire de* 6*, à savoir* 110*, requiert* 3 *bits).*

4.2 Mesures fondées sur le LBD

Une autre mesure permettant d'évaluer les clauses apprises par un solveur SAT est le LBD [2].

Définition 2 (LBD)**.** *Considérons une* clause γ *et l'affectation de ses littéraux. Soit* π *une partition des littéraux, calculée à partir du niveau de décision des littéraux. Le* LBD *de* γ *est le nombre d'éléments de* π*.*

Le LBD d'une clause est calculé la première fois lors de l'apprentissage de cette clause, et est ensuite mis à jour chaque fois que celle-ci est utilisée comme raison. Dans ce contexte, les meilleures clauses sont celles de plus petit LBD. Cette approche tire parti du fait que tous les littéraux d'une clause conflictuelle sont falsifiés, et que lorsque la clause est utilisée comme raison, seul le littéral propagé n'est pas falsifié, mais son niveau de décision est aussi celui d'un autre littéral, qui lui est falsifié et est à l'origine de la propagation. Lorsque l'on considère des contraintes PB, ce n'est plus le cas. Le LBD n'est donc pas bien défini pour ces contraintes. Pour pouvoir l'utiliser comme une mesure de la qualité des contraintes apprises, nous devons donc prendre en compte les littéraux non falsifiés apparaissant dans ces contraintes. Dans ce but, nous introduisons cinq nouvelles définitions du LBD. Pour commencer, nous considérons d'abord une sorte de définition par défaut du LBD, qui ne prend en compte que les littéraux affectés. Cette définition était utilisée dans la première version de *RoundingSat* [16].

Définition 3 (LBD_a)**.** *Considérons une* contrainte χ *et l'affectation de ses littéraux. Soit* π *une partition des littéraux* affectés*, calculée à partir du niveau de décision des littéraux. Le* LBD_a *de* χ *(« a » pour « affecté ») est le nombre d'éléments de* π*..*

Nous pouvons également supposer que les littéraux non affectés sont en fait affectés à un niveau de décision « imaginaire ». Ce niveau de décision peut être le même pour tous les littéraux, ou pas.

Définition 4 (LBD_s)**.** *Considérons une* contrainte χ *et l'affectation de ses littéraux. Soit* π *une partition des littéraux* affectés*, calculée à partir du niveau de décision des littéraux. Soit* n *le nombre d'éléments de* π*. Le* LBD_s *de* χ *(« s » pour « similaire ») est* n *si tous les littéraux de* χ *sont affectés, et* $n + 1$ *sinon.*

Définition 5 (LBD_d)**.** *Considérons une* contrainte χ *et l'affectation de ses littéraux. Soit* π *une partition des littéraux* affectés*, calculée à partir du niveau de décision des littéraux. Soit* n *le nombre d'éléments de* π*. Le* LBD_d *de* χ *(« d » pour « différent ») est* $n + u$*, où* u *est le nombre de littéraux non affectés dans* χ*.*

Une autre extension possible du LBD est d'uniquement considérer les littéraux falsifiés, comme dans la version actuelle de *RoundingSat*.

Définition 6 (LBD_f)**.** *Considérons une* contrainte χ *et l'affectation de ses littéraux. Soit* π *une partition des littéraux* falsifiés*, calculée à partir du niveau de décision des littéraux. Le* LBD_f *de* χ *(« f » pour « falsifié ») est le nombre d'éléments de* π*.*

La définition ci-dessus part de l'observation que, lorsqu'une clause est apprise, tous les littéraux de cette clause sont falsifiés. Ils sont par ailleurs également effectifs, de sorte que nous pouvons également restreindre le calcul du LBD à ces littéraux.

