Pseudo-Boolean Reasoning and Compilation

DOCTORAL THESIS

presented and publicly defended on December 14th, 2020

in partial fulfillment of requirements for the degree of

Doctor of Philosophy of Artois University
in the subject of Computer Science

by

Romain Wallon

Centre de Recherche en Informatique de Lens – CNRS UMR 8188
Université d'Artois, rue Jean Souvraz, S.P. 18 F-62307, Lens Cedex France
Secrétariat : +33 (0)3 21 79 17 23
`http://www.cril.univ-artois.fr`

Mise en page avec memcril (B. Mazure, CRIL) et thloria (D. Roegel, LORIA).

Acknowledgements

I would like to thank the members of the doctoral committee: Jakob Nordström, who invited me at KTH at the beginning of my thesis and with whom, together with his group, we had the opportunity to discuss to improve pseudo-Booolean solving and who accepted to review my thesis; João Marques-Silva, the father of CDCL, whose high level of requirements made me stress a little bit during the review process; and Nadia Creignou, who accepted to chair the committee.

I also would like to thank my advisors, Daniel, Pierre and Stefan, who guided me during the three years of my thesis, and taught me so many things about pseudo-Boolean solving and research in general.

I also thank all my friends and colleagues at CRIL for the numerous inspiring discussions we had during coffee (or tea) breaks: the two other members of the "WWF" team, Hugues and Thibault, with whom I participated to programming contests and developed the *Metrics* library, and all other PhD students, Alexis, Alix, Anasse, Anis, Ikram, Marie, Ryma and Thanh. Without all of you, the three years of the thesis would not have been the same.

I also give a particular thank you to my family, especially my parents, who supported me during all my studies.

Résumé

La représentation d'informations de nature propositionnelle est une tâche importante en intelligence artificielle, et de nombreux langages ont été mis au point à cet égard. Ces langages correspondent à différents compromis entre expressivité (c'est-à-dire, ce qu'il est possible de représenter avec ces langages) et efficacité du raisonnement sur l'information représentée. Parmi les langages existants, celui des formules en *forme normale conjonctive* (ou CNF pour *Conjunctive Normal Form*) est largement utilisé, car il fournit une manière à la fois simple et pratique de représenter des contraintes portant sur des variables booléennes. Étant donnée une formule CNF, il est fréquent de se demander si cette formule possède une solution ou non.

Ce problème est connu sous le nom de *problème de cohérence propositionnelle*, plus communément nommé *problème SAT*. Il s'agit du premier problème à avoir été démontré comme NP-complet par Stephen Cook en 1971 [Coo71]. Ce problème possède de nombreuses applications en intelligence artificielle et en informatique, par exemple dans le cadre de la vérification formelle et de la planification [Bie09, Rin09, Kro09, Zha09]. Les dernières décennies ont vu le développement d'importantes améliorations dans la résolution du problème SAT, de sorte que les solveurs SAT dits « modernes » sont capables de résoudre efficacement de nombreuses instances qui étaient complètement hors d'atteinte trente ans plus tôt [JLRS12]. Cette efficacité en pratique des solveurs SAT peut être expliquée par le développement de l'architecture CDCL (*Conflict-Driven Clause Learning*, apprentissage de clauses guidé par les conflits) [MS99] et par celui de structures de données efficaces et d'heuristiques performantes [MMZ$^+$01, ES04]. Cependant, certaines instances restent difficiles à résoudre pour les solveurs SAT actuels, en particulier celles nécessitant de « savoir compter », comme par exemple les formules dites du *principe du pigeonnier*, encodant le fait qu'il n'est pas possible de placer n pigeons dans $n - 1$ boulins [Hak85].[1]

Partant de ce constat, différentes solutions ont été étudiées. L'une d'entre-elles consiste à généraliser le format CNF en autorisant l'utilisation de contraintes pseudo-booléennes, c'est-à-dire, des équations ou inéquations linéaires en variables booléennes. Plus précisément, nous considérons le langage PBC composé de conjonctions de contraintes de la forme $\sum_{i=1}^{n} \alpha_i \ell_i \geq \delta$, où les ℓ_i sont des littéraux, les α_i sont des coefficients et δ est le degré. Le degré et les coefficients sont tous des entiers naturels. Nous considérons également le langage CARD des contraintes de cardinalité, ayant la forme $\sum_{i=1}^{n} \ell_i \geq \delta$. Ce sont des contraintes pseudo-booléennes dont tous les coefficients sont égaux à 1. Ces représentations présentent plusieurs avantages par rapport au format CNF, qu'elles généralisent (nous noterons qu'une clause est une contrainte de cardinalité de degré 1). Tout d'abord, il est bien connu que les contraintes pseudo-booléennes sont plus concises que les clauses [DGP04] : une seule contrainte pseudo-booléenne peut représenter un nombre exponentiel de clauses. Ces contraintes permettent également l'utilisation du système de preuve des *plans-coupes* [Gom58], qui est en théorie plus puissant que celui de la *résolution* traditionnellement utilisé pour raisonner avec des clauses. Formellement, le système de preuve des plans-coupes *p-simule* la résolution [CCT87], c'est-à-dire que toute preuve par résolution peut être simulée

[1]Ce principe est plus communément appelé *principe des tiroirs* en français.

dans le système des plans-coupes par une preuve de taille polynomiale par rapport à celle de la preuve originale. Ce résultat est à l'origine du développement des solveurs pseudo-booléens [RM09a], qui pour la plupart utilisent un sous-ensemble des règles du système des plans-coupes pouvant être vu comme une généralisation de la résolution [Hoo88]. Ce sous-ensemble se compose des deux règles données ci-dessous.

$$\frac{\sum_{i=1}^{n} \alpha_i \ell_i \geq \delta}{\sum_{i=1}^{n} \min(\alpha_i, \delta) \ell_i \geq \delta} \text{ (saturation)}$$

$$\frac{\alpha\ell + \sum_{i=1}^{n} \alpha_i \ell_i \geq \delta \qquad \beta\bar{\ell} + \sum_{i=1}^{n'} \beta_i \ell'_i \geq \delta' \qquad \rho, \rho' \in \mathbb{N}^* \qquad \rho\alpha = \rho'\beta}{\sum_{i=1}^{n} \rho\alpha_i \ell_i + \sum_{i=1}^{n'} \rho'\beta_i \ell'_i \geq \rho\delta + \rho'\delta' - \rho\alpha} \text{ (annulation)}$$

Grâce à ces règles, il est possible d'étendre l'inférence clausale à l'inférence pseudo-booléenne, en héritant des nombreuses techniques utilisées pour la résolution du problème SAT, notamment via l'analyse de conflit implantée dans de nombreux solveurs pseudo-booléens [DG02, CK05, SS06, LP10, EN18]. En particulier, dans ces solveurs, chaque fois qu'un *conflit* (c'est-à-dire, une contrainte falsifiée) est rencontré, la règle d'annulation est appliquée successivement entre la contrainte conflictuelle et la raison de la falsification de l'un de ses littéraux, ce qui conduit à inférer une nouvelle contrainte conflictuelle. Lorsque la contrainte produite est *assertive* (c'est-à-dire, qu'elle propage l'un de ses littéraux à un certain niveau de décision), cette contrainte est *apprise* et un retour-arrière non-chronologique est effectué au niveau de décision où le littéral est propagé, avant de reprendre la recherche.

Malheureusement, la plupart des solveurs pseudo-booléens actuels ne parviennent pas à capturer la totalité de la puissance du système des plans-coupes [VEG$^+$18]. Ceci est en partie dû à la difficulté d'implanter efficacement les règles du système des plans-coupes, et à celle de choisir quelles règles il convient d'appliquer. De plus, pour s'assurer que les contraintes produites pendant l'analyse de conflit restent falsifiées, il est nécessaire d'utiliser la règle d'affaiblissement donnée ci-dessous, et donc d'affaiblir les contraintes utilisées lors du raisonnement.

$$\frac{\alpha\ell + \sum_{i=1}^{n} \alpha_i \ell_i \geq \delta \qquad \alpha \in \mathbb{N}}{\sum_{i=1}^{n} \alpha_i \ell_i \geq \delta - \alpha} \text{ (affaiblissement)}$$

En pratique, les solveurs fondés sur le système de preuve de la résolution sont donc souvent plus efficaces, de sorte qu'il peut être préférable d'encoder les formules pseudo-booléennes données au solveur sous la forme d'une formule CNF *équisatisfiable*. Cependant, lorsque des garanties de temps de réponse sont attendues (par exemple, dans le cadre d'une application impliquant des interactions avec l'utilisateur), cette approche est insuffisante pour garantir l'efficacité du raisonnement. Dans ce cas, il peut être plus intéressant d'utiliser un autre langage qui permet de réaliser efficacement les opérations souhaitées. Cette traduction dans un autre langage est connue sous le nom de *compilation de connaissances* [GKPS95, CD97].

Dans cette optique, nous considérons dans un premier temps les contraintes pseudo-booléennes comme un langage de représentation [LMMW18]. En particulier, nous montrons que ce langage n'est pas adapté à la compilation de connaissances, comme le test de cohérence d'une formule pseudo-booléenne est NP-complet. Plus précisément, si l'on considère les critères de la carte de compilation [DM02], les contraintes pseudo-booléennes n'offrent pas de requêtes supplémentaires comparé au langage CNF, tandis que des transformations offertes par ce dernier langage ne sont plus traitables lorsque l'on considère

	CO	VA	CE	IM	EQ	SE	CT	ME
CNF	○	✓	○	✓	○	○	○	○
CARD	○	✓	○	✓	○	○	○	○
PBC	○	✓	○	✓	○	○	○	○

Table 1: Propriétés de CARD et PBC en termes de requêtes, comparées à celles de CNF. Un ✓ indique que la requête peut être effectuée en temps polynomial, tandis qu'un ○ signifie que ce n'est pas le cas, sauf si $\mathsf{P} = \mathsf{NP}$.

	CD	FO	SFO	$\wedge$C	$\wedge$BC	$\vee$C	$\vee$BC	$\neg$C
CNF	✓	○	✓	✓	✓	●	✓	●
CARD	✓	●	●	✓	✓	●	●	●
PBC	✓	●	●	✓	✓	●	●	●

Table 2: Propriétés de CARD et PBC en termes de transformations, comparées à celles de CNF. Un ✓ indique que la transformation peut être effectuée en temps polynomial, tandis qu'un ○ signifie que ce n'est pas le cas, sauf si $\mathsf{P} = \mathsf{NP}$ et un ● que la transformation ne peut pas êre calculée en temps polynomial, de manière inconditionnelle.

des contraintes pseudo-booléennes en plus des clauses (c'est le cas, par exemple, de l'oubli d'une variable ou de la clôture par disjonction bornée). Les Tableaux 1 et 2 résument l'ensemble de nos résultats sur les langages CARD et PBC, où ils sont comparés au langage CNF.

Le principal avantage des contraintes pseudo-booléennes, du point de vue de la représentation des connaissances, est donc leur concision: une seule contrainte pseudo-booléenne peut représenter un nombre exponentiel de clauses. Le diagramme à la Figure 1 permet de comparer la concision (ou efficacité spatiale) de divers langages propositionnels aux langages CARD et PBC.

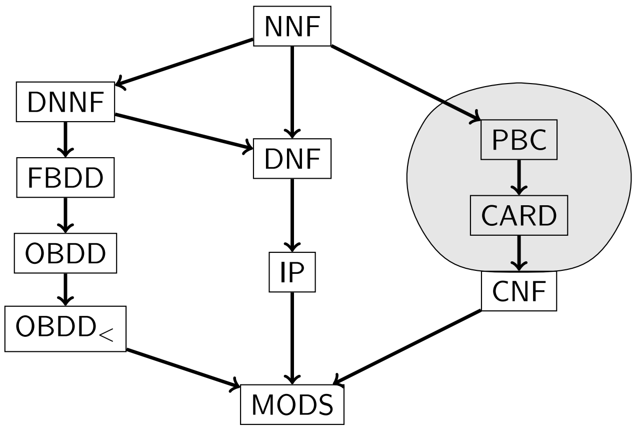

$$\begin{tikzcd}
 & \mathsf{NNF} \arrow[dl] \arrow[d] \arrow[dr] & \\
\mathsf{DNNF} \arrow[r] \arrow[d] & \mathsf{DNF} \arrow[d] & \mathsf{PBC} \arrow[d] \\
\mathsf{FBDD} \arrow[d] & \mathsf{IP} \arrow[dd] & \mathsf{CARD} \arrow[d] \\
\mathsf{OBDD} \arrow[d] & & \mathsf{CNF} \arrow[dl] \\
\mathsf{OBDD}_< \arrow[r] & \mathsf{MODS} &
\end{tikzcd}$$

Figure 1: Concision de différents langages propositionnels. Dans ce diagramme, une flèche $\mathsf{L}_1 \rightarrow \mathsf{L}_2$ indique que L_1 est strictement plus concis que L_2. L'absence de flèche entre deux langages indique que ceux-ci sont incomparables. La zone grise met en évidence nos contributions (y compris les résultats d'incomparabilité).

Comme le critère de concision ne considère que des formules équivalentes, il ne tient pas compte des encodages autorisant l'ajout de variables auxiliaires. Il est bien connu que l'utilisation de telles variables peut permettre de réduire fortement la taille des formules encodées. Pour étudier ces encodages, il est fréquent d'associer des graphes aux formules CNF qu'ils produisent, notamment le graphe primal (dont les nœuds correspondent aux variables et les arêtes représentent le fait que deux variables apparaissent conjointement dans une même clause) ou le graphe d'incidence (dont les nœuds sont des variables ou des clauses et les arêtes relient chaque variable aux clauses où elle apparaît). La largeur de ces graphes, évaluée via différentes mesures telles que la largeur d'arbre ou la largeur de clique, donne souvent des informations relatives à la difficulté du problème considéré : lorsque la largeur est petite, il existe des algorithmes efficaces en pratique permettant de résoudre SAT, mais aussi des problèmes plus complexes tels que #SAT, MAX-SAT ou même QBF [SS10a, FMR08, SS13, PSS16, STV15, Che04]. Dans ce contexte, nous étudions les formules pour lesquelles il existe des encodages de faible largeur. En particulier, nous montrons que, si la largeur des encodages est bornée, alors l'expressivité de ces encodages devient limitée, de sorte que ces encodages ne peuvent plus être utilisés que pour représenter des formules ayant une faible complexité de communication [MW19, MW20].

D'un point de vue pratique, nous étudions différentes approches pour améliorer les performances des solveurs pseudo-booléens. En particulier, nous étudions l'impact de la présence de littéraux dits *non pertinents* dans les contraintes apprises par le solveur [LMMW20]. Ces littéraux se caractérisent par le fait que leur valeur de vérité n'a pas d'impact sur celle de la contrainte dans laquelle ils apparaissent : ils peuvent alors être supprimés de cette contrainte tout en préservant l'équivalence logique.

Exemple 1

Dans la contrainte $10a + 5b + 5c + 2d + e + f \geq 15$, les trois littéraux d, e et f ne sont pas pertinents. Cette contrainte est donc équivalente (entre autres) aux deux contraintes $10a + 5b + 5c \geq 15$ et $10a + 5b + 5c \geq 11$.

Nous montrons que de tels littéraux peuvent être introduits par l'application de règles du système des plans-coupes. Lorsque ces littéraux deviennent *artificiellement pertinents*, ils peuvent conduire à l'inférence de contraintes plus faibles que ce qu'elles pourraient être si les littéraux non pertinents n'étaient pas présents, comme illustré dans l'exemple ci-dessous.

Exemple 2

Soit la contrainte $\chi \equiv 6a + 6b + 4c + 3d + 3e + 2f \geq 10$. Si la règle d'affaiblissement est appliquée sur cette contrainte pour éliminer c, nous obtenons la contrainte $\chi' \equiv 6a + 6b + 3d + 3e + 2f \geq 6$, dans laquelle f n'est plus pertinent.
Supposons maintenant que la règle d'annulation soit appliquée entre χ' et $4a + 4b + 3\bar{e} + 3g + 3h + 2i + 2j \geq 16$ pour éliminer la variable e. Nous obtenons alors la contrainte $10a + 10b + 3d + 3g + 3h + 2f + 2i + 2j \geq 19$, dans laquelle f est devenu artificiellement pertinent.
Maintenant, supposons que f ait été retiré de χ' par affaiblissement, donnant après saturation la contrainte $\chi'' \equiv 4a + 4b + 3d + 3e \geq 4$. Si cette contrainte est maintenant utilisée à la place de χ' dans l'opération d'annulation ci-dessus, nous obtenons la contrainte $8a + 8b + 3d + 3g + 3h + 2i + 2j \geq 17$, qui est strictement plus forte que la contrainte obtenue précédemment.

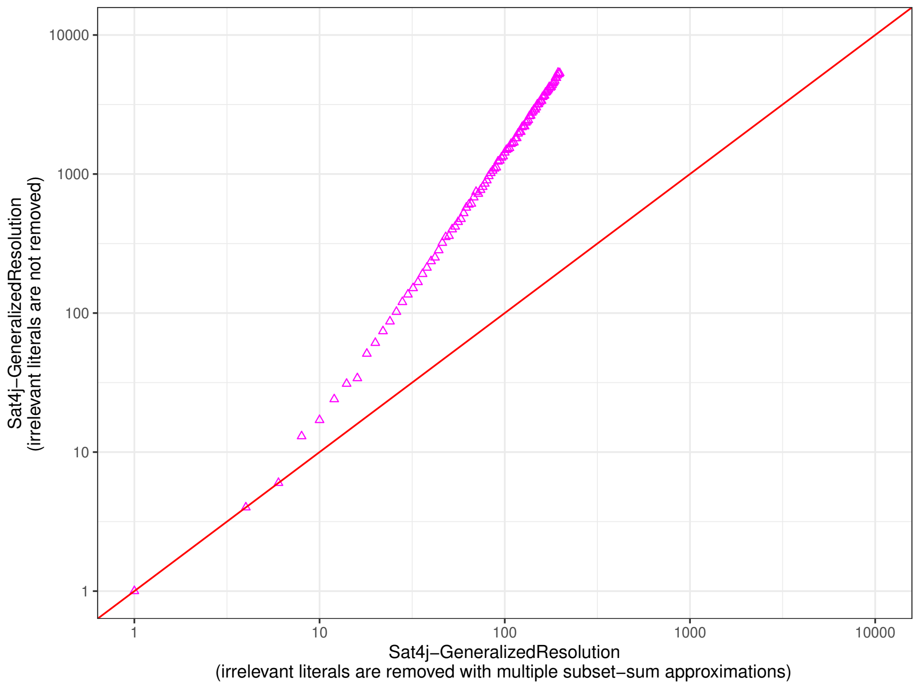


Figure 2: Comparaison de la taille de la preuve produite par *Sat4j* sur des instances de la famille `vertexcover-completegraph` avec et sans retrait des littéraux non pertinents (échelle logarithmique).

En conséquence, les contraintes apprises par le solveur en présence de littéraux non pertinents peuvent être plus faibles que lorsque ces littéraux ne sont pas supprimés. Comme nous le montrons empiriquement, cela peut conduire le solveur à produire une preuve d'incohérence exponentiellement plus longue, comme illustré à la Figure 2.

Cependant, le traitement des littéraux non pertinents reste difficile en pratique : leur détection est NP-difficile. Une solution possible pour supprimer efficacement ces littéraux est de tirer parti de la règle d'affaiblissement, afin d'affaiblir les littéraux qui n'ont pas d'effet sur le conflit en cours d'analyse, peu importe leur affectation courante (même si ces littéraux peuvent être pertinents). Nous étudions différentes stratégies d'affaiblissement et montrons que, malgré le fait qu'aucune des stratégies étudiées n'est meilleure que les autres sur l'ensemble des instances considérées, la manière d'appliquer cette règle peut avoir un impact important sur les performances du solveur, comme le montre la Figure 3 [LMW20]. Il est toutefois intéressant d'observer que, alors que la plupart des implantations des solveurs pseudo-booléens appliquent la règle d'affaiblissement sur la *raison* lors de l'analyse de conflit (*RoundingSat* [EN18] est le premier solveur à l'appliquer à la fois sur la raison *et* le conflit), il peut en fait être préférable de l'appliquer sur le conflit pour obtenir de meilleures performances.

Enfin, nous présentons différentes approches pour adapter à la résolution de problèmes pseudo-booléens les divers composants de l'algorithme CDCL, de façon à en tirer le meilleur parti possible. En effet, il est bien connu que de nombreuses fonctionnalités implantées dans les solveurs SAT utilisant la résolution sont nécessaires pour obtenir le meilleur de ces solveurs. Plus précisément, concernant l'heuristique de choix de variables, les solveurs pseudo-booléens utilisent généralement VSIDS [MMZ$^+$01], et notamment sa variante EVSIDS [ES04]. Cependant, cette heuristique ne tient pas compte de la forme particulière des contraintes pseudo-booléennes, avec leurs coefficients et leurs propriétés. C'est également le cas des stratégies de suppression des contraintes apprises, qui permettent

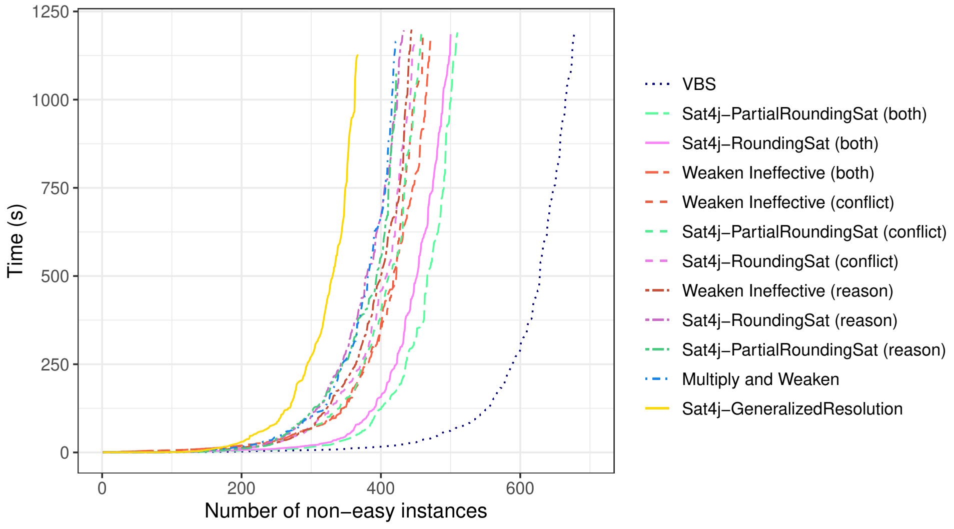


Figure 3: Cactus plot comparant les résultats des différentes stratégies d'affaiblissement implantées dans *Sat4j*.

de limiter leur nombre pour éviter de ralentir la propagation unitaire, ainsi que des politiques de redémarrage, qui permettent au solveur de ne pas rester « bloqué » dans une partie de l'espace de recherche. Dans ces deux derniers cas, il est fréquent de considérer la *qualité* des contraintes apprises. Pour évaluer au mieux cette qualité, il semble également important de considérer les particularités des contraintes pseudo-booléennes.

Nous proposons donc différentes variantes de ces stratégies, en prenant notamment en compte l'affectation courante et la taille des coefficients apparaissant dans les contraintes. Toutes ces stratégies ont été implantées dans le solveur pseudo-booléen *Sat4j* [LP10] et sont disponibles sur son dépôt.[2] La Figure 4 permet d'observer les améliorations apportées par l'utilisation des différentes stratégies sus-mentionnées. En particulier, il est intéressant de noter que leur utilisation permet aux différentes variantes de *Sat4j* d'être compétitives par rapport à différents solveurs de l'état de l'art (y compris ceux fondés sur la résolution).

En étudiant les sujets ci-dessus, de nombreuses perspectives ont été identifiées. Parmi celles-ci, nous pouvons citer l'amélioration des différentes nouvelles approches proposées dans cette thèse, notamment en exploitant leur complémentarité à l'aide, par exemple, d'algorithmes de configuration automatique permettant de déterminer dynamiquement la meilleure stratégie à adopter. D'autres pistes d'amélioration restent également à explorer, comme la détermination du niveau du retour-arrière optimal lors de l'analyse de conflit. En effet, contrairement à ce qui est observé dans les solveurs SAT classiques, il s'avère qu'apprendre la première contrainte pseudo-booléenne assertive à l'issue de l'analyse de conflit ne garantit pas de remonter au plus haut niveau possible dans l'arbre de décision. Une autre différence importante entre les solveurs pseudo-booléens et les solveurs SAT classiques est que les raisons rencontrées par les premiers au cours de l'analyse de conflit peuvent être conflictuelles. Il serait intéressant d'étudier plus en profondeur si la solution actuelle – qui ignore totalement ce constat – ne pourrait pas être améliorée, par exemple en remplaçant la contrainte conflictuelle courante par la raison falsifiée rencontrée. Enfin, une dernière perspective est de tirer parti des améliorations des solveurs pseudo-booléens proposées dans cette thèse pour réaliser des tâches plus complexes, telles que la résolution de problèmes d'optimisation ou la compilation de formules pseudo-booléennes.