Définition 7 (LBD_e)**.** *Considérons une* contrainte χ *et l'affectation de ses littéraux. Soit* π *une partition des littéraux* effectifs*, calculée à partir du niveau de décision des littéraux. Le* LBD_e *de* χ *(« e » pour « effectif ») est le nombre d'éléments de* π*.*

Exemple 6. *Les différentes mesures de* LBD*de la contrainte* χ *donnée par* $5a(0@3) + 5b(1@3) + c(?@?) + d(?@?) + e(0@1) + f(1@2) \geq 6$ *sont :*

- $LBD_a(\chi) = |\{\{a, b\}, \{e\}, \{f\}\}| = 3$
- $LBD_s(\chi) = |\{\{a, b\}, \{c, d\}, \{e\}, \{f\}\}| = 4$
- $LBD_d(\chi) = |\{\{a, b\}, \{c\}, \{d\}, \{e\}, \{f\}\}| = 5$
- $LBD_f(\chi) = |\{\{a\}, \{e\}\}| = 2$
- $LBD_e(\chi) = |\{\{a\}\}| = 1$

Remarquons que les mesures de *LBD* présentées dans cette section sont des *extensions* de la définition originale du *LBD* (Définition 2), dans le sens où elles coïncident toutes lorsqu'elles sont calculées sur des clauses.

Nous proposons donc d'utiliser des stratégies de suppression fondées sur les mesures définies dans cette section (fondées sur le degré ou le *LBD*).

5 Redémarrages

Les redémarrages sont une fonctionnalité très puissante des solveurs SAT [18]. Bien qu'elle ne soit pas complètement comprise, elle semble requise pour mieux exploiter le pouvoir d'inférence de la résolution [33, 14, 1]. Redémarrer revient alors à oublier toutes les décisions faites par le solveur. Le principal avantage de cette approche est que les mauvaises décisions prises au début de l'exécution peuvent être annulées pour éviter au solveur d'être « bloqué » dans une sous-partie de l'espace de recherche. Dans ce but, plusieurs stratégies de redémarrage ont été proposées [8], soit statiques comme celles utilisant la suite de Luby [28, 22] soit dynamiques comme dans *PicoSAT* [5] ou *Glucose* [3]. Dans cette section, nous considérons ces dernières, en utilisant les mesures définies dans la Section 4.

De telles stratégies ne sont pas exploitées dans les solveurs PB actuels. Dans *pbChaff* [12] ou *Galena* [9], il n'est pas fait mention d'une implantation des redémarrages. Concernant *Pueblo* [36], qui est fondé sur *MiniSat* [13], il est très probable qu'il en hérite sa stratégie de redémarrage, même s'il n'en est pas fait mention dans [36] non plus. Concernant les solveurs plus récents, *Sat4j* [25] implante la politique de redémarrage statique et agressive de *PicoSAT* [6], tandis que *RoundingSat* [16] utilise la suite de Luby [28, 22]. Notons que ces deux stratégies sont indépendantes du type des contraintes apprises, (comme elles sont statiques), et peuvent donc être réutilisées en l'état.

Dans cette section, nous proposons au contraire d'utiliser l'approche de *Glucose* [3]. Dans ce solveur, un redémarrage est déclenché en fonction de l'évolution de la qualité des contraintes apprises : si celle-ci diminue, le solveur est vraisemblablement en train d'explorer la mauvaise partie de l'espace de recherche. En pratique, *Glucose* mesure cette qualité à l'aide du *LBD* (voir Définition 2). Pour mesurer son évolution, le *LBD* moyen est calculé sur les (100) dernières clauses apprises. Si cette moyenne est supérieure de 70% à la moyenne des *LBD* de toutes les clauses apprises, un redémarrage doit être réalisé.

Nous exploitons donc les mesures définies dans la section précédente pour définir des stratégies de redémarrage fondées sur ces mesures, en suivant la politique de *Glucose* décrite ci-dessus.

6 Résultats expérimentaux

Cette section présente des résultats expérimentaux des différentes stratégies introduites dans cet article et implantées dans deux solveurs PB, en l'occurrence *Sat4j* [25] et *RoundingSat* [16]. Nos expérimentations ont été exécutées sur un cluster équipé de processeurs quadricœurs Intel XEON X5550 (2.66 GHz, 8 Mo de cache). Le temps d'exécution était limité à 1200 secondes et la mémoire était limitée à 32 Go.

Par souci d'espace, cette section se limite aux performances des stratégies permettant d'améliorer le plus les solveurs considérés. Le lecteur intéressé pourra se référer aux résultats détaillés de nos expérimentations qui sont publiquement accessibles [26].