[2] https://gitlab.ow2.org/sat4j/sat4j

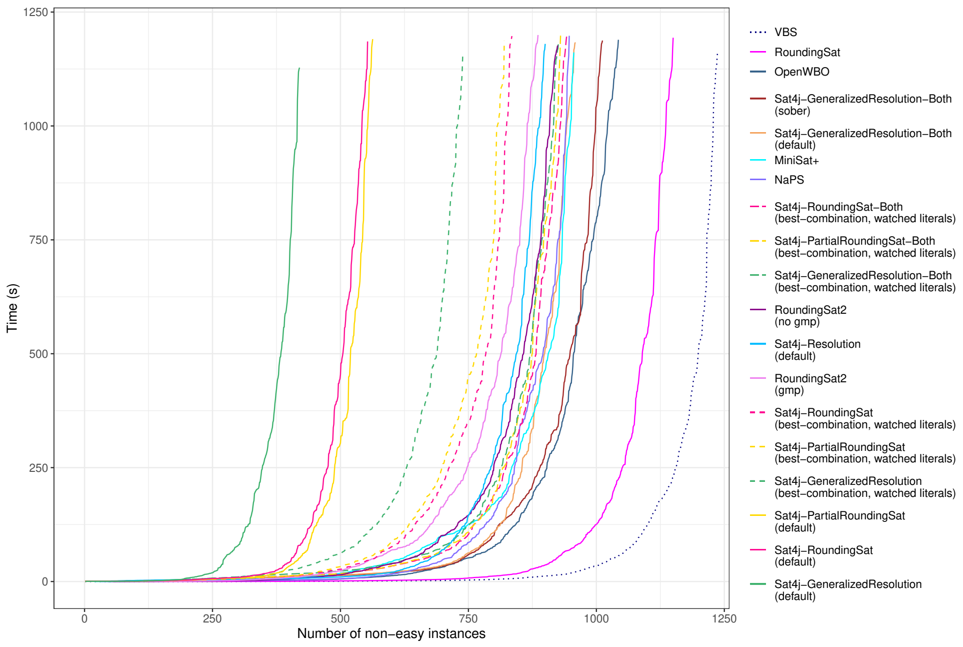


Figure 4: Cactus plot comparant différentes configurations de *Sat4j* (utilisant des *littéraux surveillés*) avec différents solveurs de l'état de l'art.

Contents

Appendices

General Introduction

Representing pieces of propositional information is an important task in artificial intelligence, and many languages have been designed in this respect. Such languages correspond to various tradeoffs between their expressivity (i.e., what you can represent with these languages) and the efficiency of the reasoning you can perform on pieces of information represented using them. Among the numerous existing languages, the language of formulae in *Conjunctive Normal Form* (CNF) is widely used, as it provides a convenient and simple way to represent constraints over Boolean variables. Given a CNF formula, a common question is to check whether this formula admits a solution or not.

This problem is known as the *satisfiability problem*, often abbreviated as *SAT*. It is the first problem proven to be NP-complete by Stephen Cook in 1971 [Coo71]. This problem has a lot of applications in artificial intelligence and in computer science, for instance in formal verification and planning [Bie09, Rin09, Kro09, Zha09]. Last decades have seen many improvements in SAT solving, and so-called "modern" SAT solvers are able to solve efficiently many instances that were completely out of reach thirty years ago [JLRS12]. This practical efficiency of SAT solvers can be explained by the development of the *conflict-driven clause learning architecture* (CDCL) [MS99] and of efficient data structures and heuristics [MMZ+01, ES04]. However, some instances remain hard to solve for current SAT solvers, especially the ones requiring "counting capabilities", as for instance so-called *pigeonhole principle* formulae, stating that it is not possible to put n pigeons in $n - 1$ holes [Hak85].

Starting from this observation, different solutions have been investigated. One of them consists in using a generalization of the CNF format based on pseudo-Boolean constraints, i.e., linear equations or inequations over Boolean variables. This representation has several benefits compared to CNF. First, it is well-known that pseudo-Boolean constraints are more succinct than clauses: a single pseudo-Boolean constraint can represent an exponential number of clauses [DGP04]. They also allow to use the *cutting planes* proof system [Gom58], that is in theory stronger than the *resolution* proof system traditionally used to reason with clauses. Formally, the cutting planes proof system *p-simulates* resolution [CCT87]: any resolution proof can be simulated by a cutting planes proof of polynomial size with respect to the size of the original proof. This has motivated the development of pseudo-Boolean solvers [RM09a], often based on a subset of the cutting planes proof system which can be viewed as a generalization of resolution [Hoo88]. This allows to extend clausal inference to pseudo-Boolean inference, inheriting many of the techniques used in SAT solving [DG02, CK05]. In particular, the conflict-driven clause learning architecture has been extended to pseudo-Boolean solving, and many solvers have been developed in this direction [DG02, CK05, SS06, LP10, EN18].

Disappointingly, most current pseudo-Boolean solvers fail to capture the whole strength of the cutting planes proof system [VEG+18]. In practice, solvers based on the resolution proof system are often more efficient, so that it may be preferable to encode the pseudo-Boolean input as an *equisatisfiable* CNF formula. However, when runtime guarantees must be provided (for instance, for an application involving interactions with users), this approach is not enough to ensure the efficiency of the reasoning. In this case, it may be more interesting to use another language that allows to perform the needed operations efficiently. This translation to another language is known as *knowledge compilation* [GKPS95, CD97].

In the first part of this thesis, we consider the compilation approach, which aims at performing the "hard" operations once and for all in an *offline* process, so that operations to be performed *online* can be executed efficiently. In a first contribution [LMMW18], we consider the criteria of the knowledge compilation map [DM02] to locate the languages of pseudo-Boolean constraints in this map. We show that the main gain offered by these languages is their succinctness compared to CNF, which can actually be limited when considering CNF encodings instead of CNF representations. Indeed, it is well-known that using auxiliary variables in such encodings allows to reduce the size of the produced formula [Tse68, PG86]. Contrastingly, our second contribution [MW19, MW20] shows that, if we bound the width of such encodings, we may significantly limit their expressivity. We also show that, in a sense, formulae of bounded width can only encode simple functions.

In the second part, we consider the resolution of the satisfiability problem for pseudo-Boolean formulae through the use of different subsets of the cutting planes proof system. As a first contribution in this direction [LMMW20], we highlight an intriguing behavior of the rules forming the cutting planes proof system: the production of *irrelevant* literals. Such literals have no impact on the truth value of the constraints in which they appear. However, we show that they harm the solver deductive power by leading to the production of weaker constraints when they appear than when they do not. In a second contribution [LMW20], we show that one can efficiently get rid of irrelevant literals by the application of the weakening rule, at the price of also weakening away relevant literals. Yet, we empirically demonstrate that carefully choosing *how* to apply the weakening rule (which is required by pseudo-Boolean solvers) may also improve their runtime. In our third contribution, we show that this runtime may also be improved by the adaption of many of the strategies implemented in CDCL SAT solvers to the pseudo-Boolean case, especially by taking into account the current assignment and the coefficients appearing in the constraints.

Part I

Pseudo-Boolean Constraints and Knowledge Compilation

Introduction

Representing and reasoning from pieces of propositional information is of major importance in computer science, and specifically in artificial intelligence (AI). Accordingly, many languages and data structures have been pointed out so far for this purpose. In this context, an important issue is to choose *which* language to use, considering the information to be represented and which operations must be supported [GKPS95]. This is not a trivial issue since there exist many different languages for representing propositional information, and none of them is the best one in an absolute way. Tradeoffs must be looked for.

The choice of the language to be used actually depends on the operations that are expected in the context of the application being considered. To help in making this choice, a knowledge compilation map has been introduced in [DM02]. It suggests to make the decision on several criteria, based on both the spatial and temporal efficiency of the languages.

A particularly interesting language for representing propositional information is that of formulae in *Conjunctive Normal Form* (CNF). Indeed, one often needs to represent a set of laws or constraints to be interpreted conjunctively, for which clauses are particularly adapted, while being simple objects. This explains why so many benchmarks have been encoded as CNF formulae in different applications of AI (often written using the DIMACS format [DIM93]). This also explains the development of a rich ecosystem around these formulae [SAT99, BHvMW09].

However, the simplicity of clauses is also a drawback: they do not allow to represent efficiently some important constraints. For instance, stating that m propositional variables among n must be set to true requires an exponential number of clauses in general [DGP04], unless one adds new variables. Depending on the application, adding new variables may be permitted, and one can rely on CNF *encodings* that take advantage of so-called *auxiliary variables* to make the CNF formula smaller in practice than the CNF *representations* that only uses the original variables. However, adding such variables may be undesirable in some circumstances, especially as doing so does not preserve logical equivalence. In this case, one needs to use a language that is better adapted to such constraints, as that of *pseudo-Boolean constraints*.

Pseudo-Boolean constraints are generalizations of clauses that allow to consider linear equations or inequations over Boolean variables, and are strongly related to so-called *threshold functions* [CLH11, Chapter 9]. Using pseudo-Boolean formulae in place of CNF formulae has several advantages. Indeed, in addition to be more space efficient than clauses [DGP04], pseudo-Boolean constraints also provide a more natural way to represent a wide variety of constraints (for instance, the *subset-sum* or *knapsack* problems can basically be encoded using a single pseudo-Boolean constraint). As they generalize clauses, it is also possible to efficiently represent any CNF formula as a pseudo-Boolean formula. One can also adapt the tools developed for CNF formulae to deal with pseudo-Boolean constraints [MS97, WS01, CK05, SS06, Dix04, LP10, EN18].

In this first part of the document, we present some theoretic properties of pseudo-Boolean formulae and their encodings. In Chapter 1, we introduce different frameworks for representing information, based on propositional logic and pseudo-Boolean constraints. Based on complexity considerations, we also study how to choose a representation language that fits one's needs based on the criteria at work in the knowledge compilation map [DM02].

Chapter 2 studies pseudo-Boolean constraints from a knowledge representation perspective. We first investigate some intrinsic properties of pseudo-Boolean constraints, and exhibit several important differences between such constraints and clauses. We then consider pseudo-Boolean and cardinality constraints as representation languages for propositional information, and compare these languages to many well-known propositional languages, using the criteria of the knowledge compilation map. This work has been published at the International Joint Conference in Artificial Intelligence (IJCAI) in 2018 [LMMW18].

In Chapter 3, we consider bounded width CNF formulae where the width is measured by different graph width measures associated with CNF formulae. We focus on the expressiveness of these formulae in the model of CNF encodings with auxiliary variables. On the one hand, we show that bounding the width leads to a dramatic loss of expressiveness, restricting the formulae to those of low communication complexity. On the other hand, we show that there are two classes of width measures, one containing primal treewidth and the other incidence cliquewidth, such that in each class the width of optimal encodings only differs by constant factors. Moreover, between the two classes the width differs at most by a factor logarithmic in the number of variables. This work has been published at the International Conference on Theory and Practice of Satisfiability Solving (SAT) in 2019 [MW19] and in the Journal of Artificial Intelligence Research (JAIR) in 2020 [MW20].

Chapter 1

Formal Preliminaries

Pseudo-Boolean constraints, and more generally propositional logic, are simple but powerful languages that may be used to represent pieces of knowledge, and to perform some reasoning on them. However, the simplicity of these frameworks has an important drawback: in practice, some tasks may be computationally hard to perform, and may thus require a lot of time to be completed. For many applications, especially those involving interactions with users, this is not acceptable, as the operations must be executed within a limited amount of time. In this case, *compiling* the representation, i.e., translating it into a different language in which performing the wanted operations is easier, is particularly useful. In this chapter, we introduce the notions used throughout the document regarding these different frameworks.

1.1 Definitions and Notations

Before getting to the heart of the matter, let us start by introducing some definitions and notations for the different objects of the frameworks we study.

1.1.1 Propositional Logic

Let us first consider *propositional logic*, for which we give some notions to describe its syntax and its semantics.

Definition 1 (Propositional Formula)

The set of *propositional formulae* $PROP_{\mathcal{PS}}$, written on the set $\mathcal{PS}$ of *propositional variables* (or simply, *variables*), is recursively defined as the smallest set (for inclusion) satisfying the following statements:

- for all $v \in \mathcal{PS} \cup \{\bot, \top\}$, $v \in PROP_{\mathcal{PS}}$,
- if $\varphi, \psi \in PROP_{\mathcal{PS}}$, then $\neg\varphi, (\varphi \wedge \psi), (\varphi \vee \psi) \in PROP_{\mathcal{PS}}$.

Remark 1

For more readability, parentheses required in the second point of the definition above are often left out when the formula is unambiguous without these parentheses.

When considering a propositional formula, we generally focus on the variables appearing in this formula, which are denoted as follows.

Notation 1

Given a propositional formula φ, the set of propositional variables appearing in φ is denoted $\mathsf{var}(\varphi)$.

Example 1

Let φ be $a \lor \neg(\neg(b \lor c) \lor d)$. φ is a propositional formula, with $\mathsf{var}(\varphi) = \{a, b, c, d\}$.

We also often need to evaluate the *space* required to represent a formula. To this end, we define the notion of *size* as follows.

Definition 2 (Size of a Propositional Formula)

The *size* of a formula φ, denoted $|\varphi|$, is the number of occurrences of the different symbols (variables and connectives) used to write the formula.

Remark 2

As parentheses are only used to clarify the expression of the formula, they are not counted in the size of the formula.

Example 2

Let us consider the formula φ in Example 1 above. We have $|\varphi| = 9$.

Having introduced the syntax of propositional logic, let us now introduce the semantics of this language. Formulae as defined above are generally used to represent *Boolean functions*.

Definition 3 (Boolean Function)

A *Boolean function* is a total function $f : \mathbb{B}^n \to \mathbb{B}$ mapping a set of propositional variables $x_1, \dots, x_n$ to a Boolean value from $\mathbb{B} = \{0, 1\}$.
For a given $X = (x_1, \dots, x_n)$, if $f(X) = 0$, we say that f *rejects* X while, if $f(X) = 1$, we say that f *accepts* X.

The representation of such Boolean functions with propositional formulae is achieved thanks to the notion of *interpretation*.

Definition 4 (Interpretation)

Let $\mathcal{V} \subseteq \mathcal{PS}$. An *interpretation* (or *assignment*) I over $\mathcal{V}$ is a total function mapping each variable $v \in \mathcal{V}$ to a Boolean value in $\mathbb{B} = \{0, 1\}$.
Given a propositional formula φ, an interpretation I is said to be *partial* with respect to φ when $\mathcal{V} \subset \mathsf{var}(\varphi)$, and *complete* when $\mathsf{var}(\varphi) \subseteq \mathcal{V}$.

Definition 5 (Extension of a Partial Interpretation)

Let φ be a propositional formula and I a partial interpretation of φ over a set of propositional variables $\mathcal{V} \subseteq \mathsf{var}(\varphi)$. An interpretation I' is an *extension* of I if and only if I' is defined over a set $\mathcal{V}'$ such that $\mathcal{V} \subseteq \mathcal{V}'$ and, for all $v \in \mathcal{V}$, we have $I'(v) = I(v)$.

From now on, and unless otherwise specified, any interpretation is assumed to be *complete*.

Definition 6 (Semantics)

Let $\varphi, \psi \in PROP_{PS}$. The *value* of a propositional formula under an interpretation I is recursively defined as:

- $I(\bot) = 0$ and $I(\top) = 1$,
- if φ is a propositional variable v, then $I(\varphi) = I(v)$,
- *negation*: $I(\neg\varphi) = 1 - I(\varphi)$,
- *conjunction*: $I(\varphi \land \psi) = \min(I(\varphi), I(\psi))$, and
- *disjunction*: $I(\varphi \lor \psi) = \max(I(\varphi), I(\psi))$.

Definition 7 (Model, Counter-Model)

A *model* (resp. *counter-model*) of propositional formula φ is a complete interpretation I such that $I(\varphi) = 1$ (resp. $I(\varphi) = 0$).

Example 3

Let us consider again the formula φ given by $a \lor \neg(\neg(b \lor c) \lor d)$. The interpretation I given by $I(a) = 1$ and $I(b) = I(c) = I(d) = 0$ is a model of φ. On the contrary, the interpretation I' with $I'(a) = I'(b) = I'(c) = 0$ and $I'(d) = 1$ is a counter-model of φ.

Definition 8 (Logical Entailment)

Let φ and ψ be two propositional formulae. We say that φ *logically entails* (or simply *entails*) ψ, denoted $\varphi \models \psi$, if and only if every model of φ is a model of ψ.

Remark 3

With the definition of logical entailment above, we have $\varphi \models \psi$ if and only if $\varphi \wedge \neg\psi \models \bot$.

Remark 4

The relation $\models$ is transitive: for any formulae φ, ψ and ξ, if $\varphi \models \psi$ and $\psi \models \xi$, then $\varphi \models \xi$.

Example 4

Let us consider again the formula φ given by $a \vee \neg(\neg(b \vee c) \vee d)$. Let ψ be the formula $a \vee b \vee \neg d$. We have that $\varphi \models \psi$ as each time φ is true, ψ is also true:

a	b	c	d	$(b \vee c)$	$\neg(b \vee c)$	$\neg(b \vee c) \vee d$	$\neg(\neg(b \vee c) \vee d)$	φ	ψ
0	0	0	0	0	1	1	0	0	1
0	0	0	1	0	1	1	0	0	0
0	0	1	0	1	0	0	1	1	1
0	0	1	1	1	0	1	0	0	0
0	1	0	0	1	0	0	1	1	1
0	1	0	1	1	0	1	0	0	1
0	1	1	0	1	0	0	1	1	1
0	1	1	1	1	0	1	0	0	1
1	0	0	0	0	1	1	0	1	1
1	0	0	1	0	1	1	0	1	1
1	0	1	0	1	0	0	1	1	1
1	0	1	1	1	0	1	0	1	1
1	1	0	0	1	0	0	1	1	1
1	1	0	1	1	0	1	0	1	1
1	1	1	0	1	0	0	1	1	1
1	1	1	1	1	0	1	0	1	1

Definition 9 (Logical Equivalence)

Two propositional formulae φ and ψ are *logically equivalent* (or simply *equivalent*), denoted $\varphi \equiv \psi$, if they have the same models.

Theorem 1 (Folklore)

Let φ, ψ and ξ be propositional formulae. Then, we have

- *commutativity*: $\varphi \lor \psi \equiv \psi \lor \varphi$ and $\varphi \land \psi \equiv \psi \land \varphi$
- *associativity*: $(\varphi \lor \psi) \lor \xi \equiv \varphi \lor (\psi \lor \xi)$ and $(\varphi \land \psi) \land \xi \equiv \varphi \land (\psi \land \xi)$
- *idempotency*: $\varphi \lor \varphi \equiv \varphi$ and $\varphi \land \varphi \equiv \varphi$
- *distributivity*: $\varphi \lor (\psi \land \xi) \equiv (\varphi \lor \psi) \land (\varphi \lor \xi)$ and $\varphi \land (\psi \lor \xi) \equiv (\varphi \land \psi) \lor (\varphi \land \xi)$
- *involution*: $\neg\neg\varphi \equiv \varphi$
- *De Morgan's laws*: $\neg(\varphi \lor \psi) \equiv \neg\varphi \land \neg\psi$ and $\neg(\varphi \land \psi) \equiv \neg\varphi \lor \neg\psi$

Example 5

Let us again consider the formula φ given by $a \lor \neg(\neg(b \lor c) \lor d)$. We have:

$$\begin{aligned}\varphi &\equiv a \lor (\neg\neg(b \lor c) \land \neg d)\\ &\equiv a \lor ((b \lor c) \land \neg d)\\ &\equiv (a \lor b \lor c) \land (a \lor \neg d)\end{aligned}$$

With the notion of logical equivalence above, we can now introduce *normal forms*. Such normal forms allow to represent any propositional formula under a specific form that is equivalent to the original formula. This is for example the case of the *Conjunctive Normal Form*.

Definition 10 (Literal)

A *literal* ℓ is a propositional variable v or its negation $\neg v$.

The negation of a literal ℓ, denoted $\neg\ell$ is the opposite literal of ℓ. If v is a propositional variable and $\ell = v$, then $\ell = \neg v$. If $\ell = \neg v$, then $\neg\ell = \neg\neg v$. As $\neg$ is an involutive connective (see Theorem 1), we simply write $\neg\ell = v$ for the purposes of notation.

Notation 2

Given a literal ℓ, we denote by $\mathsf{var}(\ell)$ the variable v such that either $\ell = v$ or $\ell = \neg v$.

Notation 3

For more readability, we also use the notation $\bar{v}$ to represent the negation of variable v (i.e., $\neg v$). Similarly, the notation $\bar{\ell}$ is used to represent the complementary of the literal ℓ (i.e., $\neg\ell$).

Definition 11 (Clause)

A *clause* is a formula having the form of a disjunction of literals, or $\bot$.

Example 6

The disjunctions $a \vee b \vee c$ and $a \vee \neg d$ are clauses.

Notation 4

Given a clause γ, we denote by $\mathsf{lit}(\gamma)$ the set of the literals appearing in γ.

Definition 12 (Conjunctive Normal Form)

A propositional formula is in *Conjunctive Normal Form* (CNF) if it is a conjunction of clauses, or $\top$.

Example 7

The formula $\psi = (a \vee b \vee c) \wedge (a \vee \neg d)$ is a CNF formula that is equivalent to the formula $\varphi = a \vee \neg(\neg(b \vee c) \vee d)$ (see Example 5).

Notation 5

Given a CNF formula φ, we denote by $\mathsf{lit}(\varphi)$ the set of the literals appearing in φ or $\top$.

Remark 5 (Representation)

It is often convenient to represent a clause as a set of literals, and a CNF formula as a set of clauses. In particular, given a literal ℓ, a clause γ and a CNF formula φ, $\ell \in \gamma$ denotes that ℓ is a literal of γ (i.e., $\ell \in \mathsf{lit}(\gamma)$), and $\gamma \in \varphi$ denotes that γ is a clause of φ. This makes sense given that $\wedge$ and $\vee$ are commutative, associative and idempotent connectives (see Theorem 1).

Example 8

The CNF formula $(a \vee b \vee c) \wedge (a \vee \neg d)$ is represented as $\{\{a, b, c\}, \{a, \neg d\}\}$.

Similarly to the CNF defined above, we may also define *Disjunctive Normal Form* (DNF).

Definition 13 (Term, Cube)

A *term* (or *cube*) is a formula having the form of a conjunction of literals, or $\top$.

Example 9

The conjunctions $b \wedge \neg d$ and $c \wedge \neg d$ are terms.

Definition 14 (Disjunctive Normal Form)

A propositional formula is in *Disjunctive Normal Form* (DNF) if it is a disjunction of terms, or $\bot$.

Example 10

The formula $\psi = a \vee (b \wedge \neg d) \vee (c \wedge \neg d)$ is a DNF formula that is equivalent to the formula $\varphi = a \vee \neg(\neg(b \vee c) \vee d)$.

The *satisfiability problem* (or *SAT problem*, for short) consists in checking whether a propositional formula φ is *consistent*.

Definition 15 (Consistency, Contradiction)

A propositional formula φ is *consistent* (or *satisfiable*) if and only if it has at least one model. Otherwise, φ is said to be *contradictory* (or *inconsistent*, *unsatisfiable*).

Example 11

The formula $\varphi = a \vee \neg(\neg(b \vee c) \vee d)$ is consistent, as it admits at least one model (see Example 3). On the contrary, the formulae $a \wedge \neg a$ and $a \wedge (\neg a \vee b) \wedge \neg b$ are inconsistent.

Definition 16 (Empty Clause)

The empty clause is the clause $\bot$. It is equivalent to an empty disjunction, and is thus always contradictory.

Definition 17 (Validity)

A propositional formula φ is *valid* if and only if every interpretation of $\mathsf{var}(\varphi)$ is a model of φ. Equivalently, φ is *valid* if and only if $\neg\varphi$ is contradictory.

Example 12

The clause $a \lor \neg a$ is valid.

Definition 18 (Empty Term)

The empty term is the term $\top$. It is equivalent to an empty conjunction, and is thus always valid.

1.1.2 Pseudo-Boolean Constraints

In the previous section, we only considered propositional formulae that are defined using *logical* connectives. We now consider an extension of this language, that allows the use of elementary *arithmetic* operations, which is made possible by the *numerical* interpretation of the propositional variables, taking their value in $\{0, 1\}$. This language is that of *pseudo-Boolean formulae*, composed of conjunctions of *pseudo-Boolean constraints*.

Definition 19 (Pseudo-Boolean Constraint)

A *pseudo-Boolean constraint* is a constraint of the form $\sum_{i=1}^{n} \alpha_i \ell_i \vartriangle \delta$, in which:

- for all $i \in \{1, \ldots, n\}, \alpha_i \in \mathbb{Z}$,
- for all $i \in \{1, \ldots, n\}, \ell_i$ is a literal,
- $\vartriangle \in \{<, \leq, =, \geq, >\}$, and
- $\delta \in \mathbb{Z}$.

Each α_i is called a *weight* or *coefficient*, and δ is called the *degree (of falsity)* or *threshold* of the constraint.

Notation 6

Given a pseudo-Boolean constraint χ, we denote by $\mathsf{var}(\chi)$ the set of propositional variables appearing in χ and by $\mathsf{lit}(\chi)$ the set of literals appearing in χ.

Let us now generalize the notion of size that we consider for propositional formulae to the case of pseudo-Boolean formulae.

Definition 20 (Size of a Pseudo-Boolean Constraint)

Let χ be the pseudo-Boolean constraint $\sum_{i=1}^{n} \alpha_i \ell_i \geq \delta$. The *size* of χ, denoted $|\chi|$, is given by $|\chi| = \sum_{i=1}^{n}(\lceil \log_2(\alpha_i + 1) \rceil + 1) + \lceil \log_2(\delta + 1) \rceil$.
In other words, the size of χ measures both the number of literals of the constraint and the size of its coefficients and degree.

Example 13

Let us consider the following constraints:

- $\chi_1 : 6\bar{b} + 6c + 4e + f + g + h \geq 7$
- $\chi_2 : 5a + 4b + c + d \geq 6$
- $\chi_3 : a + d + \bar{e} \geq 2$
- $\chi_4 : \bar{b} + c + e \geq 1$

We have that $|\chi_1| = 21$, $|\chi_2| = 16$, $|\chi_3| = 8$ and $|\chi_4| = 7$.

Remark 6

Observe that the constraint χ_4 above, which is equivalent to $\gamma = \neg b \vee c \vee e$, does not have the same size as the one of γ, as $|\chi_4| = 7$ while $|\gamma| = 6$.
While these two measures are different, this is not a problem, as they always differ at most by a linear factor. Thus, this difference does not impact our results in the following.

Similarly to propositional formulae, we define the semantics of a pseudo-Boolean constraint through the definition of a *model* of this constraint.

Definition 21 (Model, Counter-Model of a Pseudo-Boolean Constraint)

Let χ be the pseudo-Boolean constraint $\sum_{i=1}^{n} \alpha_i \ell_i \vartriangle \delta$ An interpretation I over $\mathcal{V}$ such that $\mathsf{var}(\chi) \subseteq \mathcal{V}$ is a *model* of χ if and only if the inequality $\sum_{i=1}^{n} \alpha_i I(\ell_i) \vartriangle \delta$ is satisfied. It is said to be a *counter-model* of χ otherwise.

Example 14

The interpretation I with $I(b) = I(c) = I(e) = I(g) = I(h) = 0$ and $I(a) = I(d) = I(f) = 1$ is a model of the four constraints below:

- $6\bar{b} + 6c + 4e + f + g + h \geq 7$
- $5a + 4b + c + d \geq 6$
- $a + d + \bar{e} \geq 2$
- $\bar{b} + c + e \geq 1$

The interpretation I' given by $I'(a) = I'(c) = I'(d) = I'(e) = I'(g) = I'(h) = 0$ and $I'(b) = I'(d) = I'(f) = 1$ is a counter model of all these constraints.

The definitions of logical entailment, logical equivalence, consistency, contradiction and validity defined in the previous section can be lifted to pseudo-Boolean constraints by using the definition above for the notion of model.

Let us now define a normalized form for pseudo-Boolean constraints.

Definition 22 (Normalized Pseudo-Boolean Constraint)

A *normalized pseudo-Boolean constraint* is a pseudo-Boolean constraint of the form $\sum_{i=1}^{n} \alpha_i \ell_i \geq \delta$, in which

- for all $i \in \{1, \ldots, n\}, \bar{\ell}_i \notin \mathsf{lit}(\chi)$,
- for all $i \in \{1, \ldots, n\}, \alpha_i \in \mathbb{N}$, and
- $\delta \in \mathbb{N}$.

The following proposition justifies the name of *normalized* pseudo-Boolean constraint in the definition above.

Proposition 1 (From [Bar95, BM84a, BM84b, RM09b])

Any pseudo-Boolean constraint may be written as a conjunction of normalized pseudo-Boolean constraints that is equivalent to the original constraint in time linear in the size of the constraint.

Example 15

The constraint $6\bar{b} + 6c + 4e + f + g + h \geq 7$ is a normalized pseudo-Boolean constraint.

The constraint $5\bar{a} + 4\bar{b} + \bar{c} + \bar{d} < 5$ is a non-normalized pseudo-Boolean constraint. A normalized form for this constraint is $5a + 4b + c + d \geq 7$, as shown below:

$$\begin{aligned}
5\bar{a} + 4\bar{b} + \bar{c} + \bar{d} < 5 &\equiv 5\bar{a} + 4\bar{b} + \bar{c} + \bar{d} \leq 4 && \text{(left hand side in } \mathbb{N}) \\
&\equiv -5\bar{a} - 4\bar{b} - \bar{c} - \bar{d} \geq -4 && (\times -1) \\
&\equiv -5(1-a) - 4(1-b) - (1-c) - (1-d) \geq -4 && (\bar{\ell} = 1 - \ell) \\
&\equiv -5 + 5a - 4 + 4b - 1 + c - 1 + d \geq -4 \\
&\equiv 5a + 4b + c + d - 11 \geq -4 \\
&\equiv 5a + 4b + c + d \geq 11 - 4 \\
&\equiv 5a + 4b + c + d \geq 7
\end{aligned}$$

In the following, we apply Proposition 1 and thus consider, without loss of generality, that all pseudo-Boolean constraints are normalized. We also consider two main kinds of normalized pseudo-Boolean constraints, that are widely used when encoding a large variety of problems.

Definition 23 (Cardinality Constraint)

A *cardinality constraint* is a pseudo-Boolean constraint of the form $\sum_{i=1}^{n} \ell_i \geq \delta$. In other words, a cardinality constraint is a pseudo-Boolean constraint in which all coefficients are 1.

Example 16

The constraint $a + d + \bar{e} \geq 2$ is a cardinality constraint.

Observation 1 (Clauses and Pseudo-Boolean Constraints)

A *clause* is a pseudo-Boolean constraint of the form $\sum_{i=1}^{n} \ell_i \geq 1$. In other words, a clause is a pseudo-Boolean constraint in which all coefficients and the degree are 1. In this case, we have $\sum_{i=1}^{n} \ell_i \geq 1 \equiv \bigvee_{i=1}^{n} \ell_i$. Moreover, we have that the empty clause $\perp$ is equivalent to $\sum_{i=1}^{0} \ell_i \geq 1$, i.e, $0 \geq 1$.

Example 17

The constraint $\bar{b} + c + e \geq 1$ is a clause that is equivalent to $\neg b \vee c \vee e$.

Remark 7

The definition of normalized pseudo-Boolean constraints, as given by Definition 22, has been designed so that cardinality constraints generalize clauses, and normalized pseudo-Boolean constraints generalize cardinality constraints. This way, it is easier to adapt the algorithms that have been developed for clauses and CNF formulae.

It is worth noting that a single pseudo-Boolean constraint is equivalent to infinitely many other pseudo-Boolean constraints (see, e.g., [Knu08, Section 7.1.1]). This is not only due to the fact that one may multiply the coefficients and degree of a constraint by a constant to get an equivalent formula, but also because we can use different coefficients while preserving the semantics of the constraint, as illustrated in the example below.

Example 18

Observe that the following constraints are all equivalent:

- $18a + 12b + 6c \geq 22$
- $9a + 6b + 3c \geq 11$
- $9a + 6b + 3c \geq 12$
- $8a + 6b + 3c \geq 11$
- $7a + 3b + 3c \geq 10$
- $3a + 2b + c \geq 4$
- $3a + b + c \geq 4$
- $2a + b + c \geq 3$

The normal form of pseudo-Boolean constraints given in Definition 22 does not uniquely characterize a given pseudo-Boolean constraint. In order to get a canonical representation for a pseudo-Boolean constraint, one may for instance use the *Chow parameters* of this constraint, as described in [CLH11, Section 9.6]. However, an exponential time is required to compute these parameters (even though a pseudo-polynomial time algorithm exists for computing them, see e.g. [CLH11, Section 9.6.3]).

1.1.3 Pseudo-Boolean Constraints and CNF Formulae

As mentioned in Definition 1, clauses are a particular type of pseudo-Boolean constraints. As such, it is straightforward to translate any CNF formula into a pseudo-Boolean formula, simply by converting the logical representation of every clause to a pseudo-Boolean representation. However, retrieving pseudo-Boolean constraints from clauses encoding them is a difficult task. This is why different algorithms have been proposed to retrieve such constraints, paying a particular attention to the cardinality constraints encoded as clauses [BLLM14, EN20]. The other way around, it is also possible to find a CNF *representation* of a pseudo-Boolean constraint.

Definition 24 (CNF **Representation**)

A CNF *representation* of a pseudo-Boolean formula φ is a CNF formula ψ such that $\varphi \equiv \psi$ and such that $\mathsf{var}(\varphi) = \mathsf{var}(\psi)$.

The following proposition gives a CNF representation for any pseudo-Boolean constraint.

Proposition 2 ([BSS94])

Let χ be the pseudo-Boolean constraint $\sum_{i=1}^{n} \alpha_i \ell_i \geq \delta$. Let C be the set defined as

$$C = \left\{ L | L \subseteq \{\ell_1, \ldots, \ell_n\} \text{ and } \sum_{i | \ell_i \notin L} \alpha_i < \delta \right\}$$

The following CNF formula is equivalent to the constraint χ:

$$\bigwedge_{L \in C} \bigvee_{\ell_i \in L} \ell_i$$

Example 19

Let us consider the pseudo-Boolean constraint χ given by $4a + 2b + c + d \geq 6$. If we use the same notation as in Proposition 2 above, we have:

$$D = \{\{a\}, \{a, b\}, \{a, c\}, \{a, d\},\\ \{b, c\}, \{b, d\}, \{a, b, c\}, \{a, b, d\},\\ \{a, c, d\}, \{b, c, d\}, \{a, b, c, d\}\}$$

We remark here that, each time one of the sets $L \in D$ is a subset of another set $L' \in D$, we can remove L'. Indeed, consider for instance the sets $\{a\}$ $\{a, b\}$. These two sets will produce the clauses a and $a \vee b$, respectively. However, since $a \models a \vee b$, the second clause is redundant in the presence of the first one, so that we can remove it. As a consequence, the CNF representation of χ is given by

$$\chi \equiv (a) \wedge (b \vee c) \wedge (b \vee d)$$

Proposition 2 gives a representation of a pseudo-Boolean constraint into clauses that may be exponentially large, in particular when considering cardinality constraints, because of the following proposition.

Proposition 3 ([DGP04])

Let κ be the cardinality constraint given by $\sum_{i=1}^{n} \ell_i \geq \delta$. The smallest CNF representation of κ is given by

$$\bigwedge_{\substack{L \subseteq \mathsf{lit}(\kappa) \\ |L| = \delta + 1}} \bigvee_{\ell_i \in L} \ell_i$$

Proposition 3 shows in particular that a cardinality constraint may represent exponentially many clauses, for instance if we consider $\delta = 2n$ in the proposition: in this case, $\binom{2n}{n+1}$ clauses are needed.

Example 20

Let us consider the cardinality constraint κ given by $a + b + \bar{c} + d \geq 2$. A CNF representation of κ is the conjunction of the clauses that contain 3 literals from $\mathsf{lit}(\kappa) = \{a, b, \bar{c}, d\}$. We thus have:

$$\kappa \equiv (a \vee b \vee \bar{c}) \wedge (a \vee b \vee d) \wedge (a \vee \bar{c} \vee d) \wedge (b \vee \bar{c} \vee d)$$

Thus, computing a CNF representation of a pseudo-Boolean formula may lead to an exponential blow up in the size of the representation. This is why using CNF *encodings* is often preferred.

Definition 25 (CNF **Encoding)**

Let $\mathcal{V}$ and $\mathcal{A}$ be two disjoint sets of variables, and let φ and ψ be two formulae such that $\mathsf{var}(\varphi) = \mathcal{V}$ and $\mathsf{var}(\psi) = \mathcal{V} \cup \mathcal{A}$. ψ is a CNF *encoding* of φ if and only if ψ is a CNF formula and:

- for every model M of φ, there is an extension M' of M to $\mathcal{A}$ that is a model of ψ, and
- for every counter-model C of φ, no extension M' of M to $\mathcal{A}$ is a model of ψ

Variables from $\mathcal{A}$ are called *auxiliary variables*.

The use of auxiliary variables in CNF encodings is particularly useful, as they allow to produce formulae that are smaller in practice than the CNF representations of the original formula. This explains why many CNF encodings have been designed, as for instance Tseitin's transformation [Tse68], or the translation proposed by Plaisted and Greenbaum [PG86]. However, such encodings do not always preserve equivalence. In general, they only ensure that the produced formula is *equisatisfiable* to the original formula.

Definition 26 (Equisatisfiability)

Two formulae φ and ψ are said to be *equisatisfiable* when φ is satisfiable if and only if ψ is satisfiable.

In the case of pseudo-Boolean constraints, many different CNF encodings have been proposed, such as those implemented in *MiniSat+* [ES06], *NaPS* [SN15] or *OpenWBO* [MML14]. These encodings are studied later in this document (for more details, see Section 4.3).

1.2 Complexity Theory

In this section, we give some elementary notions of computational complexity that we use throughout the document. We assume the reader to be familiar with those notions, and refer to the literature for more details [GJ79, Pap94].

The *complexity* of a given problem is a measure of the hardness of solving this particular problem. Here, a *problem* is considered in a broad sense, as a statement that describes the input we have and the output that must be produced given this input. In the following, the complexity of a problem is evaluated through the *runtime* of algorithms that solve this problem, following the RAM model described in [CLRS09].

Definition 27 (Running Time of an Algorithm)

The *running time* (or simply *runtime*) of an algorithm on a particular input X is a function f mapping X to the number of *elementary instructions* required for the execution of the algorithm on this input.

As it is common practice in the literature, we consider in the following that the runtime of an algorithm depends on the *size* of its input, and consider the *worst case* running time, defined as follows.

Definition 28 (Worst Case Running Time of an Algorithm)

Let A be an algorithm having runtime f and let s be a function computing the size of any input X of A. The *worst case runtime* of A on inputs of size n is defined as a function r given by:

$$r(n) = \max\{f(X) | s(X) = n\}$$

In this case, we say that A runs in time $r(n)$.

With the definition above, observe that the worst case running time is a *upper bound* of the actual runtime of the considered algorithm, as the number of elementary instructions may vary on different inputs of the same size. It is thus convenient to consider an *order of growth* for the runtime, most of the time through the big-oh notation (Knuth-Landau).

Definition 29 ($\mathcal{O}(\cdot)$)

Given a function $r : \mathbb{N} \to \mathbb{N}$ and a function $f : \mathbb{N} \to \mathbb{N}$, we say that r is in $\mathcal{O}(f(n))$, denoted $r(n) = \mathcal{O}(f(n))$, if there exists $n_0 \in \mathbb{N}$ and a constant $c \in \mathbb{N}$ such that, for any $n \geq n_0$, $r(n) \leq cf(n)$.

In the following, we only consider the worst case runtime, and thus simply say *runtime* to refer to the *worst case runtime*. Let us now define the *complexity* of the algorithm.

Definition 30 (Complexity of an Algorithm)

The *(time) complexity of an algorithm* is said to be in $\mathcal{O}(f(n))$ when the runtime of the algorithm is upper bounded by $f(n)$.

We often prefer to refer to *problems* rather than algorithms that solve these problems. It is thus convenient to consider the *complexity of a problem*.

Definition 31 (Complexity of a Problem)

We say that the *complexity of a problem* P is in $\mathcal{O}(f(n))$ if there is an algorithm that solves P and runs in time $\mathcal{O}(f(n))$.

In the following, we mostly focus on *decision problems*, i.e., problems that consist in checking whether a given input satisfies a particular property. When this is the case, there exists a *polynomial-sized certificate* (that the algorithm may output or not) that proves that indeed, the input satisfies the property.

They correspond to formal languages L over an alphabet V. The positive instances of the problems (i.e., the ones satisfying the property), are the elements of V^* that belong to L. We now consider different complexity classes of decision problems, defined below.

Definition 32 (Complexity Class P)

A problem belongs to the *complexity class* P if and only if its complexity is in $\mathcal{O}(p(n))$ where p is a fixed polynomial. In this case, we also say that the problem may be solved in *polynomial time*, or that it is *tractable*.

Definition 33 (Complexity Class NP)

A problem belongs to the *complexity class* NP if and only if there exists a *polynomial time* algorithm A such that, for every positive instance X of P, there is a certificate C such that A accepts when given as input X and C. Moreover, for every negative instance X of P and any certificate C, A rejects.

Remark 8

Clearly, $\mathsf{P} \subseteq \mathsf{NP}$. However, we do not know whether $\mathsf{P} = \mathsf{NP}$, and it is commonly conjectured that actually $\mathsf{P} \neq \mathsf{NP}$.

Definition 34 (Complexity Class coNP)

The decision problem consisting in checking whether a given input satisfies a property Π belongs to the *complement* of the complexity class NP, denoted coNP, if and only if the decision problem checking whether a given input does *not* satisfy the property Π belongs to the complexity class NP.

In the following, the complexity class C represents either the complexity class NP or coNP. To determine whether a given problem belongs to the complexity class C, we use the notions defined below.

Definition 35 (Polynomial Reduction)

A problem P *is reduced in polynomial time* to a problem P' if and only if there exists a polynomial-time algorithm A that, given an instance X of P, computes an instance X' of P' such that X is a positive instance of P if and only if X' is a positive instance of P'.

Definition 36 (Hardness)

A problem P' is said to be C-*hard* if and only if for every problem P in the complexity class C, there exists a polynomial reduction from P to P'.

Definition 37 (Completeness)

A problem is said to be C-*complete* if and only if it is C-hard and it belongs to the complexity class C.

In order to prove that a problem P is C-hard, it is enough to identify a C-complete problem and a polynomial reduction reducing this problem to P (since reductions are transitive). For instance, to prove that a problem is NP-hard, one can take advantage to the following theorem and reduce the SAT problem described in the previous section to the considered problem.

Theorem 2 ([Coo71])

Propositional satisfiability is NP-complete.

Remark 9

If there exists a polynomial time algorithm for solving any NP-complete problem (e.g., the SAT problem), then $\mathsf{P} = \mathsf{NP}$.

1.3 Knowledge Compilation

A convenient way of representing pieces of propositional knowledge is to use propositional formulae. However, reasoning with such formulae may be computationally hard, and some operations may require an exponential time to be performed. For many applications, and in particular those involving interactions with users, runtime guarantees must be provided, which is incompatible with the time complexity of the operations to perform. To deal with this complexity, *knowledge compilation* [CD97] has been initiated about thirty years ago, starting from the following observation: most of the time, knowledge encoded with a propositional formula does not change much over the time.

Intuitively, knowledge compilation takes advantage of this fact to perform the "hard" operations once and for all, in an *offline* process, so that the operations to perform *online* become efficient. Ideally, for applications requiring runtime guarantees, the target complexity class is P, the complexity class of the problems solvable in polynomial time. To this purpose, many different compilation languages have been designed to reach this goal. The following section presents these languages.

1.3.1 Some Compilation Languages

Most of the compilation languages we study are based on the notion of *circuit*, that generalizes the notion of formula.

Definition 38 (Circuit)

A *circuit* is a directed acyclic graph Γ such that:

- the root and internal nodes of Γ are *gates*, i.e., logical operators ($\vee$, $\wedge$ or $\neg$), and
- the leaves of Γ are propositional variables or Boolean constants.

The root of Γ is called an *output* node. The *size* of the circuit Γ, denoted $|\Gamma|$, is the number of nodes in Γ.

As it is common when considering circuits with structurally restricted underlying graphs, we assume that every input variable appears in only one input gate. This property is sometimes called the *read-once property*. Using such circuits, we can represent any propositional formula, following the semantics defined below.

Definition 39 (Semantics of a Circuit)

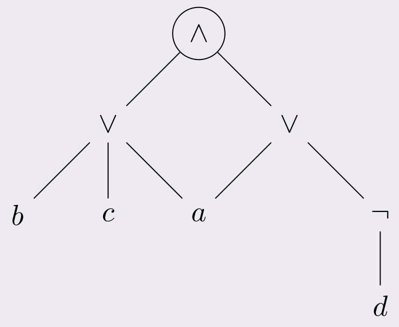
Let I be an interpretation of the variables of a circuit. The value of a node n in the circuit is recursively defined as:

- If n is a leaf labeled by a constant b, then $I(n) = I(b)$.
- If n is a leaf labeled by a variable v, then $I(n) = I(v)$.
- If n is a $\neg$ node having the node n' as child, then $I(n) = 1 - I(n')$.
- If n is an $\wedge$ node having the nodes $n_1, \ldots, n_k$ as children, then $I(n) = \min(I(n_1), \ldots, I(n_k))$.
- If n is an $\vee$ node having the nodes $n_1, \ldots, n_k$ as children, then $I(n) = \max(I(n_1), \ldots, I(n_k))$.

The definition of model given in Definition 7 can trivially be extended to circuits. Similarly, the definitions of logical entailment, logical equivalence, consistency, contradiction and validity defined in the previous section can be lifted to circuits based on this extended notion of model.

Example 21

Let us consider again the formula φ given by $(a \lor b \lor c) \land (a \lor \neg d)$. A representation of this formula as a circuit is given below:



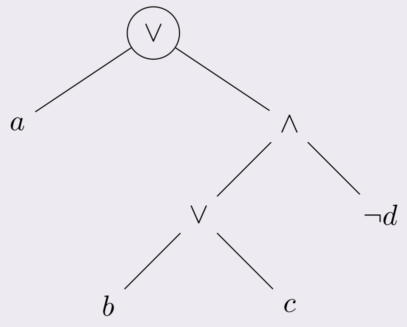
Most of the (circuit-based) compilation languages that we study are based on the language of circuit in *Negation Normal Form* (NNF).

Definition 40 (Negation Normal Form)

A circuit is said to be in *Negation Normal Form* (NNF) when all negation nodes ($\neg$) of the associated circuit only have leaves as children.

Example 22

Let us consider again the formula φ given by $a \lor \neg(\neg(b \lor c) \lor d)$. The representation of this formula as an NNF circuit is given below:



Notation 7

As negations are always parents of leaves in a formula in NNF, we often do not represent them as internal nodes. Instead, we prefer to have literals as leaves, as in the example above.

In a previous section, we already have mentioned two languages that are subsets of the NNF language, namely the CNF language (see Definition 12) and the DNF language (see Definition 14). From these two languages, we can define two other subsets of NNF, which are the language IP of *prime implicants* and PI of *prime implicates*.

Definition 41 (Prime Implicant)

Let φ be a circuit. A *prime implicant* τ of φ is a term such that $\tau \models \varphi$ and, for any term ψ such that $\psi \models \varphi$ and $\tau \models \psi$, then $\psi \equiv \tau$.

Example 23

a is a prime implicant of the formula $\varphi \equiv a \lor \neg(\neg(b \lor c) \lor d)$

Definition 42 (Prime Implicate)

Let φ be a circuit. A *prime implicate* γ of φ is a clause such that $\varphi \models \gamma$ and, for any clause ψ such that $\varphi \models \psi$ and $\psi \models \gamma$, then $\psi \equiv \gamma$.

Example 24

$a \lor b \lor c$ is a prime implicate of the formula $\varphi \equiv a \lor \neg(\neg(b \lor c) \lor d)$.

Definition 43 (IP, PI)

Let φ be a DNF (resp. CNF) formula. φ belongs to the language IP of *prime implicants* (resp. to the language PI of *prime implicates*) if it is written as the disjunction of its prime implicants (resp. the conjunction of its prime implicates).

Example 25

Let φ be the formula $a \lor \neg(\neg(b \lor c) \lor d)$. Its representation in the language IP is given by $a \lor (b \land \neg d) \lor (c \land \neg d)$ (see Example 10) while its representation in the language PI is given by $(a \lor b \lor c) \land (a \lor \neg d)$.

Let us now consider more general subsets of NNF, which are based on two properties that may be considered in an NNF formula: *decomposability* and *determinism*.

Definition 44 (Decomposability)

A conjunction $\varphi_1 \land \ldots \land \varphi_n$ is said to be *decomposable* if and only if $\mathsf{var}(\varphi_i) \cap \mathsf{var}(\varphi_j) = \emptyset$ for all $i \neq j$, i.e., if the φ_i do not share any variable.

Example 26

The conjunction $b \land \neg d$ is decomposable, while the conjunction $(a \lor b \lor c) \land (a \lor \neg d)$ is not, as a appears in both conjuncts.

Definition 45 (Determinism)

A disjunction $\varphi_1 \lor \ldots \lor \varphi_n$ is said to be *deterministic* if and only if $\varphi_i \land \varphi_j \models \bot$ for all $i \neq j$, i.e., if the φ_i do not share any model.

Example 27

The disjunction $(b \land \neg d) \lor (\neg b \land c \land \neg d)$ is deterministic while the disjunction $(b \land \neg d) \lor (c \land \neg d)$ is not. In the latter case, observe that, for instance, the model M such that $M(b) = M(c) = 1$ and $M(d) = 0$ is a model of both disjuncts.

Remark 10

Decomposability and determinism are particularly useful, for instance, to compute the number of models of a formula. Indeed, the number of models of a decomposable conjunction is exactly the product of the number of models of each conjunct, and the number of models of a deterministic disjunction is exactly the sum of the number of models of each disjunct, provided that the two disjuncts are based on the same set of variables.

Based on these two properties, we can now define some sublanguages of NNF.