6.1 Configuration des solveurs

Commençons par décrire nos implantations des différentes stratégies dans *Sat4j* [25], disponibles sur son dépôt [1]. Pour ce solveur, nous considérons trois configurations, à savoir *Sat4j-GeneralizedResolution*, *Sat4j-RoundingSat* et *Sat4j-PartialRoundingSat* [24]. Pour ces trois configurations, les stratégies par défaut sont une heuristique de choix de variable qui incrémente le score de toutes les variables apparaissant dans les contraintes rencontrées pendant l'analyse de conflit (chaque fois que ces variables sont rencontrées), une stratégie de suppression des contraintes apprises considérant leur activité [13] (les contraintes supprimées sont celles moins impliquées dans les conflits récents), et la politique de redémarrage statique de *PicoSAT* [6].

Nos expérimentations ont montré que les meilleures stratégies pour *Sat4j-GeneralizedResolution* sont l'heuristique *bump-effective*, la stratégie de suppression des contraintes apprises utilisant le LBD_s, et la politique de redémarrage dynamique fondée sur la mesure *degree*. Pour *Sat4j-RoundingSat* et *Sat4j-PartialRoundingSat*, il s'agit de l'heuristique *bump-assigned*, la stratégie de suppression des contraintes apprises utilisant la mesure *degree-bits* et la politique statique de redémarrage de *PicoSAT* [6]. Dans la suite, nous considérons les combinaisons de ces stratégies comme la configuration `best` des différents solveurs mentionnés. Notons toutefois que toutes les combinaisons des stratégies n'ont pas été testées, compte-tenu des ressources de temps de calcul

1. `https://gitlab.ow2.org/sat4j/sat4j/tree/cdcl-strategies`

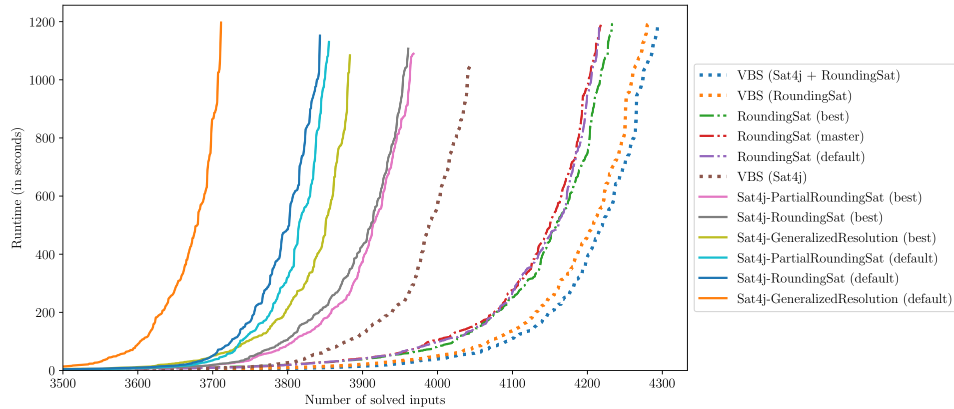

FIGURE 1 – « Cactus plot » de différentes configurations de *Sat4j* et de *RoundingSat* sur des problèmes de décision. Pour plus de lisibilité, les premières 3500 instances (faciles) sont ignorées.

nécessaires, et qu'il est possible que cette configuration ne soit pas la meilleure possible.

Pour *RoundingSat* [16], notre implantation est disponible dans un dépôt dédié [2]. Nous avons modifié ce solveur à partir du commit `a17b7d0e` (ci-après dénommé `master`) pour permettre l'utilisation des stratégies présentées dans cet article. La configuration `default` de ce solveur correspond à cette version modifiée de *RoundingSat* configurée avec les stratégies par défaut de sa version originale. Plus précisément, l'heuristique de choix de variable incrémente le score de toutes les variables apparaissant dans les contraintes rencontrées pendant l'analyse de conflit, (une seule fois, et deux fois si elles sont éliminées), la stratégie de suppression des contraintes apprises combinant leur LBD_f et leur activité, ainsi que la politique de redémarrage fondée sur la suite de Luby (avec un facteur 100) [22].

La combinaison des meilleures stratégies pour ce solveur (ci-après dénommée `best`) se compose, d'après nos expérimentations, de l'heuristique *bump-assigned* (avec une incrémentation du score des variables à chaque fois qu'elles sont rencontrées), et des stratégies de suppression des contraintes et de redémarrage fondées sur le LBD_e.