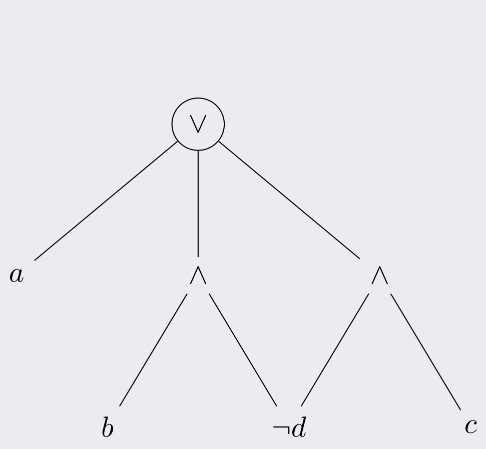
Definition 46 (DNNF, d-DNNF)

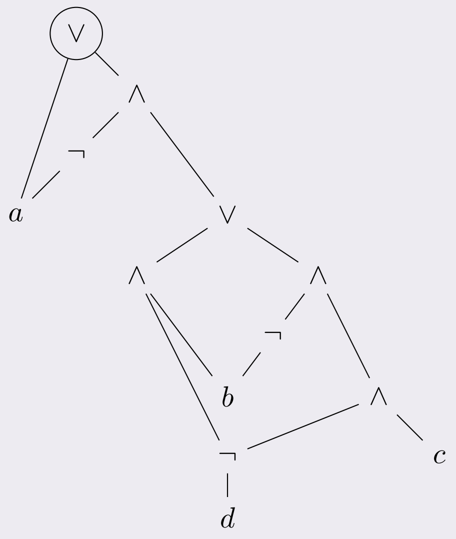
Given an NNF circuit φ, we say that φ is:

- in *Decomposable NNF* (DNNF) if all the conjunctions in φ are decomposable, and
- in *deterministic Decomposable NNF* (d-DNNF) if it is in DNNF and all the disjunctions in φ are deterministic.

Example 28

Let φ be the formula $a \lor \neg(\neg(b \lor c) \lor d)$. A representation of φ as a DNNF is given on the left below, while a d-DNNF representation of φ is given on the right:





Another main language on which different compilation languages are based is that of *Binary Decision Diagrams*.

Definition 47

A *Binary Decision Diagram* (BDD) is a directed acyclic graph Γ such that:

- the root and internal nodes of Γ are *decision nodes*, i.e., nodes representing variables,
- each arc is labeled by either 0 or 1, so as to represent the assignment of the variable of the node from which this arc starts from, and
- its leaves are exactly 0 and 1.

The *size* of the BDD Γ, denoted $|\Gamma|$, is the number of nodes in Γ.

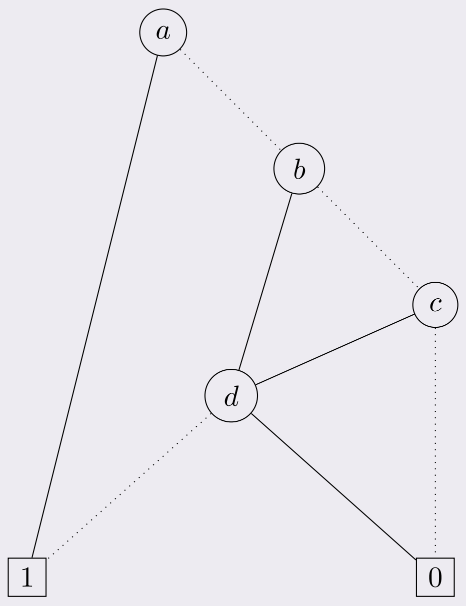
As for circuits, it is possible to represent any propositional formula using a BDD, following the semantics given below.

Definition 48 (Semantics of a BDD)

In a BDD, a path from the root to the leaf 0 (resp. 1) is a (partial or complete) interpretation falsifying (resp. satisfying) the BDD.

Example 29

Consider again the formula $\varphi = a \vee \neg(\neg(b \vee c) \vee d)$. The following diagram is a representation of φ as a BDD (for more readability, solid lines are those labeled by 1 while dotted lines are those labeled by 0).



In the following, we mostly consider the following subsets of Binary Decision Diagrams.

Definition 49 (Free Binary Decision Diagram)

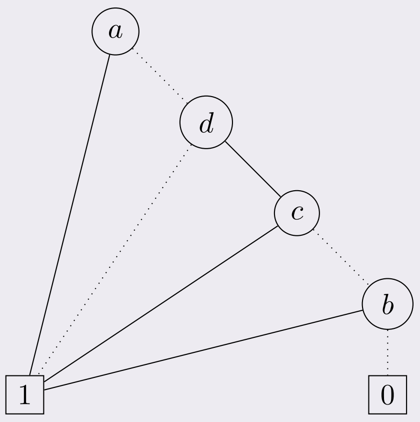
A *Free Binary Decision Diagram* (FBDD) is a BDD in which all variables appear at most once in each path from the root to any leaf of the BDD.

Definition 50 (Ordered Binary Decision Diagram)

Let φ be an FBDD and let $<$ be a total order on $\mathsf{var}(\varphi)$. Then φ is said to be an *Ordered Binary Decision Diagram with respect to* $<$ ($\mathsf{OBDD}_<$) if and only if, for each decision node N in φ and for each node M that is an ancestor of N in φ, we have $v_M < v_N$, where v_M and v_N are the variables associated with the nodes M and N, respectively.
The language of *Ordered Binary Decision Diagrams* (OBDD) is defined as the union of all the $\mathsf{OBDD}_<$ languages.

Example 30

Consider again the formula $\varphi = a \vee \neg(\neg(b \vee c) \vee d)$. The BDD given in Example 29 is an $\mathsf{OBDD}_<$ for the order $a < b < c < d$.
Below is an $\mathsf{OBDD}_<$ representing φ with respect to the order $a < d < c < b$.



The list of compilation languages presented in this section is not exhaustive. The reader is referred to the *Knowledge Compilation Map* [DM02] for more details about the numerous existing compilation languages.

1.3.2 A Knowledge Compilation Map

In order to identify the most appropriate compilation language(s) for a given application, one may consider the criteria of the *Knowledge Compilation Map* [DM02]. This map considers commonly used compilation languages, and compares their spatial efficiency and their temporal efficiency with respect to the *queries* and *transformations* they offer or not.

Expressivity and Succinctness

The first criterion of the knowledge compilation map we consider is the one evaluating the capacity of a language to represent knowledge. For a given application, the language to choose must be *expressive* enough to represent the information needed for the application.

Definition 51 (Expressiveness)

A language L is said to be *at least as expressive as* another language L', denoted $\mathsf{L} \leq_e \mathsf{L}'$ if and only if, for every Boolean function f having a representation $\varphi \in \mathsf{L}'$, there exists a formula $\psi \in \mathsf{L}$ that also represents f.

Another criterion to take into account when choosing a language for a particular application is its spatial efficiency, characterized by its *succinctness*.

Definition 52 (Succinctness)

A language L is said to be *at least as succinct as* another language L′, denoted L $\leq_s$ L′ if and only if there exists a polynomial p such that, for every Boolean function f having a representation $\varphi \in$ L′, there exists $\psi \in$ L that also represents f and is such that $|\psi| \leq p(|\varphi|)$.

Remark 11

The relation $\leq_s$ is a pre-order.

Notation 8

Given two languages L and L′, we write L $<_s$ L′ to denote that L $\leq_s$ L′ and L $\not\leq_s$ L′.

The common languages presented above, and all the others studied in the knowledge compilation map are all as expressive as the others, since they are all fully expressive. Regarding their succinctness, their comparison is given in Table 1.2.

	NNF	DNNF	d-DNNF	sd-DNNF	FBDD	OBDD	OBDD$_<$	DNF	CNF	PI	IP	MODS
NNF	$\leq$	$\leq$	$\leq$	$\leq$	$\leq$	$\leq$	$\leq$	$\leq$	$\leq$	$\leq$	$\leq$	$\leq$
DNNF	$\not\leq^{(1)}$	$\leq$	$\leq$	$\leq$	$\leq$	$\leq$	$\leq$	$\leq$	$\not\leq^{(1)}$	$\not\leq^{(1)}$	$\leq$	$\leq$
d-DNNF	$\not\leq^{(1)}$	$\not\leq^{(1)}$	$\leq$	$\leq$	$\leq$	$\leq$	$\leq$	$\not\leq^{*}$	$\not\leq^{(1)}$	$\not\leq^{(1)}$	?	$\leq$
sd-DNNF	$\not\leq$	$\not\leq$	$\leq$	$\leq$	$\leq$	$\leq$	$\leq$	$\not\leq$	$\not\leq$	$\not\leq$	$\not\leq$	$\leq$
FBDD	$\not\leq$	$\not\leq$	$\not\leq$	$\not\leq$	$\leq$	$\leq$	$\leq$	$\not\leq$	$\not\leq$	$\not\leq$	$\not\leq$	$\leq$
OBDD	$\not\leq$	$\not\leq$	$\not\leq$	$\not\leq$	$\not\leq$	$\leq$	$\leq$	$\not\leq$	$\not\leq$	$\not\leq$	$\not\leq$	$\leq$
OBDD$_<$	$\not\leq$	$\not\leq$	$\not\leq$	$\not\leq$	$\not\leq$	$\not\leq$	$\leq$	$\not\leq$	$\not\leq$	$\not\leq$	$\not\leq$	$\leq$
DNF	$\not\leq$	$\not\leq$	$\not\leq$	$\not\leq$	$\not\leq$	$\not\leq$	$\not\leq$	$\leq$	$\not\leq$	$\not\leq$	$\leq$	$\leq$
CNF	$\not\leq$	$\not\leq$	$\not\leq$	$\not\leq$	$\not\leq$	$\not\leq$	$\not\leq$	$\not\leq$	$\leq$	$\leq$	$\not\leq$	$\leq$
PI	$\not\leq$	$\not\leq$	$\not\leq$	$\not\leq$	$\not\leq$	$\not\leq$	$\not\leq$	$\not\leq$	$\not\leq$	$\leq$	$\not\leq$	$\not\leq^{(2)}$
IP	$\not\leq$	$\not\leq$	$\not\leq$	$\not\leq$	$\not\leq$	$\not\leq$	$\not\leq$	$\not\leq$	$\not\leq$	$\not\leq$	$\leq$	$\leq$
MODS	$\not\leq$	$\not\leq$	$\not\leq$	$\not\leq$	$\not\leq$	$\not\leq$	$\not\leq$	$\not\leq$	$\not\leq$	$\not\leq$	$\not\leq$	$\leq$

Table 1.2: Comparison of different compilation languages in terms of succinctness. In this table, if the symbol $\triangleright$ appears in the row of the language R and in the column of the language C, then we have R $\triangleright$ C. A $*$ represents that the result is true if NP $\subseteq$ P. Results with a (1) come from [BCMS16], while results with a (2) are from [Kal17]. Other results are from [DM02].

.

Queries Offered by a Compilation Language

Depending on the needs of the considered application, it is important to consider the temporal efficiency of different *queries* on a formula represented by the language used to represent the information we have. In particular, these criteria aim to identify which queries are tractable (i.e., solvable in polynomial time), and which are not (unconditionally or unless P $=$ NP). Such queries allow to answer different *questions* with respect to the input representation that are presented below for a given compilation language L.

Definition 53 (COnsistency, VAlidity)

A language L satisfies CO (resp. VA) if and only if there exists a polynomial time algorithm verifying whether a given representation φ in L is consistent (resp. valid).

Definition 54 (Clausal Entailment)

A language L satisfies CE if and only if there exists a polynomial time algorithm verifying whether a given representation φ in L entails a given clause γ.

Definition 55 (Sentential Entailment, EQuivalence)

A language L satisfies SE (resp. EQ) if and only if there exists a polynomial time algorithm verifying, given two representations φ and ψ in L, whether φ entails ψ (resp. φ is equivalent to ψ).

Definition 56 (IMplication by a term)

A language L satisfies IM if and only if there exists a polynomial time algorithm verifying whether a given representation φ in L is entailed by a given term τ.

Definition 57 (CounTing)

A language L satisfies CT if and only if there exists a polynomial time algorithm counting the number of models of a given representation φ in L.

Definition 58 (Model Enumeration)

A language L satisfies ME if and only if there exists a polynomial time algorithm enumerating the models of a representation φ in L in polynomial time with respect to $|\varphi|$ and the number of models of φ.

Remark 12

Alternatively, a language L satisfies ME if and only if there exists an algorithm enumerating the models of a representation φ in L with a polynomial delay between each enumerated model. The results do not change if we consider this definition of ME instead of Definition 58.

The properties of the different compilation languages of the knowledge compilation map in terms of offered queries are given in Table 1.3.

	CO	VA	CE	IM	EQ	SE	CT	ME
NNF	○	○	○	○	○	○	○	○
DNNF	✓	○	✓	○	○	○	○	✓
d-DNNF	✓	✓	✓	✓	?	○	✓	✓
sd-DNNF	✓	✓	✓	✓	?	○	✓	✓
BDD	○	○	○	○	○	○	○	○
FBDD	✓	✓	✓	✓	?	○	✓	✓
OBDD	✓	✓	✓	✓	✓	○	✓	✓
$\mathsf{OBDD}_<$	✓	✓	✓	✓	✓	✓	✓	✓
DNF	✓	○	✓	○	○	○	○	✓
CNF	○	✓	○	✓	○	○	○	○
PI	✓	✓	✓	✓	✓	✓	○	✓
IP	✓	✓	✓	✓	✓	✓	○	✓
MODS	✓	✓	✓	✓	✓	✓	✓	✓

Table 1.3: Properties of different compilation languages in terms of queries. A ✓ means that the query is offered by the language in polynomial time, and a ○ means that it is not the case, unless $\mathsf{P} = \mathsf{NP}$.

Transformations Offered by a Compilation Language

Other criteria to take into account when choosing a compilation language for a given application are the *transformations* that may be applied efficiently to the formulae of this language. In particular, this criterion aims to identify which transformations are tractable (i.e., computable in polynomial time), and which are not (unconditionally or unless $\mathsf{P} = \mathsf{NP}$). Such transformations allow to compute a formula obtained from an input formula by applying logical operations on it, presented here for a given compilation language L.

Definition 59 (Conditioning)

Given a circuit φ and a literal ℓ, the *conditioning* of φ by ℓ, denoted $\varphi|\ell$, is the circuit obtained by replacing each occurrence of ℓ in φ by $\top$ if ℓ is a positive literal and by $\bot$ otherwise. The conditioning of φ by a consistent term τ is equivalent to the conditioning of φ by all the literals in τ.

Definition 60 (ConDitioning by a Term)

A language L satisfies CD if and only if there exists an algorithm computing the representation in L of the conditioning of a representation φ of L by a consistent term τ in polynomial time with respect to $|\varphi|$ and $|\tau|$.

Definition 61 (Forgetting of a variable)

Let φ be a circuit and v be a propositional variable. The *forgetting of v in φ*, denoted $\exists v\varphi$, is the most general logical consequence of φ that does not depend on v, in the sense that φ is equivalent to a formula in which v does not occur. Otherwise said, for any circuit ψ that does not contain v, $\varphi \models \psi$ if and only if $\exists v\varphi \models \psi$. In particular, $\exists v\varphi \equiv (\varphi|v) \vee (\varphi|\neg v)$, so that any model of $\exists v\varphi$ can be extended to a model of φ.
The forgetting of a set of variables $\mathcal{V}$ in φ is the forgetting in φ of all the variables in $\mathcal{V}$.

Definition 62 (FOrgetting, Singleton FOrgetting)

A language L satisfies FO (resp. SFO) if and only if there exists an algorithm A computing a representation in L of the forgetting of a set of propositional variables (resp. a singleton) $\mathcal{V}$ in a representation φ of L such that A runs in polynomial time with respect to $|\varphi|$ and $|\mathcal{V}|$.

Definition 63 (Closure under Conjunction, Disjunction)

A language L satisfies $\wedge$C (resp. $\vee$C) if and only if there exists an algorithm computing the representation in L of the conjunction (resp. disjunction) of a finite set of representations of L in polynomial time with respect to the size of the representations in the set.

Definition 64 (Bounded Closure under Conjunction, Disjunction)

A language L satisfies $\wedge$BC (resp. $\vee$BC) if and only if there exists an algorithm computing the representation in L of the conjunction (resp. disjunction) of two representations φ and ψ of L in polynomial time with respect to $|\varphi|$ and $|\psi|$.

Definition 65 (Closure under Negation)

A language L satisfies $\neg$C if and only if there exists an algorithm computing the representation in L of the negation of a representation φ of L in polynomial time with respect to $|\varphi|$.

The properties of the different compilation languages of the knowledge compilation map in terms of offered transformations are given in Table 1.4.

	CD	FO	SFO	$\wedge$C	$\wedge$ BC	$\vee$C	$\vee$BC	$\neg$C
NNF	✓	○	✓	✓	✓	✓	✓	✓
DNNF	✓	✓	✓	○	○	✓	✓	○
d-DNNF	✓	○	○	○	○	○	○	?
sd-DNNF	✓	○	○	○	○	○	○	?
BDD	✓	○	✓	✓	✓	✓	✓	✓
FBDD	✓	●	○	●	○	●	○	✓
OBDD	✓	●	✓	●	○	●	○	✓
OBDD$_<$	✓	●	✓	●	✓	●	✓	✓
DNF	✓	✓	✓	●	✓	✓	✓	●
CNF	✓	○	✓	✓	✓	●	✓	●
PI	✓	✓	✓	●	●	●	✓	●
IP	✓	●	●	●	✓	●	●	●
MODS	✓	✓	✓	●	✓	●	●	●

Table 1.4: Properties of different compilation languages in terms of transformations. A ✓ means that the transformation is offered by the language in polynomial time, a ○ means that it is not the case, unless P = NP and a ● means that it is not unconditionally.

Chapter 2

Pseudo-Boolean Constraints from a Knowledge Representation Perspective

As mentioned in the previous chapter, one of the main advantages of pseudo-Boolean constraints is that a single pseudo-Boolean constraint may represent an exponential number of clauses (see Proposition 3). This means that the language of pseudo-Boolean formulae is *strictly more succinct* than the language of CNF formulae. Starting from this observation, this chapter studies the properties of pseudo-Boolean constraints from a knowledge representation perspective. Specifically, we consider the criteria of the knowledge compilation map [DM02] – namely succinctness, polynomial-time queries and transformations – to systematically compare pseudo-Boolean languages with other well-known compilation languages, with a particular attention paid to the CNF language.

2.1 Some Properties of Pseudo-Boolean Constraints

First, let us study some intrinsic properties of pseudo-Boolean constraints, by considering the two languages defined below.

Definition 66 (1-PBC, 1-CARD)

1-PBC (resp. 1-CARD) is the language of pseudo-Boolean formulae composed of a single normalized pseudo-Boolean constraint (resp. cardinality constraint).

It is clear that, despite being more expressive than clauses, these languages are not expressive enough to represent all Boolean functions (see, e.g., [Sma07]), so we do not consider them as *representation languages*. Instead, we consider other kind of properties for these languages, showing that, in a sense, the gain in expressivity with respect to clauses is accompanied with an increased difficulty for the handling of pseudo-Boolean constraints.

For instance, as already mentioned in Chapter 1, there exists an infinite number of normalized pseudo-Boolean constraints that are equivalent to a given pseudo-Boolean constraint. This is not only due to the fact that one may arbitrarily multiply the coefficients and the degree of a constraint by a constant, but also to more subtle properties, such as that illustrated by the *increasible-degree* problem. This problem is deciding whether it is possible to increase the degree of a pseudo-Boolean constraint while preserving equivalence, as in the following example.

Example 31

Let χ be the pseudo-Boolean constraint $9a + 6b + 3c + d \geq 11$, and let χ' be the pseudo-Boolean constraint $9a + 6b + 3c + d \geq 12$. Observe that $\chi \equiv \chi'$, so that the degree of χ can be increased while preserving equivalence.

Proposition 4

The *increasible-degree* problem is coNP-hard.

Proof. Let us reduce the subset-sum problem to the complement of the increasible-degree problem. We consider a set $S = \{\alpha_i | 1 \leq i \leq n\} \subseteq \mathbb{N}$ and a number $\delta \in \mathbb{N}$. We want to know whether there exists a subset S' of S such that $\sum_{\alpha_i \in S'} \alpha_i = \delta$. To represent this problem, let us consider the pseudo-Boolean constraint χ given by $\sum_{i=1}^{n} \alpha_i v_i \geq \delta$, in which $v_i = 1$ means that $\alpha_i \in S'$.

On the one hand, there exists a set S' as described above if and only if the constraint $\sum_{i=1}^{n} \alpha_i v_i = \delta$ has a solution.

On the other hand, if there exists a solution of this constraint, then it is not possible to increase the degree of χ while preserving equivalence, and reciprocally. Indeed, none of the models of $\sum_{i=1}^{n} \alpha_i v_i = \delta$ are models of $\sum_{i=1}^{n} \alpha_i v_i \geq \delta + 1$.

Thus, there exists a solution to the subset-sum problem if and only if it is not possible to increase the degree of χ while preserving equivalence, so that the NP-complete subset-sum problem can be reduced to the complement of the increasible-degree problem, hence the result.

□

A related problem is the *maximum-degree* problem, which is finding the maximum possible degree for a pseudo-Boolean constraint. This problem is particularly useful from a knowledge representation perspective, at it allows to strengthen the constraint *over the reals*, while the constraint remains equivalent over the Booleans. Yet, this problem is also coNP-hard.

Corollary 1

The maximum-degree problem is coNP-hard.

Proof. The result is trivial, since if the maximum degree for a constraint χ is strictly greater than the degree of this constraint, then it is possible to increase this degree while preserving equivalence. Thus, the increasible-degree problem can be reduced to the maximum-degree problem, and the claim follows.

□

Remark 13

It is never possible to increase the degree of a consistent cardinality constraint $\sum_{i=1}^{n} \ell_i \geq \delta$, as doing so would not preserve the models of this constraint that satisfy exactly δ literals.

Let us now consider the queries and transformations that may be performed on a single pseudo-Boolean constraint or on a cardinality constraint. The results are given in Tables 2.1 and 2.2.

	CO	VA	CE	IM	EQ	SE	CT	ME
1-CARD	✓	✓	✓	✓	✓	✓	✓	✓
1-PBC	✓	✓	✓	✓	◦	◦	◦	✓

Table 2.1: Properties of 1-CARD and 1-PBC in terms of queries. A ✓ means that the query is offered by the language in polynomial time, and a ◦ means that it is not the case, unless $\mathsf{P} = \mathsf{NP}$.

	CD	FO	SFO	$\wedge$C	$\wedge$BC	$\vee$C	$\vee$BC	$\neg$C
1-CARD	✓	✓	✓	●	●	●	●	✓
1-PBC	✓	✓	✓	●	●	●	●	✓

Table 2.2: Properties of 1-CARD and 1-PBC about transformations. A ✓ means that the language offers the transformation in polynomial time, whereas a ● means that it does not (unconditionally).

Even though performing such operations on these types of constraint is not always meaningful, the main purpose of this study is to highlight that performing some of them (especially, computing the results of different queries) is already hard (while they may be trivial for clauses, for instance). The proofs of these results follow.

Proposition 5

Let χ be the pseudo-Boolean constraint $\sum_{i=1}^{n} \alpha_i \ell_i \geq \delta$. The constraint χ is consistent if and only if $\sum_{i=1}^{n} \alpha_i \geq \delta$.

Proof. On the one hand, if $\sum_{i=1}^{n} \alpha_i \geq \delta$, then satisfying all the literals of χ also satisfies the constraint. On the other hand, if $\sum_{i=1}^{n} \alpha_i < \delta$, it is clear that χ cannot be satisfied, as even satisfying all its literals is not enough to satisfy the constraint.

□

Corollary 2

Checking whether a cardinality constraint (resp. a pseudo-Boolean constraint) is consistent can be performed in polynomial time, i.e., 1-CARD (resp. 1-PBC) satisfies CO.

Proof. Let χ be the pseudo-Boolean constraint $\sum_{i=1}^{n} \alpha_i \ell_i \geq \delta$. Computing $\sum_{i=1}^{n} \alpha_i$ and checking whether this value is at least equal to δ can be done in polynomial time, and is enough to determine the consistency of χ by Proposition 5.

□

Proposition 6

Given a cardinality constraint (resp. a pseudo-Boolean constraint) χ and a term τ, computing $\chi|\tau$ can be done in polynomial time. As a consequence, 1-CARD (resp. 1-PBC) satisfies CD.

Proof. Replacing a literal by a Boolean constant in a cardinality constraint (resp. a pseudo-Boolean constraint) gives a new cardinality constraint (resp. pseudo-Boolean constraint) by a normalization running in polynomial time.

□

Corollary 3

Given a cardinality constraint (resp. a pseudo-Boolean constraint) χ, enumerating the models of χ can be done with a polynomial delay. Otherwise said, 1-CARD (resp. 1-PBC) satisfies ME.

Proof. The result is also a direct consequence of Corollary 2 and Proposition 6: as 1-CARD (resp. 1-PBC) satisfies both CO and CD, it also satisfies ME [DM02, Lemma A.3].

□

Proposition 7

Let χ be the pseudo-Boolean constraint $\sum_{i=1}^{n} \alpha_i \ell_i \geq \delta$, and let γ be a clause. We have that $\chi \models \gamma$ if and only if the following holds:

$$\sum_{\substack{i=1 \\ \ell_i \notin \mathsf{lit}(\gamma)}}^{n} \alpha_i < \delta$$

Proof. Recall that $\chi \models \gamma$ if and only if $\chi \wedge \neg\gamma \models \bot$. Observe that $\neg\gamma$ is a term that contains exactly the negation of the literals of γ, and

$$\chi \wedge \neg\gamma \equiv \sum_{\substack{i=1 \\ \ell_i \notin \mathsf{lit}(\gamma)}}^{n} \alpha_i \ell_i \geq \delta$$

Thus, $\chi \models \gamma$ if and only if the constraint above is inconsistent. By Proposition 5, this is the case exactly when

$$\sum_{\substack{i=1 \\ \ell_i \notin \mathsf{lit}(\gamma)}}^{n} \alpha_i < \delta$$

and the claim follows.