6.2 Problèmes de décision

Considérons dans un premier temps les performances des différents solveurs sur des problèmes de décision. Dans ce but, nous les avons exécutés sur la totalité des problèmes de décision utilisés dans les compétitions PB depuis la première édition [29, 34] et contenant des « petits » entiers, pour un total de 5582 instances. La Figure 1 donne les résultats obtenus pour les différents solveurs, dans leurs configurations `default` et `best`.

Le « cactus plot » montre que les différentes configurations de *Sat4j* sont significativement améliorées par nos stratégies dédiées. Notons de plus que la meilleure configuration de *Sat4j-GeneralizedResolution* est plus efficace que les implantations par défaut de *RoundingSat* dans *Sat4j*. Nous pouvons également noter une légère amélioration de *RoundingSat* par rapport à sa configuration par défaut, qui n'est cependant pas aussi importante que dans *Sat4j*.

Remarquons que combiner les meilleures stratégies n'est pas suffisant pour obtenir le meilleur de toutes les stratégies étudiées. En particulier, pour chaque fonctionnalité considérée, le *VBS* (pour *Virtual Best Solver*) des différentes stratégies, c'est-à-dire, le solveur obtenu en sélectionnant la meilleure stratégie pour chaque instance, a de bien meilleures performances que les stratégies considérées individuellement. Ce constat s'applique par ailleurs à toutes les configurations de *Sat4j* et *RoundingSat*. Ceci suggère qu'aucune stratégie n'est meilleure que les autres sur toutes les instances, et qu'elles sont en fait complémentaires.

6.3 Problèmes d'optimisation

Considérons maintenant les performances des différents solveurs sur des problèmes d'optimisation. Dans ce but, nous avons exécuté ces solveurs sur la totalité des problèmes d'optimisation utilisés dans les compétitions PB depuis la première édition [29, 34] et contenant des « petits » entiers, pour un total de 4374 instances. Au vu du temps de calcul considérable nécessaire pour réaliser nos expérimentations exhaustives sur les problèmes de décision (plus de 8 ans CPU), nous considérons ici uniquement les meilleures configurations des solveurs identifiées sur les problèmes de

2. https://gitlab.com/pb-cdcl-strategies/roundingsat/-/tree/cdcl-strategies

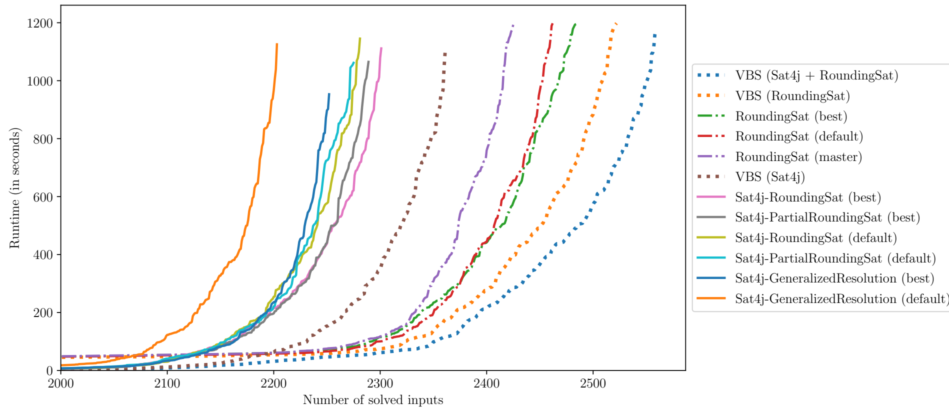

FIGURE 2 – « Cactus plot » de différentes configurations de *Sat4j* et de *RoundingSat* sur des problèmes d'optimisation. Pour plus de lisibilité, les premières 2000 instances (faciles) sont ignorées.

décision (ce qui a demandé malgré tout 9 mois CPU de calcul). La Figure 2 donnent les résultats obtenus.

Comme pour les problèmes de décision, nous pouvons observer sur le cactus plot que tous les solveurs sont améliorés par les stratégies dédiées sur les problèmes de décision, avec une amélioration particulièrement importante pour *Sat4j-GeneralizedResolution*.

6.4 Discussion

Sans surprise, la stratégie ayant le plus d'impact, en particulier sur *Sat4j*, est l'heuristique de choix de variable. En pratique, il s'agit de la stratégie individuelle qui améliore le plus les performances de *Sat4j*. Par exemple, nous avons constaté que *Sat4j-GeneralizedResolution* résout les problèmes (d'optimisation) de la famille `factor` beaucoup plus efficacement grâce à la stratégie *bump-effective* (modifier les stratégies de suppression et de redémarrage ne change presque rien pour cette famille). Nous avons étudié plus en détail le comportement du solveur sur ces instances pour mieux comprendre ce gain, et nous nous sommes aperçus que la production de littéraux non-pertinents (c'est-à-dire, des littéraux apparaissant dans la contrainte, mais n'affectant jamais sa valeur de vérité, quelque soit leur affectation) pénalise fortement le solveur sur cette famille. Il est connu que ces littéraux peuvent impacter la taille des preuves produites par les solveurs PB [23]. Nos expérimentations montrent ici qu'ils peuvent aussi polluer l'heuristique du solveur. Notons en effet que l'heuristique *bump-effective* n'incrémente jamais le score de ces littéraux, qui sont toujours ineffectifs. Cette heuristique propose donc une nouvelle approche pour composer avec ces littéraux.

L'impact important des variantes d'EVSIDS dans *Sat4j* peut aussi expliquer pourquoi le gain obtenu avec *RoundingSat* est plus limité. En effet, les affaiblissements agressifs appliqués par *RoundingSat* (qui sont par ailleurs plus agressifs que ceux pratiqués dans *Sat4j-RoundingSat* et *Sat4j-PartialRoundingSat*) ont tendance à identifier les littéraux impliqués dans le conflit. Ceci est particulièrement visible si l'on regarde le comportement des différentes stratégies de *bumping* dans *RoundingSat* : il n'y a presque aucune différence entre elles. Ceci suggère que le gain sur ce solveur est en grande partie obtenu par les stratégies de suppression des contraintes apprises et de redémarrage, dont l'impact sur les performances du solveur est plus limité que celui de l'heuristique de choix de variable en général.

Nous avons notamment pu observer que, dans *Sat4j*, ne jamais supprimer les contraintes apprises est en fait préférable à la stratégie (par défaut) supprimant les contraintes de faible activité. Cela peut s'expliquer par le fait que les solveurs PB sont souvent plus lents en pratique que le solveurs SAT, car les opérations qu'ils doivent réaliser, comme la détection des propagations ou l'application de la résolution généralisée, sont plus complexes que leur pendant dans les solveurs SAT. Cela signifie que le nombre de conflits par seconde dans un solveur PB est plus faible que dans un solveur SAT, tout comme le nombre de contraintes apprises. En conséquence, les solveurs PB n'ont pas besoin d'effacer des contraintes aussi souvent qu'un solveur SAT.

De plus, il est important de noter que les différentes stratégies considérées sont souvent très liées au sein du solveur, et peuvent donc interagir entre elles. C'est particulièrement vrai pour les stratégies de suppression des contraintes apprises et de redémarrage, qui utilisent les mêmes mesures de qualité. Par exemple, alors que les meilleures stratégies individuelles dans *RoundingSat* sont la politique de redémarrage de *PicoSAT* et la suppression utilisant le LBD_s, le meilleur gain est obtenu

en utilisant le LBD_e pour ces deux fonctionnalités.

Une autre conséquence des liens étroits entre les différentes stratégies dans le solveur est que des détails d'implantation peuvent avoir des effets de bord inattendus sur les performances du solveur. Par exemple, pour implanter les nouvelles stratégies dans *RoundingSat*, nous avons dû adapter son code et changer certaines structures de données dans l'heuristique de choix de variable (en remplaçant un ensemble ordonné par une structure associative ne préservant pas l'ordre), faisant que les scores des *mêmes* variables sont incrémentés, mais dans un *ordre différent*. Comme l'ordre d'insertion et de mise à jour des variables est utilisé comme départage par EVSIDS, l'ordre dans lequel les variables sont sélectionnées varie entre les configurations `master` et `default` du solveur, ce qui rend plus difficile l'interprétation des résultats de *RoundingSat*, notamment dans le cas des problèmes d'optimisation.