□

Corollary 4

Given a cardinality constraint κ of degree δ and a clause γ, we have $\kappa \models \gamma$ if and only if $|\mathsf{lit}(\kappa) \backslash \mathsf{lit}(\gamma)| < \delta$.

Proof. The result is a direct consequence of Proposition 7, as we simply apply the criterion described in this proposition to the case in which each α_i is 1.

$\square$

Corollary 5

Given a cardinality constraint (resp. a pseudo-Boolean constraint) χ and a clause γ, checking whether χ entails γ can be done in polynomial time. Otherwise said, 1-CARD (resp. 1-PBC) satisfies CE.

Proof. The result is a direct consequence of Proposition 7, as its criterion can be checked in polynomial time.

$\square$

Proposition 8

Let χ be a cardinality constraint (resp. a pseudo-Boolean constraint), and let δ be the degree of χ. The constraint χ is valid if and only if $\delta = 0$.

Proof. Let χ be the constraint $\sum_{i=1}^{n} \alpha_i \ell_i \geq \delta$.

On the one hand, if $\delta = 0$, it is clear that $\sum_{i=1}^{n} \alpha_i \ell_i \geq 0$ is always satisfied, as all coefficients are non-negative integers (recall that all constraints are supposed to be normalized).

On the other hand, if $\delta > 0$, then the constraint cannot be valid. Indeed, if an interpretation I falsifies all the literals of χ – which is possible since all constraints are supposed to be normalized, and thus cannot contain a literal and its negation – then $\sum_{i=1}^{n} \alpha_i \ell_i = 0 < \delta$. So, I is a counter-model of χ, which is thus not valid.

This proof also holds in the particular case of a cardinality constraint, and the claim follows.

$\square$

Corollary 6

Checking whether a cardinality constraint (resp. a pseudo-Boolean constraint) is valid can be done in polynomial time, i.e., 1-CARD (resp. 1-PBC) satisfies VA.

Proof. Checking whether a cardinality constraint (resp. a pseudo-Boolean constraint) has a degree equal to 0 can be done in polynomial time, and is enough to determine the validity of χ by Proposition 8.

$\square$

Corollary 7

Given a cardinality constraint (resp. a pseudo-Boolean constraint) χ and a term τ, checking whether τ entails χ can be done in polynomial time.

Proof. The result is a direct consequence of Proposition 6 and Corollary 6: as 1-CARD (resp. 1-PBC) satisfies both CD and VA, it also satisfies IM [DM02, Lemma A.7].

□

Proposition 9

Let κ and κ' be two consistent cardinality constraints $\kappa = \sum_{\ell \in L} \ell \geq \delta$ and $\kappa' = \sum_{\ell' \in L'} \ell' \geq \delta'$, respectively. $\kappa \models \kappa'$ if and only if $\delta' = 0$ or $|L \backslash L'| \leq \delta - \delta'$.

Proof. First, observe that, if $\delta' = 0$, then κ' is valid, and the result is trivial. Let us now consider the case in which $\delta' > 0$.

First, assume that $|L \backslash L'| \leq \delta - \delta'$, and let us prove that $\kappa \models \kappa'$. Let I be an interpretation that is a model of κ, and let M be the set of the literals satisfied by I. Such an interpretation exists, as κ is assumed consistent. Let us now prove that I satisfies at least δ' literals of L', i.e., that M contains these literals. Observe that M is equal to the disjoint union $(M \cap L') \cup (M \backslash L')$. In terms of cardinality, we thus have that $|M| = |M \cap L'| + |M \backslash L'|$. As I is a model of κ, it satisfies at least δ literals of κ, so that $\delta \leq |M|$. We thus conclude that $\delta \leq |M \cap L'| + |M \backslash L'|$ (1). Moreover, since $M \subseteq L$, we also have that $M \backslash L' \subseteq L \backslash L'$, and thus $|M \backslash L'| \leq |L \backslash L'| \leq \delta - \delta'$ (2). If we now combine (1) and (2), we get $\delta \leq |M| = |M \cap L'| + |M \backslash L'| \leq |M \cap L'| + \delta - \delta'$. In particular, $\delta \leq |M \cap L'| + \delta - \delta'$, and thus $\delta - \delta + \delta' \leq |M \cap L'|$, i.e., $\delta' \leq |M \cap L'|$, so that I is a model of κ'.

Now, suppose that $|L \backslash L'| > \delta - \delta'$, and let us prove that $\kappa \not\models \kappa'$. To this end, let us construct a model of κ that is not a model of κ'. Let us note M the set of the literals satisfied by this model. Consider first the case in which $|L \backslash L'| \geq \delta$, and let $M = L \backslash L'$. The interpretation satisfying exactly the literals of M satisfies at least δ literals of κ, but none of κ'. This interpretation is thus a model of κ, but not of κ'. Let us now consider the case in which $|L \backslash L'| < \delta$, and let us construct M as follows. First, let us put all the literals of $L \backslash L'$ in M. Then, we choose exactly $\delta - |L \backslash L'|$ literals in $L \cap L'$. This is possible as κ is assumed to be consistent: it is possible to satisfy δ literals in L. Since we have $|L \backslash L'| < \delta$, the $\delta - |L \backslash L'|$ literals that are missing to satisfy the constraint are thus necessarily in $L \cap L'$. By construction, we thus have that $|M \cap L'| = \delta - |L \backslash L'|$. Since we have supposed that $|L \backslash L'| > \delta - \delta'$, we have that $|M \cap L'| < \delta - \delta + \delta' = \delta'$. So, the interpretation satisfying the literals of M is a model of κ, since it satisfies δ literals of this constraint, but it is not a model κ', since it satisfies strictly less than δ' of its literals.

In both cases, we have built a model M of κ that is not a model of κ', and the claim follows.

□

Corollary 8

Given two cardinality constraints κ and κ', checking whether $\kappa \models \kappa'$ can be done in polynomial time. Otherwise said, 1-CARD satisfies SE.

Proof. If one of the two constraints is inconsistent – which can be decided in polynomial time by Corollary 2 – it is easy to verify whether $\kappa \models \kappa'$. Otherwise, one simply applies Proposition 9, which can be done in polynomial time.

□

Corollary 9

Given two cardinality constraints κ and κ', checking whether $\kappa \equiv \kappa'$ can be done in polynomial time. Said differently, 1-CARD satisfies EQ.

Proof. To check whether $\kappa \equiv \kappa'$, one simply need to check whether $\kappa \models \kappa'$ and $\kappa' \models \kappa$, which can be done in polynomial time by Corollary 8.

□

Proposition 10

Given two pseudo-Boolean constraints χ and χ', checking whether $\chi \equiv \chi'$ is coNP-hard. Said differently, 1-PBC does not satisfy EQ.

Proof. Let us reduce the *increasible-degree* problem, which is coNP-hard by Proposition 4, to the problem of the equivalence of two pseudo-Boolean constraints.

To do so, simply observe that given a pseudo-Boolean constraint $\sum_{i=1}^{n} \alpha_i \ell_i \geq \delta$ is equivalent to the other constraint $\sum_{i=1}^{n} \alpha_i \ell_i \geq \delta + 1$ if and only if it is possible to increase the degree of the former constraint.

□

Corollary 10

The language 1-PBC does not satisfy SE, unless P = NP.

Proof. As 1-PBC does not satisfy EQ unless P = NP by Proposition 10, it cannot satisfy SE unless P = NP, as checking whether two pseudo-Boolean constraints χ and χ' are equivalent can be achieved by checking that $\chi \models \chi'$ and $\chi' \models \chi$.

□

Proposition 11

Given a pseudo-Boolean constraint χ, determining a pseudo-Boolean constraint that is equivalent to $\neg\chi$ can be achieved in polynomial time. Said differently, 1-PBC satisfies ¬C.

Proof. Let χ be the pseudo-Boolean constraint $\sum_{i=1}^{n} \alpha_i \ell_i \geq \delta$. By definition, $\neg\chi \equiv \sum_{i=1}^{n} \alpha_i \ell_i < \delta$.

The negation of χ is thus obtained by normalizing this constraint, which gives $\sum_{i=1}^{n} \alpha_i \bar{\ell_i} \geq \sum_{i=1}^{n} \alpha_i - \delta$. As explained previously, this operation can be performed in polynomial time.

□

Corollary 11

Given a cardinality constraint κ, determining a cardinality constraint that is equivalent to $\neg\kappa$ can be achieved in polynomial time. Otherwise said, 1-CARD satisfies $\neg$C.

Proof. Let κ be the cardinality constraint $\sum_{i=1}^{n} \ell_i \geq \delta$. If we apply the same reasoning as in the proof of Proposition 11, we have that $\neg\kappa \equiv \sum_{i=1}^{n} \bar{\ell}_i \geq n - \delta$. Once again, this constraint can trivially be obtained from κ in polynomial time.

□

Proposition 12

Counting the model of a cardinality constraint can be done in polynomial time. Said differently, 1-CARD satisfies CT.

Proof. Let κ be the cardinality constraint $\sum_{i=1}^{n} \ell_i \geq \delta$. The models of κ are the interpretations that satisfy at least δ of the literals of κ. There are exactly $\sum_{i=\delta}^{n} \binom{n}{i}$ such interpretations, which can be computed in polynomial time. Indeed, observe that, given κ, the value of n is given in unary (it is the number of literals in the constraint), so that the sum can be computed with a number that is polynomial in n and δ, with $\delta \leq n$.

□

Proposition 13

Counting the number of models of a pseudo-Boolean constraint is NP-hard. Said differently, 1-PBC does not satisfy CT, unless P = NP

Proof. Let us show that the subset-sum problem can be reduced to the problem of counting the number of models of a pseudo-Boolean constraint. Let $S = \{\alpha_i | 1 \leq i \leq n\} \subseteq \mathbb{N}$, and $\delta \in \mathbb{N}$. We want to find a subset S' of S such that $\sum_{\alpha_i \in S'} = \delta$. Let $x_1, \ldots, x_n$ be propositional variables, such that $x_i = 1$ if and only if $\alpha_i \in S'$.

In order to solve this problem, let us consider the two pseudo-Boolean constraints χ_+ defined by $\sum_{i=1}^{n} \alpha_i x_i \geq \delta$ and χ_- defined by $\sum_{i=1}^{n} \alpha_i \bar{x}_i \geq \sum_{i=1}^{n} \alpha_i + \delta$. Note that χ_- is equivalent to $\sum_{i=1}^{n} \alpha_i x_i \leq \delta$.

Now, observe that any interpretation satisfies at least one of these constraints, and that all interpretations satisfying both constraints are exactly those verifying $\sum_{i=1}^{n} \alpha_i x_i = \delta$.

As there exist exactly 2^n interpretations of the n literals of χ_+ and χ_-, the number of interpretations satisfying both constraints, and thus satisfying $\sum_{i=1}^{n} \alpha_i x_i = \delta$, is equal to $\#(\chi_+) + \#(\chi_-) - 2^n$, where $\#(\chi_+)$ and $\#(\chi_-)$ denote the number of models of χ_+ and χ_-, respectively.

As this number is not equal to 0 if and only if the subset S' exists, the subset-sum problem can be reduced to the counting of the models of a pseudo-Boolean constraint, and we have the result.

□

Proposition 14

Computing the forgetting of a set of variables in a cardinality constraint (resp. pseudo-Boolean constraint) can be done in polynomial time. Said differently, the language 1-CARD (resp. 1-PBC) satisfies FO.

Proof. Let χ be the constraint $\sum_{i=1}^{n} \alpha_i \ell_i + \sum_{i=n+1}^{m} \alpha_i \ell_i \geq \delta$, and let $V = \{\mathsf{var}(\ell_i) | n+1 \leq i \leq m\}$.

It is easy to see that the forgetting of V in χ is equivalent to the disjunction of the two constraints $\sum_{i=1}^{n} \alpha_i \ell_i \geq \delta$ and $\sum_{i=1}^{n} \alpha_i \ell_i \geq \delta - \sum_{i=n+1}^{m} \alpha_i$. Now, observe that the former constraint entails the latter, so that the disjunction is actually equivalent to this latter constraint, and

$$\exists V \chi \equiv \sum_{i=1}^{n} \alpha_i \ell_i \geq \delta - \sum_{i=n+1}^{m} \alpha_i$$

As the result we obtain here is a single pseudo-Boolean constraint that may be obtained in polynomial time, we have the result for 1-PBC. Moreover, if χ is a cardinality constraint, the constraint obtained above is also a cardinality constraint, thus we also have the result for 1-CARD.

□

Corollary 12

Both languages 1-CARD and 1-PBC satisfy SFO.

Proof. The result is immediate, as both languages satisfy FO.

□

Proposition 15

The conjunction of two pseudo-Boolean constraints cannot in general be represented as a pseudo-Boolean constraint. Thus, 1-CARD and 1-PBC do not satisfy $\wedge$BC.

Proof. Let us consider the two clauses (and thus, cardinality and pseudo-Boolean constraints) $a + b \geq 1$ and $\bar{a} + \bar{b} \geq 1$. Let us denote by φ the conjunction of these two constraints. The models of φ are exactly $a \wedge \neg b$ and $\neg a \wedge b$.

Suppose that the pseudo-Boolean constraint χ defined by $\alpha_1 a + \alpha_2 \bar{a} + \beta_1 b + \beta_2 \bar{b} \geq \delta$ is equivalent to φ. Then, by definition, φ and χ must have the same models. We thus have the following inequations:

$$-\alpha_2 - \beta_2 \geq 1 - \delta \tag{1}$$
$$\alpha_2 + \beta_1 \geq \delta \tag{2}$$
$$\alpha_1 + \beta_2 \geq \delta \tag{3}$$
$$-\alpha_1 - \beta_1 \geq 1 - \delta \tag{4}$$

in which, for instance the inequation (1) represents that $\neg a \wedge \neg b$ is not a model of χ, whereas the inequation (2) represents that $\neg a \wedge b$ is a model of χ

Now, observe that, on the one hand, if we add together the two inequations (1) and (2), we get the inequations

$$\beta_1 - \beta_2 \geq 1 \tag{5}$$

while, on the other hand, if we add together the two inequations (3) and (4), we get

$$\beta_2 - \beta_1 \geq 1 \tag{6}$$

If we now add together the two inequations (5) and (6), we get the contradiction $0 \geq 2$. As a consequence, there is no pseudo-Boolean constraint χ representing φ, and in particular no cardinality constraint that represent it either.

$\square$

Corollary 13

Computing the conjunction of cardinality constraints (resp. pseudo-Boolean constraints) is not possible in the language 1-CARD (resp. 1-PBC). Said differently, 1-CARD (resp. 1-PBC) does not satisfy $\wedge$C.

Proof. The result is a direct consequence of Proposition 15.

$\square$

Proposition 16

The disjunction of two pseudo-Boolean constraints cannot in general be represented as a pseudo-Boolean constraint. Thus, 1-CARD and 1-PBC do not satisfy $\vee$BC.

Proof. As 1-CARD (resp. 1-PBC) verifies $\neg$C, but does not satisfy $\wedge$BC, it cannot satisfy $\vee$BC. Otherwise, applying De Morgan's rules would contradict that 1-CARD (resp. 1-PBC) does not satisfy $\wedge$BC.

$\square$

Corollary 14

Computing the disjunction of cardinality constraints (resp. pseudo-Boolean constraints) is not possible in the language 1-CARD (resp. 1-PBC). Said differently, 1-CARD (resp. 1-PBC) does not satisfy $\vee$C.

Proof. The result is a direct consequence of Proposition 16.

$\square$

This corollary ends our study of the queries and transformations offered or not by the formulae composed of a single pseudo-Boolean constraint. We now consider the languages PBC and CARD.

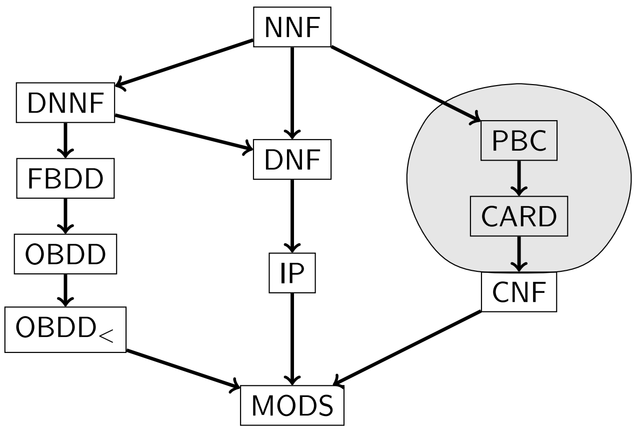


Figure 2.1: Succinctness of various propositional languages. In this diagram, an arrow $\mathsf{L}_1 \to \mathsf{L}_2$ means that L_1 is strictly more succinct than L_2, i.e., $\mathsf{L}_1 \leq_s \mathsf{L}_2$ and $\mathsf{L}_2 \not\leq_s \mathsf{L}_1$. No arrow between two languages means that they are incomparable. The grayed area highlights the results we claim in this section (including incomparability).

2.2 Succinctness of Pseudo-Boolean Constraints

In this section, we study the relative succinctness of pseudo-Boolean languages with many other languages from the knowledge compilation map [DM02]. For this purpose, let us introduce the languages PBC and CARD.

Definition 67 (PBC, CARD)

PBC (resp. CARD) is the language of pseudo-Boolean formulae composed of a (finite) conjunction of normalized pseudo-Boolean constraints (resp. cardinality constraints).

In the following PBC and CARD are in particular compared to NNF, CNF, DNF, IP, DNNF, FBDD, OBDD, OBDD$_<$, and MODS (see Subsection 1.3.1 for a description of these languages). The results we obtain are summarized by the diagram in Figure 2.1. The proofs of these results are given below.

Proposition 17

PBC is strictly more succinct than CARD, i.e., PBC $<_s$ CARD.

Proof. First, since CARD is a subset of PBC, we get that PBC $\leq_s$ CARD.

Now, let us prove that CARD $\not\leq_s$ PBC. Let χ be the pseudo-Boolean constraint defined by $\delta x + \sum_{i=1}^{2\delta} x_i \geq \delta$. Let κ be the (non-valid) cardinality constraint $\sum_{i=1}^{n} \ell_i \geq \delta'$ such that $\chi \models \kappa$, and $\mathsf{var}(\kappa) \subseteq \mathsf{var}(\chi)$. We first show that, necessarily, $\delta' = 1$, i.e., κ is a clause.

Let us suppose that $\delta' > 1$, and let us consider M the model of χ such that x is satisfied, and all literals (except x) in κ are falsified. Under M, we necessarily have $\sum_{i=1}^{n} \ell_i \leq 1$, since only x is satisfied by construction. Indeed, if x positively appears in κ, then $\sum_{i=1}^{n} \ell_i = 1$. Otherwise, x appears negatively or does not appear in κ, and we have $\sum_{i=1}^{n} \ell_i = 0$. In both cases, since we have assumed $\delta' > 1$, κ is not satisfied, so M is not a model of κ. This contradicts that $\chi \models \kappa$, so $\delta' \leq 1$. Since κ is supposed to be non-valid, δ' cannot be equal to 0, so $\delta' = 1$.

The only way to represent χ as a conjunction of cardinality constraints is thus to use clauses, since all these constraints must be implied by χ. However, by Proposition 3, we know that an exponential number of clauses is required to represent χ, which completes the proof.

□

Proposition 18

NNF is more succinct than PBC, i.e., NNF $\leq_s$ PBC.

Proof. It is well known that any pseudo-Boolean constraint can be represented by an equivalent NNF representation of polynomial size (see, e.g., [Vol99]). Now, when a conjunction of pseudo-Boolean constraints is considered, an NNF representation of this conjunction consists of the conjunction of all these "elementary" NNF representations – one per pseudo-Boolean constraint – which produces an NNF formula. Once again, the result has a polynomial size.

□

Corollary 15

NNF is more succinct than CARD, i.e., NNF $\leq_s$ CARD.

Proof. The result is a direct consequence of Proposition 18, as CARD is a sublanguage of PBC.

□

Proposition 19

CARD is strictly more succinct than CNF, i.e., CARD $<_s$ CNF.

Proof. The results is a direct consequence of Proposition 3.

□

Corollary 16

PBC is strictly more succinct than CNF, i.e., PBC $<_s$ CNF.

Proof. The result is a direct consequence of Proposition 19, as CARD is a sublanguage of PBC.

□

Corollary 17

CARD and PBC are strictly more succinct than PI and MODS, i.e.,

- CARD $<_s$ PI and PBC $<_s$ PI
- CARD $<_s$ MODS and PBC $<_s$ MODS

Proof. On the one hand, we have by Proposition 19 that CARD $<_s$ CNF. On the other hand, we have by [DM02] that CNF $<_s$ PI and CNF $<_s$ MODS. By transitivity of $<_s$, the corollary follows. A similar argument applies to PBC $<_s$ PI and PBC $<_s$ MODS

□

Corollary 18

The languages DNNF, d-DNNF, sd-DNNF, FBDD, OBDD, OBDD$_<$, DNF and IP are not at least as succinct as CARD and PBC, i.e.,

- DNNF $\not\leq_s$ CARD and DNNF $\not\leq_s$ PBC
- d-DNNF $\not\leq_s$ CARD and d-DNNF $\not\leq_s$ PBC
- sd-DNNF $\not\leq_s$ CARD and sd-DNNF $\not\leq_s$ PBC
- FBDD $\not\leq_s$ CARD and FBDD $\not\leq_s$ PBC
- OBDD $\not\leq_s$ CARD and OBDD $\not\leq_s$ PBC
- OBDD$_<$ $\not\leq_s$ CARD and OBDD$_<$ $\not\leq_s$ PBC
- DNF $\not\leq_s$ CARD and DNF $\not\leq_s$ PBC
- IP $\not\leq_s$ CARD and IP $\not\leq_s$ PBC

Proof. We only prove the result for DNNF $\not\leq_s$ CARD. A similar proof can be made for all the other languages.

Let us suppose that DNNF $\leq_s$ CARD. By Proposition 19, we have that CARD $<_s$ CNF. By transitivity of $\leq_s$, we would then have that DNNF $<_s$ CNF. This would contradict the fact that DNNF $\not\leq_s$ CNF [BCMS16]. Hence, DNNF $\not\leq_s$ CARD.

□

Proposition 20

PBC is not at least as succinct as OBDD$_<$, i.e., PBC $\not\leq_s$ OBDD$_<$.

Proof. Let φ the formula defined as $\bigoplus_{i=1}^{n} x_i$. Then φ is true if and only if there is an odd number of satisfied x_i. This formula can be written as an OBDD$_<$ representation of polynomial size [Bry86]. Let us prove that every representation of φ as a conjunction of pseudo-Boolean constraints requires an exponential number of constraints.

Let χ be the non-valid pseudo-Boolean constraint $\sum_{i=1}^{m} \alpha_i \ell_i \geq \delta$ such that $\varphi \models \chi$ and $\mathsf{var}(\chi) \subseteq \mathsf{var}(\varphi)$. Let us denote by $\mathcal{L}^+$ and $\mathcal{L}^-$ the sets of positive and negative literals of χ, respectively.

Claim 1. *We have* $\mathsf{var}(\chi) = \mathsf{var}(\varphi)$.

Proof. If there is $x \in \mathsf{var}(\varphi)$ such that $x \notin \mathsf{var}(\chi)$, then any counter-model of χ can be turned into a model M of φ by keeping the same assignment, except for x, which must be satisfied. M is still a counter-model of χ, which contradicts that $\varphi \models \chi$.

□

Claim 2. *$\mathcal{L}^-$ contains an even number of literals.*

Proof. Let us suppose that $\mathcal{L}^-$ contains an odd number of literals. Let M be the interpretation falsifying all the literals appearing in χ. Clearly, M is not a model of χ, since this constraint is supposed to be non-valid. However, M satisfies all x_i such that $\bar{x}_i \in \mathcal{L}^-$. As there is an odd number of such x_i, M is a model of φ. This contradicts that $\varphi \models \chi$, so $\mathcal{L}^-$ contains an even number of literals.

□

Claim 3. *For all $i \in \{1, \dots, n\}$, we have that $\alpha_i = \delta$.*

Proof. Without loss of generality, we can suppose that $\alpha_i \leq \delta$ for all $i \in \{1, \dots, n\}$. Indeed, if for some i, we have $\alpha_i > \delta$, it is easy to see that we can replace it by δ while preserving equivalence (this rule is known as the *saturation* rule).

Now, suppose that there exists $i_0 \in \{1, \dots, m\}$, such that $\alpha_{i_0} < \delta$ and that the variable associated with ℓ_{i_0} is x. There are two possible cases: either $\ell_{i_0} \in \mathcal{L}^+$ or $\ell_{i_0} \in \mathcal{L}^-$. If $\ell_{i_0} \in \mathcal{L}^+$, let us consider the interpretation M satisfying x and all x_i such that $\bar{x}_i \in \mathcal{L}^-$. By Claim 2, M satisfies thus an odd number of variables, so it is a model of φ. Otherwise, $\ell_{i_0} \in \mathcal{L}^-$, and let us consider the interpretation M satisfying all x_i such that $\bar{x}_i \in \mathcal{L}^- \setminus \{\ell_{i_0}\}$. By Claim 2, M satisfies once again an odd number of variables, so it is a model of φ.