7 Conclusion

Dans cette article, nous avons présenté différentes heuristiques, stratégies de suppression des contraintes apprises et politiques de redémarrage dédiées à la résolution native de problèmes PB. Ces stratégies généralisent celles implantées dans les solveurs SAT, et sont conçues pour tenir compte des propriétés des contraintes PB pour s'adapter au contexte CDCL. Nos expérimentations ont montré que l'un des aspects essentiels des contraintes PB à considérer est l'impact de l'affectation courante sur leurs littéraux. C'est en particulier vrai pour les heuristiques fondées sur EVSIDS, mais aussi pour les stratégies de suppression des contraintes apprises et de redémarrage, au travers de nouvelles mesures de LBD. Combinées, ces stratégies permettent d'améliorer les performances de *RoundingSat* et *Sat4j*, avec des améliorations significatives pour ce dernier, à la fois sur des problèmes de décision et d'optimisation.

Néanmoins, aucune des ces stratégies n'est meilleure que les autres sur toutes les instances : leur VBS est clairement meilleur que chaque stratégie individuelle, même en considérant leur association. Cet article a cependant montré qu'une meilleure adaptation de ces stratégies aux solveurs PB peut améliorer leurs performances. Une perspective dans ce domaine serait de trouver de meilleures manières d'adapter les différentes stratégies considérées, voire d'en définir de nouvelles conçues *spécifiquement* pour les solveurs PB. Une autre piste à explorer est de trouver comment combiner au mieux ces stratégies pour en obtenir le meilleur, tout en considérant les interactions qu'elles peuvent avoir, par exemple en utilisant des techniques de configuration dynamique d'algorithmes pour sélectionner les stratégies les plus appropriées en fonction de l'état du solveur [4].

Remerciements

Les auteurs remercient les relecteurs pour leurs nombreux commentaires ayant permis d'améliorer cet article. Ces travaux ont partiellement bénéficié du soutien du Ministère de l'Enseignement Supérieur et de la Recherche et du Conseil Régional des Hauts-de-France au travers du « Contrat de Plan État Région (CPER) DATA ». Cet article bénéficie du soutien de la Chaire « Integrated Urban Mobility » portée par l'X - École Polytechnique et La Fondation de l'École Polytechnique, et soutenue par Uber. La responsabilité des Partenaires de la Chaire ne peut en aucun cas être mise en cause en raison du contenu de la présente communication, qui n'engage que son auteur.

Références

[1] Albert Atserias, Johannes Klaus Fichte et Marc Thurley : Clause-Learning Algorithms with Many Restarts and Bounded-Width Resolution. *JAIR*, 40:353–373, 2011.

[2] Gilles Audemard et Laurent Simon : Predicting Learnt Clauses Quality in Modern SAT Solvers. *In Proceedings of IJCAI'09*, pages 399–404, 2009.

[3] Gilles Audemard et Laurent Simon : Refining Restarts Strategies for SAT and UNSAT. *In* Michela Milano, éditeur : *Proceedings of CP'12*, pages 118–126, 2012.

[4] André Biedenkapp, H. Furkan Bozkurt, Theresa Eimer, Frank Hutter et Marius Lindauer : Dynamic Algorithm Configuration : Foundation of a New Meta-Algorithmic Framework. *In Proceedings of ECAI'20*, 2020.

[5] Armin Biere : Adaptive Restart Strategies for Conflict Driven SAT Solvers. *In Proceedings of SAT 2008*, pages 28–33, 2008.

[6] Armin Biere : PicoSAT Essentials. *JSAT*, 4(2-4):75–97, 2008.

[7] Armin Biere et Andreas Fröhlich : Evaluating CDCL variable scoring schemes. *In Proceedings of SAT'15*, pages 405–422, 2015.

[8] Armin Biere et Andreas Fröhlich : Evaluating CDCL Restart Schemes. *In Proceedings of Pragmatics of SAT 2015 and 2018*, volume 59 de *EPiC Series in Computing*, pages 1–17, 2019.

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

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

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

[12] Heidi E. DIXON et Matthew L. GINSBERG : Inference Methods for a Pseudo-Boolean Satisfiability Solver. *In Proceedings of AAAI'02*, pages 635–640, 2002.

[13] Niklas EÉN et Niklas SÖRENSSON : An Extensible SAT-solver. *In Proceedings of SAT'04*, pages 502–518, 2004.

[14] Jan ELFFERS, Jesús GIRÁLDEZ-CRÚ, Jakob NORDSTRÖM et Marc VINYALS : Using Combinatorial Benchmarks to Probe the Reasoning Power of Pseudo-Boolean Solvers. *In Proceedings of SAT'18*, pages 75–93, 2018.

[15] Jan ELFFERS, Jesús GIRÁLDEZ-CRU, Stephan GOCHT, Jakob NORDSTRÖM et Laurent SIMON : Seeking Practical CDCL Insights from Theoretical SAT Benchmarks. *In Proceedings of IJCAI'18*, pages 1300–1308, 2018.

[16] Jan ELFFERS et Jakob NORDSTRÖM : Divide and Conquer : Towards Faster Pseudo-Boolean Solving. *In Proceedings of IJCAI'18*, pages 1291–1299, 2018.

[17] Stephan GOCHT, Jakob NORDSTRÖM et Amir YEHUDAYOFF : On Division Versus Saturation in Pseudo-Boolean Solving. *In Proceedings of IJCAI'19*, pages 1711–1718, 2019.

[18] Carla P. GOMES, Bart SELMAN et Henry KAUTZ : Boosting Combinatorial Search through Randomization. *In Proceedings of AAAI'98*, page 431–437, 1998.

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

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

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

[22] Jinbo HUANG : The Effect of Restarts on the Efficiency of Clause Learning. *In Proceedings of IJCAI'07*, pages 2318–2323, 2007.

[23] Daniel LE BERRE, Pierre MARQUIS, Stefan MENGEL et Romain WALLON : On Irrelevant Literals in Pseudo-Boolean Constraint Learning. *In Proceedings of IJCAI'20*, pages 1148–1154, 2020.

[24] Daniel LE BERRE, Pierre MARQUIS et Romain WALLON : On Weakening Strategies for PB Solvers. *In Proceedings of SAT'20*, pages 322–331, 2020.

[25] Daniel LE BERRE et Anne PARRAIN : The SAT4J library, Release 2.2, System Description. *JSAT*, pages 59–64, 2010.

[26] Daniel LE BERRE et Romain WALLON : On Dedicated CDCL Strategies for PB Solvers Companion Artifact, mai 2021.

[27] Jia Hui LIANG, Vijay GANESH, Pascal POUPART et Krzysztof CZARNECKI : Learning Rate Based Branching Heuristic for SAT Solvers. *In Proceedings of SAT'16*, pages 123–140, 2016.

[28] Michael LUBY, Alistair SINCLAIR et David ZUCKERMAN : Optimal speedup of Las Vegas algorithms. *In Information Processing Letters*, pages 173–180, 1993.

[29] Vasco MANQUINHO et Olivier ROUSSEL : The First Evaluation of Pseudo-Boolean Solvers (PB'05). *JSAT*, pages 103–143, 2006.

[30] Joao MARQUES-SILVA et Karem A. SAKALLAH : GRASP : A Search Algorithm for Propositional Satisfiability. *IEEE Trans. Computers*, pages 220–227, 1999.

[31] Matthew W. MOSKEWICZ, Conor F. MADIGAN, Ying ZHAO, Lintao ZHANG et Sharad MALIK : Chaff : Engineering an Efficient SAT Solver. *In Proceedings of DAC'01*, pages 530–535, 2001.

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

[33] Knot PIPATSRISAWAT et Adnan DARWICHE : On the power of clause-learning SAT solvers as resolution engines. *Artif. Intell.*, 175(2):512–525, 2011.

[34] Olivier ROUSSEL : Pseudo-Boolean Competition 2016. http://www.cril.fr/PB16/, 2016 (accessed May 20, 2020).

[35] Olivier ROUSSEL et Vasco M. MANQUINHO : Pseudo-Boolean and Cardinality Constraints. *In* Armin BIERE, Marijn HEULE, Hans van MAAREN et Toby WALSH, éditeurs : *Handbook of Satisfiability*, volume 185 de *Frontiers in Artificial Intelligence and Applications*, pages 695–733. IOS Press, 2009.

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

[37] 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 Proceedings of SAT'18*, pages 292–310, 2018.