In both cases, $\sum_{i=1}^{n} \alpha_i M(\ell_i) = \alpha_{i_0} < \delta$, so M is not a model of χ. This contradicts that $\varphi \models \chi$, and the claim follows.

□

So, χ has the form $\sum_{i=1}^{n} \delta \ell_i \geq \delta$, which is clearly equivalent to the clause $\sum_{i=1}^{n} \ell_i \geq 1$. Then, the only way to represent φ as a conjunction of pseudo-Boolean constraints is to use clauses, since all these constraints must be implied by φ. Since φ requires an exponential number of clauses to be represented without introducing new variables, the proposition follows.

□

Corollary 19

CARD is not at least as succinct as OBDD$_<$, i.e., CARD $\not\leq_s$ OBDD$_<$.

Proof. Towards a contradiction let us suppose that CARD $\leq_s$ OBDD$_<$. By Proposition 17, we have that PBC $<_s$ CARD. By transitivity of $\leq_s$, we would then have PBC $\leq_s$ OBDD$_<$, which contradicts Proposition 20.

□

Corollary 20

CARD and PBC are not at least as succinct as the languages NNF, DNNF, d-DNNF, sd-DNNF, FBDD and OBDD, i.e.,

- CARD $\not\leq_s$ NNF and PBC $\not\leq_s$ NNF
- CARD $\not\leq_s$ DNNF and PBC $\not\leq_s$ DNNF
- CARD $\not\leq_s$ d-DNNF and PBC $\not\leq_s$ d-DNNF
- CARD $\not\leq_s$ sd-DNNF and PBC $\not\leq_s$ sd-DNNF
- CARD $\not\leq_s$ FBDD and PBC $\not\leq_s$ FBDD
- CARD $\not\leq_s$ OBDD and PBC $\not\leq_s$ OBDD

Proof. We only prove the result for PBC $\not\leq_s$ NNF. A similar proof can be given for all the other languages.

Suppose that PBC $\leq_s$ NNF. By [DM02], we have that NNF $\leq_s$ OBDD$_<$. By transitivity of $\leq_s$, we would then have that PBC $\leq_s$ OBDD$_<$, which contradicts Proposition 20. Hence, PBC $\not\leq_s$ NNF.

□

Proposition 21

PBC is not at least as succinct as IP, i.e., PBC $\not\leq_s$ IP.

Proof. Let φ the formula from IP defined as $\bigvee_{i=1}^{n}(x_i \wedge y_i)$ [DM02]. Let us show that representing φ as a conjunction of pseudo-Boolean constraints requires an exponential number of constraints. Let χ be the non-valid pseudo-Boolean constraint $\sum_{i=1}^{n} \alpha_i \ell_i \geq \delta$ such that $\varphi \models \chi$, with $\mathsf{var}(\chi) \subseteq \mathsf{var}(\varphi)$.

Claim 4. *For all $i \in \{1, \dots, n\}$, x_i or y_i appears positively in χ.*

Proof. If neither x_i nor y_i appear in $\mathsf{var}(\chi)$, then the model of φ satisfying both x_i and y_i but falsifying all the literals of χ is not a model of χ, which contradicts $\varphi \models \chi$.

□

Claim 5. *For all $i \in \{1, \dots, n\}$, if x_i appears positively in χ, and y_i does not, then the weight of x_i is δ. Symmetrically, for all $i \in \{1, \dots, n\}$, if y_i appears positively in χ, and x_i does not, then the weight of y_i is δ.*

Proof. If only x_i appears positively in χ, say $x_i = \ell_{i_0}$, and $\alpha_{i_0} < \delta$, then the model of φ satisfying both x_i and y_i but falsifying all the other literals of χ is not a model of χ, which contradicts $\varphi \models \chi$. The proof for the case when only y_i appears positively in χ is similar.

□

Claim 6. *For all $i \in \{1, \dots, n\}$, if x_i and y_i appear positively in χ, then the sum of their weights is at least equal to δ.*

Proof. If both x_i and y_i appear positively as ℓ_{i_0} and l_{i_1} in χ, respectively, and $\alpha_{i_0} + \alpha_{i_1} < \delta$, then the model of φ satisfying these two literals but falsifying all the others of χ is not a model of χ, which contradicts $\varphi \models \chi$.

□

Claim 7. *Without loss of generality, χ contains only positive literals.*

Proof. If $\bar{y}_i$ appears in χ, let χ' be the pseudo-Boolean constraint obtained by removing the literal $\bar{y}_i$ from χ. Any model M of φ is a model of χ'. Indeed, if M satisfies y_i, then it is obviously a model of χ'. Otherwise, there exists $i_0 \neq i$ such that M satisfies x_{i_0} and y_{i_0}. By Claims 4, 5 and 6, χ' is also satisfied by M. So, $\varphi \models \chi'$, and since we obviously have that $\chi' \models \chi$, it is possible to replace χ by χ' without losing information. All negative literals can be removed by repeating the same argument. The case when $\bar{x}_i$ appears in χ follows symmetrically.

□

Now, let K be the pseudo-Boolean formula $\bigwedge_{j=1}^{m} \chi_j$ such that $\varphi \equiv K$. In particular, $\varphi \models \chi_j$ for each $j \in \{1, \dots, m\}$. From Claim 7, we have that each χ_j has the form $\sum_{i=1}^{n} \alpha_{0,i,j} x_i + \alpha_{1,i,j} y_i \geq \delta_j$, with, for all $i \in \{1, \dots, n\}$, $\alpha_{0,i,j} + \alpha_{1,i,j} \geq \delta_j$ by applying Claim 6.

Any counter-model of φ must be a counter-model of K. It remains to show, in the rest of this proof, that a subset of these counter-models requires the use of an exponential number of pseudo-Boolean constraints to get a formula logically equivalent to φ.

Any interpretation satisfying for all $i \in \{1, \dots, n\}$ exactly one of x_i or y_i cannot be a model of K, since it is not a model of φ. Then, it must not satisfy at least one of the constraints χ_j. Formally, this means that the following property must be satisfied.

Property 1. *For any function $f : \{1, \dots, n\} \to \{0, 1\}$, there exists $j \in \{1, \dots, m\}$ such that $\sum_{i=1}^{n} \alpha_{f(i),i,j} < \delta_j$.*

For each of the inequalities induced by Property 1, we define a tuple $(f(1), \dots, f(n)) \in \mathbb{B}^n$. We claim that two inequalities associated with such tuples which have a Hamming distance at least equal to 2 cannot be satisfied for a same j. To see this, let $(f_1(1), \dots, f_1(n))$ and $(f_2(1), \dots, f_2(n))$ be two tuples with a Hamming distance at least 2 for a same j. Their associated inequalities are $\sum_{i=1}^{n} \alpha_{f_1(i),i,j} < \delta_j$ (1) and $\sum_{i=1}^{n} \alpha_{f_2(i),i,j} < \delta_j$ (2).

Since the Hamming distance between the two tuples is at least 2, there exists $i_0, i_1 \in \{1, \dots, n\}$, with $i_0 \neq i_1$, such that $f_1(i_0) \neq f_2(i_0)$ and $f_1(i_1) \neq f_2(i_1)$. By adding (1) and (2), since f_1 and f_2 take their values in $\{0, 1\}$, we get:

$$\sum_{\substack{i=1 \\ i \notin \{i_0, i_1\}}}^{n} (\alpha_{f_1(i),i,j} + \alpha_{f_2(i),i,j}) + \alpha_{0,i_0,j} + \alpha_{1,i_0,j} + \alpha_{0,i_1,j} + \alpha_{1,i_1,j} < 2\delta_j$$

However, this is impossible since Claim 6 gives us $\alpha_{0,i_0,j} + \alpha_{1,i_0,j} \geq \delta_j$ and $\alpha_{0,i_1,j} + \alpha_{1,i_1,j} \geq \delta_j$, so $\alpha_{0,i_0,j} + \alpha_{1,i_0,j} + \alpha_{0,i_1,j} + \alpha_{1,i_1,j} \geq 2\delta_j$. We thus need two distinct constraints to satisfy the inequalities of Property 1 associated with these tuples.

Claim 8. *There exists a set of 2^{n-1} tuples of $\mathbb{B}^n$ having pairwise Hamming distance at least 2.*

Proof. Consider the set S of solutions over $\{0, 1\}$ of $\sum_{i=1}^{n} z_i \equiv 0 \bmod 2$. The elements of S have pairwise Hamming distance at least 2. Finally, S contains 2^{n-1} elements since any assignment to $z_1, \dots, z_{n-1}$ can be extended to a solution in S.

□

Hence, to eliminate all counter-models of φ, at least 2^{n-1} distinct constraints must be used in its representation in PBC (those associated with the tuples of Claim 8). So, representing φ as a conjunction of pseudo-Boolean constraints requires exponentially many constraints.

□

Corollary 21

CARD is not at least as succinct as IP, i.e., CARD $\not\leq_s$ IP.

Proof. Towards a contradiction let us suppose that CARD $\leq_s$ IP. By Proposition 17, we have that PBC $<_s$ CARD. By transitivity of $\leq_s$, we would then have PBC $\leq_s$ IP, which contradicts Proposition 21.

□

Corollary 22

CARD and PBC are not at least as succinct as DNF, i.e., CARD $\not\leq_s$ DNF and PBC $\not\leq_s$ DNF.

Proof. The result is a direct consequence of Proposition 21, as IP is a sublanguage of DNF.

□

In terms of succinctness, the main difference between the pseudo-Boolean languages and CNF is that PBC and CARD are strictly more succinct than CNF. Compared to the other languages, there is not a great difference. Let us now consider the queries and transformations that are offered by these languages.

2.3 Querying and Transforming Pseudo-Boolean Constraints

We now present the results about the queries and transformations offered by PBC and CARD, summarized in Tables 2.3 and 2.4.

	CO	VA	CE	IM	EQ	SE	CT	ME
CNF	○	✓	○	✓	○	○	○	○
CARD	○	✓	○	✓	○	○	○	○
PBC	○	✓	○	✓	○	○	○	○

Table 2.3: Properties of CARD and PBC in terms of queries, compared to CNF. A ✓ means that the query is offered by the language in polynomial time, and a ○ means that it is not the case, unless P $=$ NP.

	CD	FO	SFO	$\wedge$C	$\wedge$BC	$\vee$C	$\vee$BC	$\neg$C
CNF	✓	○	✓	✓	✓	●	✓	●
CARD	✓	●	●	✓	✓	●	●	●
PBC	✓	●	●	✓	✓	●	●	●

Table 2.4: Properties of CARD and PBC concerning transformations, compared to CNF. A ✓ means that the language offers the transformation in polynomial time, whereas a ○ means that it does not unless P $=$ NP and a ● means that it does not unconditionally.

Let us now prove the results of these tables.

Proposition 22

CARD does not satisfy any of CO, CE, EQ, SE, CT and ME, unless P = NP.

Proof. Since the translation of any CNF formula into a conjunction of cardinality constraints can be done in polynomial time, and since CNF does not satisfy any of these properties unless P = NP, the proposition follows.

□

Corollary 23

PBC does not satisfy CO, CE, EQ, SE, CT and ME, unless P = NP.

Proof. The result is an immediate consequence of Proposition 22, as CARD is a sublanguage of PBC.

□

Proposition 23

CARD and PBC satisfy VA.

Proof. A conjunction of pseudo-Boolean constraints is valid if and only if each of its constraint is valid, which can be checked in polynomial time by Corollary 6.

□

Proposition 24

CARD and PBC satisfy CD.

Proof. Computing the conditioning of any formula from PBC (resp. CARD) is easily done by applying it on each constraint of the formula (see Proposition 6).

□

Corollary 24

CARD and PBC satisfy IM.

Proof. As CARD and PBC satisfy both VA (Proposition 23) and CD (Proposition 24), the corollary follows immediately by Lemma A.7 from [DM02].

□

Proposition 25

CARD and PBC satisfy both $\wedge$BC and $\wedge$C.

Proof. Given a conjunctively-interpreted set of formulae from CARD (resp. PBC), computing the conjunction of these formulae can trivially be done in polynomial time, by taking the union of the considered sets of constraints. □

Proposition 26

CARD does not satisfy $\vee$BC.

Proof. Let φ be the disjunction of the two cardinality constraints $\kappa = y \geq 1$ (i.e., $\kappa \equiv y$) and $\kappa' = \sum_{i=1}^{2\delta} x_i \geq \delta$. It is easy to see that φ is equivalent to $\delta y + \sum_{i=1}^{2\delta} x_i \geq \delta$. As shown in the proof of Proposition 17, a representation of this constraint using only cardinality constraints requires exponentially many constraints. Hence, the claim follows. □

Proposition 27

PBC does not satisfy $\vee$BC.

Proof. To prove this result, we show that any representation of the inequality $\Delta = \sum_{i=1}^{2\delta} x_i \neq \delta$, which is equivalent to the disjunction of $\sum_{i=1}^{2\delta} x_i \geq \delta+1$ and $\sum_{i=1}^{2\delta} \bar{x}_i \geq \delta+1$, requires an exponential number of pseudo-Boolean constraints when $\delta \geq 2$.

Let us consider a non-valid pseudo-Boolean constraint $\chi = \sum_{i=1}^{m} \alpha_i \ell_i \geq \nu$, such that $\Delta \models \chi$, with $\mathsf{var}(\chi) \subseteq \mathsf{var}(\Delta)$. Let us note $\mathcal{L}^+$ and $\mathcal{L}^-$ the sets of positive and negative literals of χ, respectively.

Claim 9. *We have* $\mathsf{var}(\chi) = \mathsf{var}(\Delta)$.

Proof. Consider a counter-model I of χ. Since $\Delta \models \chi$, I satisfies δ of the x_i. If $x \in \mathsf{var}(\Delta)$ but $x \notin \mathsf{var}(\chi)$, then the interpretation I' defined as satisfying exactly the same literals as I and satisfying also x satisfies $\delta + 1$ of the x_i, so it is a model of Δ, but not a model of χ, which contradicts $\Delta \models \chi$. □

Claim 10. *The set* $\mathcal{L}^+$ *and* $\mathcal{L}^-$ *contain the same number of literals, i.e.,* χ *contains exactly* δ *positive literals and* δ *negative literals.*

Proof. If $\mathcal{L}^-$ contains strictly more literals than $\mathcal{L}^+$, then by Claim 9, $\mathcal{L}^-$ contains necessarily strictly more than δ literals. The interpretation satisfying exactly each x_i such that $\bar{x}_i \in \mathcal{L}^-$ satisfies strictly more than δ of the x_i, so it is a model of Δ, but not a model of χ as $\nu > 0$.

Otherwise, if $\mathcal{L}^+$ contains strictly more literals than $\mathcal{L}^-$, then by Claim 9, $\mathcal{L}^+$ contains necessarily strictly more than δ literals. The interpretation satisfying exactly each x_i such that $\bar{x}_i \in \mathcal{L}^-$ satisfies strictly less than δ of the x_i, so it is a model of Δ, but not a model of χ.

In both cases, $\Delta \not\models \chi$, so $\mathcal{L}^+$ and $\mathcal{L}^-$ contain the same number of literals. They necessarily contain δ literals each, by Claim 9.

□

Claim 11. *We have, for all $i \in \{1, \dots, m\}$, $\alpha_i = \delta$.*

Proof. Let us suppose that there exists $i_0 \in \{1, \dots, m\}$, such that $\alpha_{i_0} < \delta$ and that the variable associated with ℓ_{i_0} is x. There are two possible cases: either $\ell_{i_0} \in \mathcal{L}^+$ or $\ell_{i_0} \in \mathcal{L}^-$.

If $\ell_{i_0} \in \mathcal{L}^+$, let us consider the interpretation M satisfying exactly all the x_i such that $\bar{x}_i \in \mathcal{L}^-$ and x. Then, by Claims 9 and 10, M satisfies $(n+1)$ of the x_i, so it is a model of Δ.

Otherwise, $\ell_{i_0} \in \mathcal{L}^-$ and let us consider the interpretation M satisfying exactly all the x_i such that $\bar{x}_i \in \mathcal{L}^-$, except x. Then, M satisfies $(n-1)$ of the x_i, so it is a model of Δ.

In both cases, under M, we have $\sum_{i=1}^{m} \alpha_i \ell_i = \alpha_{i_0} < \delta$, so M is not a model of χ. $\Delta \models \chi$ is contradicted, so for all $i \in \{1, \dots, m\}$, we have $\alpha_i = \delta$.

□

Thus, χ has the form $\sum_{i=1}^{2\delta} \delta \ell_i \geq \delta$, which is equivalent to the clause $\sum_{i=1}^{2\delta} \ell_i \geq 1$ The only way to represent Δ as a conjunction of pseudo-Boolean constraints is then to use clauses, since all the constraints must be implied by Δ. However, Δ requires an exponential number of clauses to be represented without introducing any new variable. Indeed, a clause as those we have produced in this proof can only eliminate a single counter-model of Δ, since all those clauses contain all the variables, and there exist $\binom{2n}{n}$ counter-models. This concludes the proof.

□

Corollary 25

CARD and PBC do not satisfy $\vee$C.

Proof. As CARD and PBC do not satisfy $\vee$BC by Propositions 26 and 27, respectively, they cannot satisfy $\vee$C.

□

Corollary 26

CARD and PBC do not satisfy $\neg$C.

Proof. Since both languages satisfy $\wedge$C (Proposition 25) and do not satisfy $\vee$C, they cannot satisfy $\neg$C. Otherwise, De Morgan's rules would allow to compute $\vee$C in polynomial time.

□

Proposition 28

CARD does not satisfy SFO.

Proof. [3] Let $\delta \geq 2$ and $n = 2(\delta - 1)$. Let κ and κ' be the two formulae defined by the cardinality constraints $\sum_{i=1}^{n} x_i + y + z \geq \delta$ and $\sum_{i=n+1}^{2n} x_i + y + \bar{z} \geq \delta$, respectively. Observe that only the variables y and z are common to both constraints. Let us denote by φ the formula obtained by forgetting z in the conjunction $\kappa \wedge \kappa'$, i.e., $\varphi \equiv \exists z(\kappa \wedge \kappa')$. Let us prove that any conjunction of cardinality constraints representing φ requires exponentially many constraints.

First, observe that, by definition, $\varphi \equiv ((\kappa \wedge \kappa')|z) \vee ((\kappa \wedge \kappa')|\bar{z})$, so any model of φ is also a model of the three formulae:

- $\varphi_1 = \sum_{i=1}^{n} x_i + y \geq \delta - 1$
- $\varphi_2 = \sum_{i=n+1}^{2n} x_i + y \geq \delta - 1$
- $\varphi_3 = \sum_{i=1}^{n} x_i + y \geq \delta \vee \sum_{i=n+1}^{2n} x_i + y \geq \delta$

Let I be an interpretation of the variables $x_1, \ldots, x_{2n}$ and y such that $\sum_{i=1}^{n} x_i = \delta - 1$ and $\sum_{i=n+1}^{2n} x_i = \delta - 1$ are satisfied by I, while y is falsified by I. Observe that I is a counter-model of φ, as it does not satisfy φ_3.

Now, consider the clause γ defined by $\sum_{I(x_i)=0} x_i + y \geq 1$, which is obviously falsified by I by construction (it contains exactly all the literals that are falsified by I). Let us prove that every conjunction of cardinality constraints representing φ contains γ. We have in particular the following claim.

Claim 12. *The clause γ is entailed by φ.*

Proof. Let M be a model of φ. There are two cases: either M satisfies y or it does not. If M satisfies y, then it trivially also satisfies γ. Otherwise, M must satisfy the three formulae φ_1, φ_2 and φ_3. In particular, as y is falsified by M, we have that M must satisfy at least $2\delta - 1$ of the x_i. Now, observe that, γ contains exactly $2\delta - 2$ of the x_i. As there are $4\delta - 4$ such x_i, it follows that M must necessarily satisfy one of those appearing in γ. Thus, M is a model of γ. As the same applies to any model of φ, the claim follows. □

Let K be a conjunction of cardinality constraints representing φ. As φ is falsified by I, we have that K contains a constraint that is both entailed by φ and falsified by I. This constraint κ_0 has the form $\sum_{x \in X} x \geq k$, where $X \subseteq \{x_1, \ldots, x_{2n}\} \cup \{y\}$.

Claim 13. *The set X contains exactly the variables that are falsified by I.*

Proof. Towards a contradiction, suppose that there exists a variable v such that v is falsified by I and $v \notin X$. The interpretation obtained from I by changing the assignment of v so that this variable is satisfied does not satisfy κ_0, as v does not appear in this constraint. However, it is a model of φ, (see the proof of Claim 12 above), which contradicts that κ_0 is entailed by φ. Hence, all the variables falsified by I appear in κ_0.

Symmetrically, assume that there exists a variable v such that v is satisfied by I and $v \in X$. Let I' be the interpretation such that $I'(y) = 1$, $I'(v) = 0$ and $I'(x_i) = I(x_i)$ for any $x_i \neq v$. Note that the left-hand side of κ_0 has the same value for I and I'. Indeed, κ_0 contains both v and y (as y is falsified by I), and their values are simply switched between I and I'. As I does not satisfy κ_0, neither does I'. However, I' is a model of φ (see the proof of Claim 12 above), which contradicts the fact that κ_0 is entailed by φ. Hence, X does not contain any variable satisfied by I.

Altogether, X contains exactly the variables satisfied by I. □

[3] Many thanks to Petr Savicky for this proof.

Additionally, we show now that $k = 1$ is the only value of k for which the constraint κ_0 is entailed by φ.

Claim 14. *The constraint κ_0 is a clause.*

Proof. Towards a contradiction, suppose that $k > 1$. Consider the interpretation M of φ satisfying exactly all x_i such that $x_i \notin X$ and y. Then, M is a model of φ (see the proof of Claim 12 above), but not a model of κ_0, as it only satisfies one literal from this constraint, which contradicts κ_0 being entailed by φ.

$\square$

This reasoning pattern can be applied to any interpretation I as described above. As there are $\binom{n}{\delta-1}^2 = \binom{n}{n/2}^2$, such interpretations, and since for each of them, the clause γ is different, any representation of φ containing only cardinality constraints must contain an exponential number of clauses, hence the proposition follows.

$\square$

Proposition 29

PBC does not satisfy SFO.

Proof. Let χ and χ' be the two pseudo-Boolean constraints defined by $\sum_{i=1}^{2\delta} x_i \geq \delta + 1$ and $\sum_{i=1}^{2\delta} \bar{x}_i \geq \delta + 1$, respectively. Let us consider, for s a newly introduced variable, the formulae χ_s defined by $(\delta + 1)s + \sum_{i=1}^{2\delta} x_i \geq \delta + 1$ and χ'_s defined by $(\delta + 1)\bar{s} + \sum_{i=1}^{2\delta} \bar{x}_i \geq \delta + 1$ which are obtained from χ and χ' in polynomial time.

Let φ be the conjunction $\chi_s \wedge \chi'_s$. By construction, $\varphi|s \equiv \chi'$ and $\varphi|\bar{s} \equiv \chi$, so that $\exists x \varphi \equiv \chi \vee \chi'$. If an algorithm existed to forget a single variable in a set of pseudo-Boolean constraints, then one could use it to compute in polynomial time the disjunction of χ and χ'. However, we have proven that it is not possible (cf. Proposition 27). Then, so it is for the forgetting of a single variable in a conjunction of pseudo-Boolean constraints.

$\square$

Corollary 27

CARD and PBC do not satisfy FO.

Proof. As CARD and PBC do not satisfy SFO by Propositions 28 and 29, respectively, they cannot satisfy FO.

$\square$

Chapter 3

Graph Width Measures for CNF Encodings with Auxiliary Variables

As discussed in Chapter 2, pseudo-Boolean constraints do not bring advantages compared to CNF from a knowledge representation perspective, except for their succinctness. In particular, we do not gain any query or transformation, and even lose some that are tractable when given a CNF formula as input. In this context, and as mentioned in Section 1.1.3, it may be more convenient, depending on the needs, to use a CNF *encoding* for the considered pseudo-Boolean formula, as it may often be smaller in practice than its CNF *representation*, which has to be equivalent to the original formula.

In the context of propositional satisfiability, we often consider graphs associated with the CNF formulae we study, such as their primal or incidence graphs. One often considers different width measures of these graphs, such as treewidth and cliquewidth: if the corresponding width is small, there are algorithms that solve SAT, but also more complex problems like #SAT or MAX-SAT or even QBF efficiently [SS10a, FMR08, SS13, PSS16, STV15, Che04]. There is also a considerable body of work on reasoning problems from artificial intelligence restricted to knowledge encoded by CNF formulae with restricted underlying graphs: for example, treewidth restrictions have been studied for abduction, closed-world reasoning, circumscription, disjunctive logic programming [GPW10] and answer set programming [JPW09].

Curiously, however, there seems to be very little work on the natural question of what we can actually encode with these restricted CNF formulae. This question is pertinent because good algorithms for problems are less attractive if they cannot deal with interesting instances. This chapter provides an answer to this question, based on recent machinery developed in the area of knowledge compilation. In particular, we use a combination of the algorithm proposed by [BCMS15], the width notion for DNNF developed by [CM19] and the lower bound techniques introduced by [PD10] and [BCMS16].

Specifically, we show lower bounds on the size or width of representations for Boolean functions, by taking advantage of *communication complexity*. Actually, the area has been partially developed for this purpose, and there is a large literature on this [KN97, Hro97, Juk12]. In particular, there are many results for showing lower bounds on different forms of branching programs by means of communication complexity [Weg00, DHJ$^+$04]. More recently, this approach has been generalized to more general languages considered in knowledge compilation [PD10, BCMS16]. However, beyond the lower bounds on treewidth already discussed in [BKM11], we are not aware of any use of communication complexity to show bounds on width measures of CNF formulae.

3.1 Preliminaries

Let us introduce some preliminaries on graphs associated with CNF formulae, graph width measures, communication complexity and structured deterministic DNNF.

3.1.1 Graphs Associated to CNF Formulae

In this chapter, we focus on CNF *representations* (see Definition 24) and CNF *encodings* (see Definition 25) of Boolean functions. We recall that the main difference between the two representations is that the latter allows the introduction of *auxiliary variables* that do not appear in the former. More precisely, given a Boolean function f, every assignment accepted by f is a model of a CNF representation φ of f and can be extended to a model of a CNF encoding ψ of f by assigning its auxiliary variables. On the contrary, every assignment rejected by f is a counter-model of φ and cannot be extended to a model of ψ. Note that the definition of CNF encoding does not indicate how to extend the considered assignments, and it may in general exist multiple possible extensions, unless auxiliary variables are *dependent*.

Definition 68 (Dependent Auxiliary Variable [GMT02])

Given a CNF encoding ψ of a Boolean function f, an auxiliary variable v of ψ is called *dependent* if and only if for every model M of f, all extensions M' of M satisfying ψ take the same value on v.
We say that ψ has *dependent auxiliary variables* if all its auxiliary variables are dependent. Note that for such an encoding the extension M' is unique.

CNF formulae are often assigned two graphs that represent their structure, namely the *primal graph* and the *incidence graph*. To define these graphs, we use standard notations from graph theory and assume the reader to have a basic background in the area [Die12].

Definition 69 (Primal Graph of a CNF Formula)

The *primal graph* of a CNF formula φ has as vertices the variables of φ and two variables v and v' are connected by an edge if and only if there is a clause γ such that $v \in \mathsf{var}(\gamma)$ and $v' \in \mathsf{var}(\gamma)$.

Definition 70 (Incidence Graph of a CNF Formula)

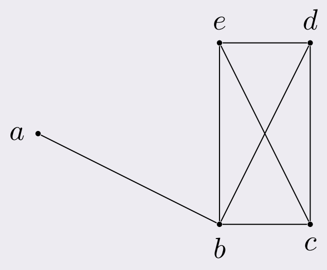
The *incidence graph* of a CNF formula φ has as vertex set the union of the variable set and the clause set of φ. Edges in the incidence graph are exactly the pairs (v, γ) such that v is a variable and γ is a clause such that $v \in \mathsf{var}(\gamma)$.

Example 32

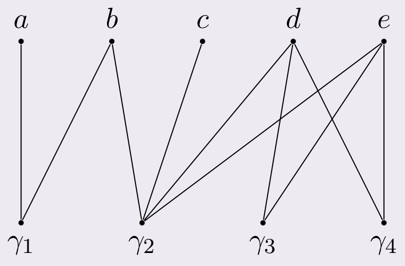
Let us consider the following clauses:

- $\gamma_1 = a \vee \neg b$
- $\gamma_2 = b \vee c \vee \neg d \vee \neg e$
- $\gamma_3 = \neg d \vee e$
- $\gamma_4 = d \vee e$

and let the CNF formula φ be defined as $\varphi = \gamma_1 \wedge \gamma_2 \wedge \gamma_3 \wedge \gamma_4$. Its primal graph is given by:



while the incidence graph of φ is given by:



3.1.2 Graph Width Measures

In this section, we introduce several graph width measures we consider in the rest of this chapter. Many of those measures are based on the notion of *tree decomposition*.

Definition 71 (Tree Decomposition)

A *tree decomposition* $(T, (B_t)_{t \in \mathcal{V}(T)})$ of a graph $G = (V, E)$ consists of a tree T and, for every node t of T, a set $B_t \subseteq V$ called *bag* such that:

- $\bigcup_{t \in \mathcal{V}(T)} B_t = V$,
- for every edge $(u, v) \in E$, there is a bag B_t such that $\{u, v\} \subseteq B_t$, and
- for every $v \in V$, the set $\{t \in \mathcal{V}(T) \mid v \in B_t\}$ is connected in T.

Definition 72 (Treewidth)

The *width* of a tree decomposition $(T, (B_t)_{t \in \mathcal{V}(T)})$ is defined as $\max\{|B_t| \mid t \in \mathcal{V}(T)\} - 1$. The *treewidth* $\mathsf{tw}(G)$ of a graph G is defined as the minimum width taken over all tree decompositions of G.

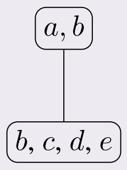
Remark 14

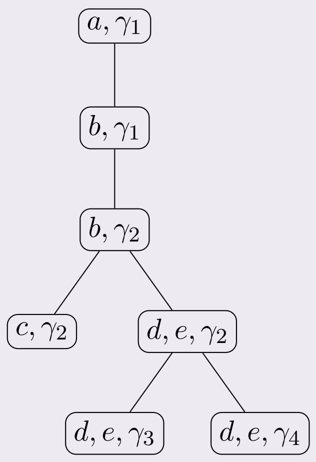
Intuitively, the treewidth of a graph G estimates how close is G to being a tree. In particular, trees are exactly the connected graphs of treewidth 1.

For every CNF formula φ, we define the *primal treewidth* $\mathsf{tw}_\mathsf{p}(\varphi)$ of φ as the treewidth of its primal graph and the *incidence treewidth* $\mathsf{tw}_\mathsf{i}(\varphi)$ of φ as that of its incidence graph.

Example 33

Let us again consider the formula φ of Example 32. A tree decomposition of the primal graph of φ is given on the left below, while a tree decomposition of its incidence graph is given on the right:





Both of these decompositions are optimal: it is well-known that for every tree decomposition of a graph G, the vertices of every clique must be contained in a common bag. So, in this case, b, c, d, e must be in one bag for every tree decomposition of the primal graph of φ and thus $\mathsf{tw}_\mathsf{p}(F) \geq 3$, which shows that the decomposition above is optimal and $\mathsf{tw}_\mathsf{p}(F) = 3$. Concerning the treewidth of the incidence graph, remark that this graph has a cycle and is thus not a tree. Since trees are well-known to be the only connected graphs of treewidth 1, it follows that $\mathsf{tw}_\mathsf{i}(F) \geq 2$ and thus the decomposition is optimal and $\mathsf{tw}_\mathsf{i}(F) = 2$.

Let us a consider a graph $G = (V, E)$, and let us denote by $N(v)$ the open neighborhood of a vertex v of G. We say that two vertices u, v in G have the same *neighborhood type* if and only if $N(u)\backslash\{v\} = N(v)\backslash\{u\}$. It can be shown that having the same neighborhood type is an equivalence relation on V.

A generalization of treewidth is *modular treewidth* which is defined as follows.

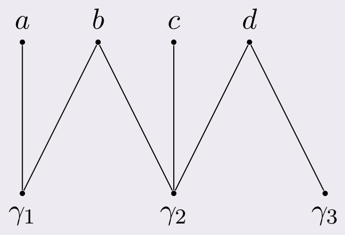
Definition 73 (Modular Treewidth)

Let G be a graph and G' be the graph obtained from G by contracting all vertices sharing a neighborhood type. Said differently, from every equivalence class we delete all vertices but one in G. The *modular treewidth* of G is defined to be the treewidth of G'.

The modular treewidth $\mathsf{mtw}(F)$ of a CNF formula φ is defined as the modular treewidth of its incidence graph.

Example 34

Let us consider again the formula φ from Example 32. A contraction of the incidence graph of the CNF formula φ is given below. In the original graph, d and e have the same neighborhood type, as do γ_3 and γ_4. We thus get the shown contraction by deleting e and γ_4. Note that the obtained graph is a tree, so that $\mathsf{mtw}(\varphi) = 1$.



Definition 74 (Cliquewidth)

The *cliquewidth* $\mathsf{cw}(G)$ of a graph G is defined as the minimum number of labels needed to construct G with the following operations:

- creating a new vertex with label i,
- taking the disjoint union of two labeled graphs,
- connecting by an edge all vertices with a label i to all vertices with a label j for $i \neq j$, and
- renaming a label i to j for $i \neq j$.

The *incidence cliquewidth* $\mathsf{cw}(\varphi)$ of a CNF formula φ is defined as the cliquewidth of the incidence graph of φ [SS13].

Finally, we consider the adaption of cliquewidth to signed graphs. To this end, let us make some additional definitions.

Definition 75 (Signed Incidence Graph)

The *signed incidence graph* G' of a CNF formula φ is the graph we get from the incidence graph $G = (V, E)$ by labeling the edges with $\{+, -\}$ as follows:

- every edge (v, γ) such that v appears positively in γ is labeled by $+$, and
- every edge (v, γ) such that v appears negatively in γ is labeled by $-$.

Definition 76 (Signed Cliquewidth)

The *signed cliquewidth* of a graph G' is defined as the minimum number of labels needed to construct G' with the following operations:

- creating a new vertex with label i,
- taking the disjoint union of two labeled graphs,
- connecting all vertices with a label i to all vertices with a label j for $i \neq j$ by an edge with label $+$,
- connecting all vertices with a label i to all vertices with a label j for $i \neq j$ by an edge with label $-$, and
- renaming a label i to j for $i \neq j$.

The *signed incidence cliquewidth* $\mathsf{scw}(\varphi)$ of CNF formulae φ is defined as the signed cliquewidth of its signed incidence graph [FMR08].

In the following, we deal with other graph width measures for a CNF formula φ, namely dual treewidth $\mathsf{tw_d}(\varphi)$ and MIM-width $\mathsf{mimw}(\varphi)$. Since for those notions we will only use some of their properties, we do not give their definition here and refer to the literature [SS10a, FMR08, Vat12, STV15, SS13]. We also consider the treewidth $\mathsf{tw}(C)$ and the cliquewidth $\mathsf{cw}(C)$ of Boolean circuits C, defined as the treewidth and cliquewidth of the underlying graph of C, respectively.

3.1.3 Communication Complexity

Here we give some very basic notions of communication complexity [KN97], focusing only on so-called *combinatorial rectangles*, which are an important object in the field.

Definition 77 (Combinatorial Rectangle)

Let X be a set of variables and $\Pi = (Y, Z)$ a partition of X. A *combinatorial rectangle* respecting Π is a Boolean function $r(X)$ that can be written as a conjunction

$$r(X) = r_1(Y) \wedge r_2(Z)$$

Definition 78 (Rectangle Cover)

Let f be a Boolean function on a variable set X. A *rectangle cover of size s* respecting a partition $\Pi = (Y, Z)$ is defined to be a representation

$$f(X) = \bigvee_{i=1}^{s} r^i(X) = \bigvee_{i=1}^{s} r_1^i(Y) \wedge r_2^i(Z)$$

where all $r^i(X) = r_1^i(Y) \wedge r_2^i(Z)$ are combinatorial rectangles respecting Π.

Example 35

By definition, all formulae in disjunctive normal forms are rectangle covers of the functions they compute respecting all possible partitions. For example, the formula φ given by

$$(\neg x \wedge \neg y \wedge z) \vee (x \wedge y \wedge z) \vee (x \wedge \neg y \wedge \neg z)$$

is a rectangle cover of size 3 respecting every partition of $\{x, y, z\}$. However, for example the partition $(\{x, y\}, \{z\})$ has the smaller rectangle cover

$$(((\neg x \wedge \neg y) \vee (x \wedge y)) \wedge z) \vee (x \wedge \neg y \wedge \neg z)$$

which has size 2. It is not hard to see that there is no smaller rectangle cover of φ for this partition.

Definition 79 (Non-Deterministic Communication Complexity)

The *non-deterministic communication complexity* of a Boolean function f with respect to a partition $\Pi = (Y, Z$ of the variables of f, denoted by $\mathsf{cc}(f, \Pi) = \mathsf{cc}(f, (Y, Z))$ is defined as $\log(s_{\min})$ where $s_{\min}$ is the minimum size of any rectangle cover of f respecting Π.

Definition 80 (Best-Case Non-Deterministic Communication Complexity)

The *best-case non-deterministic communication complexity* with $\frac{1}{3}$-balance of a Boolean function f in variables X, denoted $\mathsf{cc}_{\mathsf{best}}^{1/3}(f)$, is defined as $\mathsf{cc}_{\mathsf{best}}^{1/3}(f) := \min_{\Pi}(\mathsf{cc}(f, \Pi))$ where the minimum is over all partitions $\Pi = (Y, Z)$ of X with $\min(|Y|, |Z|) \geq |X|/3$.

Example 36

Consider the function $\mathsf{eq}_n(x_1, \dots x_n, y_1, \dots, y_n)$ which is true if and only if for every $i \in \{1, \dots, n\}$ we have $x_i = y_i$. It is well-known that, for the partition $\Pi_1 = (\{x_1, \dots, x_n\}, \{y_1, \dots, y_n\})$, we have $\mathsf{cc}(\mathsf{eq}_n, \Pi_1) = n$ [KN97, Chapter 2]. However, for the partition

$$\Pi_2 = (\{x_1, y_1, \dots, x_{\lceil n/2 \rceil}, y_{\lceil n/2 \rceil}\}, \{x_{\lceil n/2 \rceil + 1}, y_{\lceil n/2 \rceil + 1}, \dots, x_n, y_n\})$$

we have that

$$\mathsf{eq}_n(x_1, \dots x_n, y_1, \dots, y_n) = \left(\bigwedge_{i=1}^{\lceil n/2 \rceil} x_i = y_i \right) \wedge \left(\bigwedge_{i=\lceil n/2 \rceil}^{n} x_i = y_i \right)$$

is a rectangle cover of size 1 respecting Π_2. Thus, we have $\mathsf{cc}^{1/3}_{\mathsf{best}}(\mathsf{eq}_n) = \mathsf{cc}(\mathsf{eq}_n, \Pi_2) = 0$.

3.1.4 Structured Deterministic DNNF

We now introduce some representation languages used in the remainder of this chapter. For all circuits in the following, we assume that $\wedge$-gates have exactly two inputs while the number of inputs of $\vee$-gates may be arbitrary.

Definition 81 (V-Tree [PD08])

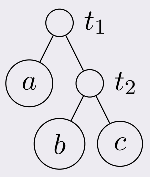
A *v-tree* T for a variable set V is a full binary tree whose leaves are in bijection with V. We call the variable assigned by this bijection to a leaf v the *label* of v.

Notation 9

For a node t of a v-tree T, we denote by T_t the subtree of T that has t as its root and by $\mathsf{var}(T_t)$ the variables that are labels of leaves in T_t.

Example 37

A v-tree for the variable set $\{a, b, c\}$ is given by:



For the internal nodes of this v-tree, the node names are given on the right of the nodes whereas for leaves we assume that the name is the label.

We now give some definitions introduced in [CM19].

Definition 82 (Complete Structured DNNF)

A *complete structured* DNNF D structured by a v-tree T is a Boolean circuit with the following properties: there is a labeling μ of the nodes in T with subsets of gates of D such that:

- For every gate g of D, there is a unique node t_g of T with $g \in \mu(t_g)$.
- If t is a leaf labeled by a variable x, then $\mu(t)$ may only contain x and $\neg x$. Moreover, for every input gate g, the node t_g is a leaf.
- For every $\vee$-gate g, all inputs are $\wedge$-gates in $\mu(t_g)$.
- Every $\wedge$-gate g has exactly two inputs g_1, g_2 that are both $\vee$-gates or input gates. Moreover, t_{g_1} and t_{g_2} are the children of t_g in T and in particular $t_{g_1} \neq t_{g_2}$.

A complete structured DNNF is called *deterministic* when the DNNF circuit itself is deterministic.

In the following, we do not mention the v-tree that structures a complete structured DNNF in cases where the form of the v-tree is unsubstantial.

Definition 83 (Width of a Complete Structured DNNF)

The *width* wi(D) of a complete structured DNNF D is defined as the maximal number of $\vee$-gates in any set $\mu(t)$.

Remark 15

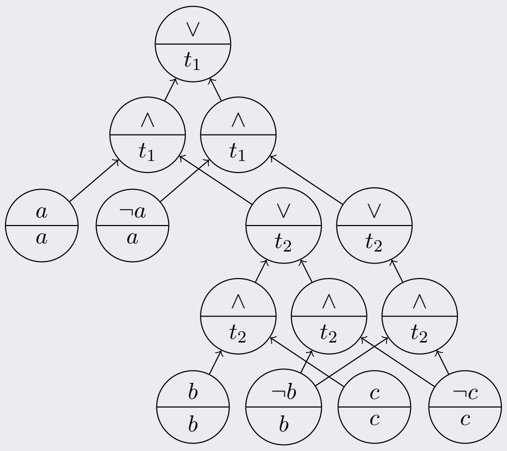
Note that we do not allow constant input gates in Definition 82. We remark that if we allowed those, we could always get rid of them in the circuit by propagation without changing any other properties of the circuit [CM19, Section 4]. We also remark that in a complete structured DNNF D, we can *forget* a variable v (see Definition 61) and construct a complete structured DNNF D' computing $\exists v D$, by setting all occurrences of v and $\neg v$ to 1 and propagating the constants in the obvious way. This operation does not increase the width [CM19]. However, if D is deterministic, this is generally not the case for D'.

Remark 16

Intuitively, a complete structured DNNF is a DNNF in which the gates are organized into blocks $\lambda(t)$ which form a tree shape. In every block one then computes a 2-DNF whose inputs are gates from the blocks that are the children of $\lambda(t)$ in the tree shape.

Example 38

A complete structured DNNF structured by the v-tree of Example 37 is given below. All gates of the DNNF show the operation of the gate (on top) and the name t of the node in the v-tree for which this gate is in $\mu(t)$ (on bottom).



3.2 The Effect of Auxiliary Variables

In this section, we motivate the use of auxiliary variables when considering width measures of CNF encodings. To this end, we will show with an example that auxiliary variables may arbitrarily reduce the treewidth of encodings. Note that this is not very surprising since it is not too hard to see that CNF representations of, say, the parity function, are of high treewidth. However, in this case the *size* of the representation is exponential, so in a sense parity is a hard function for CNF representations anyway. Here we will show that even for functions that have small CNF representations there can be a large gap between the treewidth of representations and CNF encodings with auxiliary variables. That is why we think it is useful to systematically study width measures for CNF encodings.

As an example for a function where auxiliary variables have a dramatic impact on width, consider the at-most-one function on variables $x_1, \ldots, x_n$ which accepts exactly those assignments in which at most one variable is assigned to 1:

$$\text{at-most-one}(x_1, \ldots, x_n) \equiv \sum_{i=1}^{n} x_i \leq 1$$

This constraint is actually equivalent to the cardinality constraint $\sum_{i=1}^{n} \bar{x}_i \geq n-1$ and can be represented by a CNF formula of quadratic size, given by:

$$\text{at-most-one}(x_1, \ldots, x_n) = \bigwedge_{i,j \in \{1,\ldots,n\}, i<j} \neg x_i \vee \neg x_j$$

However, this representation has as primal graph the clique K_n which is of treewidth $n-1$. We will see that in fact there is no representation of at-most-one that is of smaller primal treewidth unless one adds auxiliary variables, in which case there is a simple encoding of primal treewidth 2.

Theorem 3

Any CNF representation of at-most-one of n inputs without auxiliary variables has primal treewidth $n-1$. However, there is a CNF encoding of at-most-one of primal treewidth 2.

To prove Theorem 3, we split the statement into two lemmas.

Lemma 1

Any CNF representation of at-most-one of n inputs without auxiliary variables has primal treewidth $n-1$.

Proof. Let $x_1, \ldots, x_n$ be the variables of at-most-one. We proceed with two claims.

Claim 15. *Every non-tautological clause γ of any* CNF *representation of at-most-one must contain at least the negation of two variables from $x_1, \ldots, x_n$.*

Proof. Suppose that a clause γ does not contain two such literals. Then, there are two possible cases: either γ contains no negated variables, or exactly one. In the first case, the model of at-most-one setting all variables to 0 does not satisfy γ, so γ cannot be part of the CNF representation. In the second case, let x_i be the (only) variable of at-most-one appearing negatively in γ. Then, the model of at-most-one setting only x_i to 1 and all other variables to 0 does not satisfy γ, so γ cannot be part of the CNF representation either. Hence, at least two negated variables must appear in C.

□

From Claim 15, we deduce that all pairs of variables must appear conjointly in at least one clause.

Claim 16. *For each pair of variables x_i, x_j from $x_1, \ldots, x_n$ with $i \neq j$, there is a clause in the* CNF *representation of at-most-one containing both $\neg x_i$ and $\neg x_j$.*

Proof. Suppose that, for a pair x_i, x_j, such a clause does not exist. Let I be the assignment that sets exactly the variables x_i, x_j to 1 and all other variables to 0. Let γ be a clause from the CNF representation. By our previous claim, γ contains two negated variables from $x_1, \ldots, x_n$. Because of our assumption, at least one of these literals is neither $\neg x_i$ nor $\neg x_j$, and this literal is satisfied by I. Thus γ is satisfied by I. Since this is true for every clause γ, it follows that I satisfies all the clauses of the representation, so it is one of its models. However, I is not a model of at-most-one. As a consequence, a clause containing both $\neg x_i$ and $\neg x_j$ must exist, which is also true for every pair x_i, x_j.

□

Claim 16 shows that for each pair of variables, there is a clause containing both of them. It follows that all variables are connected to all other variables in the primal graph of the representation. So the primal graph is a clique which has treewidth $n-1$.

□

We now prove the second part of Theorem 3, which shows that if we allow the use of auxiliary variables, we may decrease the treewidth dramatically.

Lemma 2

There is a CNF encoding of at-most-one of primal treewidth 2.

Proof. We use the well-known ladder encoding presented in [GN04] and [BHvMW09, Section 2.2.5]. We introduce the auxiliary variables $y_0, \dots, y_n$. The encoding consists of the following clauses, for every $i \in \{1, \dots, n\}$:

- the validity clauses $\neg y_{i-1} \vee y_i$, and
- clauses representing the constraint $x_i \leftrightarrow (\neg y_{i-1} \wedge y_i)$

It is easy to see that this encoding is correct: the auxiliary variables y_i encode if one of the variables x_j for $j \leq i$ is assigned to 1. Concerning the treewidth bound, we construct for every index $i \in \{1, \dots, n\}$ the bag B_i corresponding to the set of variables $\{y_{i-1}, y_i, x_i\}$. Then, $(P_n, (B_i)_{i \in \{1,\dots,n\}})$ where P_n has nodes $\{1, \dots, n\}$ and edges $\{(i, i+1) \mid i \in \{1, \dots, n-1\}\}$ is a tree decomposition of the encoding of width 2.

□

3.3 Width and Communication Complexity

In this section, we show that from communication complexity we get lower bounds for the various width notions of Boolean functions. The main building block is the following result that is an application of the main result of [PD10] to complete structured DNNF.

Theorem 4

Let D be a complete structured DNNF structured by a v-tree T computing a Boolean function f in variables X. Let t be a node of T and let $Y = \mathsf{var}(T_t)$ and $Z = X \backslash \mathsf{var}(T_t)$. Finally, let n be the number of $\vee$-gates in $\mu(t)$. Then, there is a rectangle cover of f respecting the partition (Y, Z) of size at most n.

Note that [PD10] considers models that are structured DNNF which are not necessarily complete, a slightly more general model than ours. Thus their statement is slightly different. However, it is easy to see that in our restricted setting, their proof shows the statement we give above [BCMS16, Section 5]. Since Theorem 4 is somewhat technical, it will be more convenient here to use the following easy consequence.

Proposition 30

Let D be a complete structured DNNF structured by a v-tree T computing a function f in variables X. Let t be a node of T and let $Y = \mathsf{var}(T_t)$ and $Z = X \setminus \mathsf{var}(T_t)$. Then, $\log(\mathsf{wi}(D)) \geq \mathsf{cc}(f, (Y, Z))$.

Proof. From Theorem 4 and the definition of width, it follows directly that the size of an optimal rectangle cover of f respecting (Y, Z) is upper bounded by the width of D. Taking the logarithm on both sides yields the claim.

□

In many cases, instead of considering explicit v-trees, it is more convenient to simply use best-case communication complexity.

Corollary 28

Let f be a Boolean function in variables X. Then, for every complete structured DNNF computing f, we have $\mathsf{wi}(D) \geq 2^{\mathsf{cc}_{\mathsf{best}}^{1/3}(f)}$.

Proof. For every v-tree with X on the leaves, there is a node t such that $|X|/3 \leq |\mathsf{var}(T_t)| \leq 2|X|/3$. Plugging this into Proposition 30 directly yields the result. □

We will use Corollary 28 to turn compilation algorithms that produce complete structured DNNF based on a parameter of the input [ACMS18, BS17] into inexpressivity bounds based on this parameter. We first give an abstract version of this result that we will instantiate for concrete measures later on.

Theorem 5

Let L be a (fully expressive) representation language for Boolean functions. Let p be a parameter $\mathsf{p} : \mathsf{L} \to \mathbb{N}$. Assume that there is for every Boolean function f and every $\lambda \in \mathsf{L}$ that encodes f a complete structured DNNF D with $\mathsf{wi}(D) \leq 2^{\mathsf{p}(\lambda)}$. Then, we have:

$$\mathsf{p}(\lambda) \geq \mathsf{cc}_{\mathsf{best}}^{1/3}(f)$$

Proof. From the assumption, we get $\mathsf{p}(\lambda) \geq \log(\mathsf{wi}(D))$. Then we apply Corollary 28 to directly get the result. □

Intuitively, it is exactly the algorithmic usefulness of parameters that makes the resulting instances inexpressive. Note that it is not surprising that instances whose expressiveness is severely restricted allow for good algorithmic properties. However, here we see that the inverse of this statement is also true in a quite harsh way: if a parameter has good algorithmic properties allowing efficient compilation into DNNF, then this parameter puts strong restrictions on the complexity of the expressible functions.

From Theorem 5, we directly get lower bounds for many of the width measures from the literature [PSS13, SS10a, FMR08, SS13, STV15]. The first result considers the parameters with respect to which SAT is fixed-parameter tractable.

Corollary 29

There is a constant $b > 0$ such that, for every Boolean function f and every CNF φ encoding f, we have $\min\{\mathsf{tw_i}(\varphi), \mathsf{tw_p}(\varphi), \mathsf{tw_d}(\varphi), \mathsf{scw}(\varphi)\} \geq b \cdot \mathsf{cc}_{\mathsf{best}}^{1/3}(f)$.

Proof. This follows directly from Theorem 5 and the fact that for all these parameters there are algorithms that, given an input CNF of parameter value k, construct an equivalent complete structured DNNF of width $2^{\mathcal{O}(k)}$. □

Using the compilation algorithm from [ACMS18, AMS18], we get essentially the same result for circuit representations.

Corollary 30

There is a constant $b > 0$ such that for every Boolean function f and every circuit C encoding f we have:

$$\min\{\mathsf{tw}(C), \mathsf{cw}(C)\} \geq b \cdot \mathsf{cc}_{\mathsf{best}}^{1/3}(f)$$

Remark 17

For treewidth 1, the circuits of Corollary 30 boil down to so-called read-once functions, which have been studied extensively [CLH11, Chapter 10].

Finally, we give a version for parameters that allow polynomial-time algorithms when fixed but no fixed-parameter algorithms.

Corollary 31

There is a constant $b > 0$ such that for every Boolean function f in n variables and every CNF φ encoding f we have $\min\{\mathsf{mimw}(\varphi), \mathsf{cw}(\varphi), \mathsf{mtw}(\varphi)\} \geq b \cdot \frac{\mathsf{cc}_{\mathsf{best}}^{1/3}(f)}{\log(n)}$.

Proof. All of the width measures in the statement allow compilation into complete structured DNNF of size (and thus also width) $n^{\mathcal{O}(k)}$ for parameter value k and n variables [BCMS15]. Thus, with Theorem 5, for each measure there is a constant b' with $\log(n^k) = k\log(n) \geq b'\mathsf{cc}_{\mathsf{best}}^{1/3}(f)$, which completes the proof. □

Note that the bounds of Corollary 31 are lower by a factor of $\log(n)$ than those of Corollary 29. We will see in the next section that in a sense this difference is unavoidable.

3.4 Relations between Different Width Measures of Encodings

In this section, we show that the different width measures for optimal CNF encodings are strongly related. To this end, we show the relation of treewidth to all other width measures we consider. We then combine these relationships between treewidth and other width measures to analyze the relationships between all width measures we consider. We start by proving that primal treewidth bounds imply bounds for modular treewidth and cliquewidth.

Theorem 6

Let k be a positive integer and f be a Boolean function of n variables that has a CNF encoding φ of primal treewidth at most $k\log(n)$. Then f also has a CNF encoding φ' of modular incidence treewidth and cliquewidth $\mathcal{O}(k)$. Moreover, if φ has dependent auxiliary variables, then so has φ'.

Before we prove Theorem 6, let us here discuss this result a little. It is well-known that the modular treewidth and the cliquewidth of a CNF formula can be much smaller than its treewidth [SS13]. Theorem 6 strengthens this by saying essentially that for *every* function we can gain a factor logarithmic in the number of variables.

In particular, this shows that the lower bounds we can get from Corollary 31 are the best possible ones: the maximal lower bounds we can show are of the form $n/\log(n)$ and since there is always an encoding of every function of treewidth n, by Theorem 6 there is always an encoding of cliquewidth roughly $n/\log(n)$. Thus the maximal lower bounds of Corollary 31 are tight up to constants.

Note that for Theorem 6, it is important that we are allowed to change the encoding. For example, the primal graph of the formula $\varphi = \bigwedge_{i,j\in\{1,\dots,n\}}(x_{i,j}\vee x_{i+1,j})\wedge(x_{i,j}\vee x_{i,j+1})$ has the $(n\times n)$-grid as a minor and thus treewidth n [Die12, Chapter12]. But the incidence graph of φ has no modules and also has the $(n\times n)$-grid as a minor, so φ has modular incidence treewidth at least n as well. So we gain nothing by going from primal treewidth to modular treewidth without changing the encoding. What Theorem 6 tells us is that there is a different formula φ' that encodes the function of φ, potentially with some additional variables, such that the treewidth of φ' is at most $\mathcal{O}(n/\log(n))$.

Let us note that encodings with dependent auxiliary variables are often useful, e.g., when considering counting problems. In fact, for such CNF encodings, the number of models is the same as for the function they encode. It is thus interesting to see that dependence of the auxiliary variables can be maintained by the construction of Theorem 6. We will see that this is also the case for most other constructions we make.

Proof (of Theorem 6). The basic idea is that we do not treat the variables in the bags of the tree decomposition individually but organize them in groups of size $\log(n)$. We then simulate the clauses of the original formula by clauses that work on the groups. Since for every group there are only a linear number of assignments, all encoding sizes stay polynomial. We now give the details of the proof.

Let $(T,(B_t)_{t\in T})$ be a tree decomposition of φ of width at most $k\log(n)$. For every clause γ of φ there is a bag $\lambda(\gamma)$ that contains the variables of γ. By adding some copies of bags, we may assume without loss of generality that for every bag B there is at most one clause with $\lambda(\gamma)\subseteq B$ and call this clause $\lambda^{-1}(B)$.

In a first step, we construct a coloring $\mu:\mathsf{var}(\varphi)\to\{1,\dots,k+1\}$ such that in every bag there are at most $\log(n)$ variables of every color. This can be done iteratively as follows: first split the bag B_r at the root r into color classes as required. Since there are at most $k\log(n)+1$ variables in B_r by assumption, we can split them into $k+1$ color classes of size at most $\log(n)$ arbitrarily. Now let t be a node of T with parent t'. By the coloring of the variables in $B_{t'}$, some of the variables in B_t are already colored. We simply add the variables not appearing in $B_{t'}$ arbitrarily to color classes such that no color class is too big. Again, since B_t contains at most $k\log(n)+1$ variables, this is always possible. Moreover, due to the connectivity condition, there is for every variable v a unique node t_v that is closest to the root under the bags containing v. Consequently, we can make no contradictory decisions during this coloring process, so μ is well-defined.

We now construct φ'. To this end, we first introduce for every variable v and every node t such that $v \in B_t$ a new variable v_t. Now for every node t with parent t' and every color i, we add a set $\mathcal{C}_{t',t,i}$ of clauses in all variables $v_t, v_{t'}$ with $\mu(v) = i$. We construct these clauses in such a way that they are satisfied by exactly the assignments in which for each pair $v_t, v_{t'}$ such that both these variables exist, both variables take the same value. Note that the clauses in $\mathcal{C}_{t,t',i}$ have at most $2\log(n)$ variables, so there are at most n^2 of them. Moreover, they contain all the same variables. The result is a formula in which all v_t for a variable v take the same value in all satisfying assignments.

In a next step, we do for each clause γ the following: let $t = \lambda(\gamma)$. For every color i, we define $V_{i,t}$ to be the set of variables v_t such that $\mu(v) = i$. We add a fresh variable $y_{\gamma,i}$ and clauses $\mathcal{C}_{\gamma,i}$ in the variables $V_{i,t} \cup \{y_{\gamma,i}\}$ that accept exactly the assignments I with

- $I(y_{\gamma,i}) = 1$ and there is a $v_t \in V_{i,t}$ such that setting v to $I(v_t)$ satisfies γ, or
- $I(y_{\gamma,i}) = 0$ and there is no $v_t \in V_{i,t}$ such that setting v to $I(v_t)$ satisfies γ.

Next, we add the clause $\gamma' = \bigvee_{i\in\{1,\dots,k+1\}} y_{\gamma,i}$. Finally, for every variable v, rename one arbitrary variable v_t to v. This completes the construction of φ'.

We claim that φ' is an encoding of f. To see this, first note that, as discussed before, for every variable v of φ, in the satisfying assignments of φ', all v_t and v take the same value. So, we define for every assignment I of φ a partial assignment I' of φ' as an extension of I by setting $I'(v_t) = I(v)$ for every v_t. The assignment I satisfies a clause γ if and only if there is at least one variable v of γ such that $I(v)$ makes γ true. Let $\mu(v) = i$, then I satisfies γ if and only if $\mathcal{C}_{\gamma,i}$ is satisfied by the extension of I' that sets $y_{\gamma,i}$ to 1. So I satisfies γ if and only if there is an extension of I' that satisfies $\mathcal{C}_{\gamma,i}$. Consequently, I satisfies φ if and only if there is an extension I' of I that satisfies φ', so φ' is an encoding of f as claimed.

To see that the construction maintains dependence of auxiliary variables, observe first that the auxiliary variables already present in φ are still in φ' and they are still dependent. We claim that all the new variables depend on those of φ. For the variables v_t, this is immediate since they must take the same value as v in every model. Moreover, the variables $y_{\gamma,i}$ depend on the v_t by definition. As a consequence, all auxiliary variables are dependent.

We now show that the modular treewidth of φ' is at most $\mathcal{O}(k)$. First, note that all sets $V_{i,t}$ are modules as are the clause sets $\mathcal{C}_{t,t',i}$ and $\mathcal{C}_{\gamma,i}$. Without loss of generality, we may assume that, for every t, there is at most one clause γ with $\lambda(\gamma) = t$ and that T is a binary tree. We construct a tree decomposition $(T, (B_t)_{t\in\mathcal{V}(T)})$ as follows: we put a representative of $V_{i,t}$, $\mathcal{C}_{t,t',i}$, $\mathcal{C}_{t',t,i}$ and $\mathcal{C}_{\gamma,i}$ into B'_t. Moreover, we add $y_{\gamma,i}$ and γ' to B'_t. It is easy to see that constructed like this, $(T, (B_t)_{t\in\mathcal{V}(T)})$ is a tree decomposition of width at most $\mathcal{O}(k)$.

Finally, we will show that the incidence graph of the formula φ' can be constructed with $\mathcal{O}(k)$ labels. In this construction, the relabeling operation will only ever be used to *forget* labels, i.e., we change a label i into a global dummy label d such that vertices labeled by d are never used in joining operations.

In a first step, we color T with 4 colors such that for every node t, the node t, its at most two children and its parent all have different colors. We denote the color of t by $\eta(t)$. Then, for every t individually, we create the nodes in $\mathcal{C}_{\gamma,i}$, $V_{t,i}$ where γ is such that $\lambda(\gamma) = t$. The clauses in $\mathcal{C}_{\gamma,i}$ get label $(i, \eta(t), 0)$ and the variables in $V_{t,i}$ get label $(i, \eta(t), 1)$. By joining the vertices with labels $(i, \eta(t), 0)$ with those with $(i, \eta(t), 1)$, we connect the variables in $V_{t,i}$ with the clauses in $\mathcal{C}_{\gamma,i}$. We then create the $y_{\gamma,i}$, each with individual labels and connect them to the clauses with label $(i, \eta, 0)$. Finally, we create the clause vertex γ' with an individual label and connect it to the $y_{\gamma,i}$. We then forget the labels of all vertices except the $V_{t,i}$. We call the resulting graph G_t.

Note that at this point, the only thing that remains to do is to introduce the clauses in the $\mathcal{C}_{t,t',i}$ and connect them to the variables in G_t and $G_{t'}$. To do so, we work in a bottom-up fashion along T. For the leaves of T, there is nothing to do. So let t be an internal node of T with children t_1, t_2. The case in which t only has one child is treated analogously. By induction, we assume that we have graphs G'_{t_1} and G'_{t_2} containing G_{t_1} and G_{t_2} as respective subgraphs such that:

- all variables appearing in G'_{t_j} are already connected to all clauses, except the variables in the $V_{t_j,i}$ which are not yet connected to the clauses $\mathcal{C}_{t,t_1,i}$,
- all vertices in G'_{t_j} except for those in the V_{t_i} have the dummy label d.

We proceed as follows: we make a disjoint union of G_t, G'_{t_1} and G'_{t_2}. Then we create nodes for all clauses in the $\mathcal{C}_{t,t_1,i}$ giving them the label $(i, \eta(t), 2)$. Then we connect all nodes with label $(i, \eta(t_1), 1)$ to those with label $(i, \eta(t), 2)$, i.e., we connect the nodes in $V_{t_1,i}$ with the clauses in $\mathcal{C}_{t,t_1,i}$. Then we connect all nodes with label $(i, \eta(t), 1)$ to those with label $(i, \eta(t), 2)$, i.e., we connect the nodes in $V_{t,i}$ with the clauses in $\mathcal{C}_{t,t_1,i}$. We proceed analogously with t_2. Finally, we forget all labels but those for the $V_{t,i}$. This completes the construction.

Verifying the clauses in φ', one can see that the resulting graph is indeed the incidence graph of φ'. Moreover, we have only used $\mathcal{O}(k)$ clauses by construction. This completes the proof.

□

We now show that the reverse of Theorem 6 is also true: upper bounds for many width measures imply also bounds for the primal treewidth of CNF encodings. Note that this is at first sight surprising since without auxiliary variables many of those width measures are known to be far stronger than primal treewidth.

Theorem 7

Let f be a Boolean function of n variables.

a. If f has a CNF encoding φ of modular treewidth, cliquewidth or mim-width k then f also has a CNF encoding φ' of primal treewidth $\mathcal{O}(k\log(n))$ with $O(kn\log(n))$ auxiliary variables and $n^{\mathcal{O}(k)}$ clauses.
b. If f has a clausal encoding φ of incidence treewidth, dual treewidth, or signed incidence cliquewidth k, then f also has a clausal encoding φ' of primal treewidth $\mathcal{O}(k)$ with $\mathcal{O}(nk)$ auxiliary variables and $2^{\mathcal{O}(k)}n$ clauses.

To show Theorem 7 and several similar results for other width measures in this section, we make a detour through DNNF. The idea is to show that from certain DNNF representations of functions, we can get CNF encodings of primal treewidth strongly related to the width of the DNNF. Since many width measures can be used to construct small width DNNFs, we get small width CNF encodings for these width measures. We now give a precise statement of the relation between DNNF and treewidth of CNF encodings.

Lemma 3

Let f be a Boolean function in n variables that is computed by a complete structured DNNF D of width k. Then, f has a CNF encoding φ of primal treewidth $9\log(k)$ with $\mathcal{O}(n\log(k))$ variables and $\mathcal{O}(nk^3)$ clauses. Moreover, if D is deterministic, then φ has dependent auxiliary variables.

The proof of Lemma 3 will rely on so-called proof trees in DNNF, a concept that has found wide application in circuit complexity and in particular also in knowledge compilation. To this end, we make the following definition.

Definition 84

A *proof tree* $\mathcal{T}$ of a complete structured DNNF D is a circuit constructed as follows:

1. The output gate of D belongs to $\mathcal{T}$.
2. Whenever $\mathcal{T}$ contains an $\vee$-gate, we add exactly one of its inputs.
3. Whenever $\mathcal{T}$ contains an $\wedge$-gate, we add both of its inputs.
4. No other gates are added to $\mathcal{T}$.

Note that the choice in Step 2 is non-deterministic, so there are in general many proof trees for D. Observe also that due to the structure of D given by its v-tree, every proof tree is in fact a tree which justifies the name. Moreover, letting T be the v-tree of D, every proof tree of D has exactly one $\vee$-gate and one $\wedge$-gate in the set $\mu(t)$ for every non-leaf node t of T. For every leaf t, every proof tree contains an input gate v or $\neg v$ where v is the label of t in T.

The following simple observation that can easily be shown by using distributivity is the main reason for the usefulness of proof trees.

Observation 2

Let D be a complete structured DNNF and I an assignment to its variables. Then I satisfies D if and only if it satisfies one of its proof trees. Moreover, if D is deterministic, then every assignment I that satisfies D satisfies exactly one proof tree of D.

Proof (of Lemma 3). Let D be the complete structured DNNF computing f and let T be the v-tree of D. The idea of the proof is to use auxiliary variables to “guess” for every t an $\vee$-gate and an $\wedge$-gate. Then we use clauses along the v-tree T to verify that the guessed gates in fact form a proof tree and check in the leaves of T if the assignment to the variables of f satisfies the encoded proof tree. We now give the details of the construction.

We first note that, as shown by [CM19], in complete structured DNNF of width k, one may assume that every set $\mu(t)$ contains at most k^2 $\wedge$-gates so we assume this to be the case for D. For every node t of T, we introduce a set V_t of $3\log(k)$ auxiliary variables to encode one $\vee$-gate and one $\wedge$-gate of $\mu(t)$ if t is an internal node. If t is a leaf, V_t encodes one of the at most 2 input gates in $\mu(t)$. We now add clauses that verify that the gates chosen by the variables V_t encode a proof tree by doing the following for every t that is not a leaf: first, add clauses in V_t that check if the chosen $\wedge$-gate is in fact an input of the chosen $\vee$-gate. Since V_t has at most $3\log(k)$ variables, this introduces at most k^3 clauses. Let t_1 and t_2 be the children of t in T. Then we add clauses that verify if the $\wedge$-gate chosen in t has as input either the $\vee$-gate chosen in t_1 if t_1 is not a leaf, or the input gate chosen in t_1 if t_1 is a leaf. Finally, we add analogous clauses for t_2. Each of these clause sets is again in $3\log(k)$ variables, so there are at most $2k^3$ clauses in them overall. The result is a CNF formula that accepts an assignment if and only if it encodes a proof tree of D.

We now show how to verify if the chosen proof tree is satisfied by an assignment to f. To this end, for every leaf t of T labeled by a variable x, add clauses that check if an assignment to x satisfies the corresponding input gate of D. Since $\mu(t)$ contains at most 2 gates, this only requires at most 4 clauses. This completes the construction of the CNF encoding. Overall, since T has n internal nodes, the CNF has $n(3\log(k)+1)$ variables and $3nk^3+4n$ clauses.

It remains to show the bound on the primal treewidth. To this end, we construct a tree decomposition $(T,(B_t)_{t\in\mathcal{V}(T)})$ with the v-tree T as underlying tree as follows: for every internal node $t \in \mathcal{V}(T)$, we set $V_t = V_t \cup V_{t_1} \cup V_{t_2}$ where t_1 and t_2 are the children of t. Note that for every clause that is used for checking if the chosen nodes form a proof tree, the variables are thus in a bag B_t. For every leaf t, set $B_t = V_t \cup \{v\}$ where v is the variable that is the label of t. This covers the remaining clauses. It follows that all edges of the primal graph are covered. To check the third condition of the definition of a tree decomposition, note that every auxiliary variable in a set V_t appears only in B_t and potentially in $B_{t'}$ where t' is the parent of t in T. Thus $(T,(B_t)_{t\in\mathcal{V}(T)})$ constructed in this way is a tree decomposition of the primal graph of φ. Obviously, the width is bounded by $9\log(k)$ since every V_t has size $3\log(k)$, which completes the proof.

□

Proof (of Theorem 7). We first prove a. As shown by [BCMS15], whenever the function f has a CNF encoding φ with one of the width measures from this statement bounded by k, then there is also a complete structured DNNF D of width $n^{\mathcal{O}(k)}$ computing φ. Now forget all auxiliary variables of φ to get a DNNF representation D' of f. Note that since forgetting does not increase the width [CM19], D' also has width at most $n^{\mathcal{O}(k)}$. We then simply apply Lemma 3 to get the result.

To see b, just observe that, following the same construction, the width of D is $2^{\mathcal{O}(k)}$ for all considered width measures [BCMS15].

□

Remark that the construction of Theorem 7 has a surprising property: the size and the number of auxiliary variables of the constructed encoding φ' does *not* depend on the size of the initial encoding at all. Both depend only on the number of variables in f and the width.

To maintain dependence of the auxiliary variables in the above construction, we have to work some more than for Theorem 7. We start with some definitions.

Definition 85 (Reduced Complete Structured DNNF)

We say that a complete structured DNNF is *reduced* if, from every gate, there is a directed path to the output gate. Note that every complete structured DNNF can be turned into a reduced DNNF in linear time by a simple graph traversal and that this transformation maintains determinism and structuredness by the same v-tree.

The following property will be useful.

Lemma 4

Let d be a reduced complete structured DNNF and let g be a gate in D. Let I_g be an assignment to $\mathsf{var}(g)$, the variables in the subcircuit rooted in g, that satisfies g. Then, I_g can be extended to an assignment I that satisfies D.

Proof. We use the fact that an assignment to D is satisfying if and only if there is a proof tree that witnesses this. So let $\mathcal{T}_g$ be a proof tree that witnesses I_g satisfying g. We extend it to a proof tree for an extension I of I_g as follows: first add a path from g to the output gate to $\mathcal{T}_g$ and then iteratively add more gates as required by the definition of proof trees where the choices in $\vee$-gates are performed arbitrarily. The result is an extension $\mathcal{T}$ of $\mathcal{T}_g$ which witnesses that an assignment I that extends I_g satisfies D. $\square$

Definition 86 (Definability)

Let f be a function in variables $\mathcal{V} \cup \{z\}$. We say that z is *definable* in $\mathcal{V}$ with respect to f if there is a function g such that for all assignments I with $f(I) = 1$ we have $I(z) = g(I|\mathcal{V})$.

Lemma 5

Let f be a function in variables $\mathcal{V} \cup \{z\}$ such that z is definable in $\mathcal{V}$ with respect to f. Let D be a reduced complete structured deterministic DNNF computing f. Then the complete structured DNNF D' we get from D by forgetting z is deterministic as well.

Proof. By way of contradiction, assume this were not the case. Then there is an $\vee$-gate g in D' and an assignment I' to $\mathcal{V}$ such that two children g_1 and g_2 are satisfied by I'. By Lemma 4, we may assume that I' satisfies D'. Then, there are extensions I_1 and I_2 of I that assign a value to z such that I_1 satisfies g_1 and I_2 satisfies g_2 in D. Note that both I_1 and I_2 satisfy D and thus, by definability, I_1 and I_2 assign the same value to z. So $I_1 = I_2$ and hence I_1 satisfies both g_1 and g_2 in D which contradicts the determinism of D. $\square$

Theorem 8

Let f be a Boolean function of n variables.

a. If f has a CNF encoding φ with dependent auxiliary variables of modular treewidth, cliquewidth or mim-width k then f also has a CNF encoding φ' with dependent auxiliary variables of primal treewidth $\mathcal{O}(k \log(n))$ with $\mathcal{O}(kn \log(n))$ auxiliary variables and $n^{\mathcal{O}(k)}$ clauses.
b. If f has a CNF encoding with dependent auxiliary variables of incidence treewidth, dual treewidth, or signed incidence cliquewidth k, then f also has a clausal encoding φ' with dependent auxiliary variables of primal treewidth $\mathcal{O}(k)$ with $\mathcal{O}(nk)$ auxiliary variables and $2^k n$ clauses.

Proof. The proof is essentially the same as that of Theorem 7 with some additional twists. First, observe that the complete structured DNNF D constructed with the construction of [BCMS15] is deterministic. Then we use Lemma 5 when forgetting the auxiliary variables and get a D' that is deterministic without increasing the width. Then, since D' is deterministic, we can construct a CNF encoding with dependent auxiliary variables using Lemma 3. $\square$

Next we will show that signed incidence cliquewidth is linearly related to primal treewidth when allowing auxiliary variables. We will state a result similar to Lemma 3. To do so, we start with a special case for which we introduce some more definitions.

Definition 87 (Special Tree Decomposition [Cou12])

A *special tree decomposition* of a graph G is defined as a tree decomposition $(T, (B_t)_{t\in\mathcal{V}(T)})$ in which for every vertex $v \in \mathcal{V}(G)$, the set $\{t \in \mathcal{V}(T) \mid v \in B_t\}$ lies on a leaf-root path in T.

Definition 88 (Special Treewidth)

The *special treewidth* of a graph G is defined as the smallest width of any special tree decomposition of G.

The *primal special treewidth* of a CNF formula is the special treewidth of its primal graph.

Lemma 6

Every CNF formula of primal special treewidth k has signed incidence cliquewidth at most $k + 1$.

Proof. Let $(T, (B_t)_{t\in\mathcal{V}(T)})$ be a special tree decomposition of the primal graph of a CNF formula φ. It is well-known that for every clause γ there is a node $t = \lambda(\gamma)$ of T such that all variables of γ are in B_t. By adding copies of some bags B_t along a root-leaf path in T, we may assume that $\lambda(\gamma) \neq \lambda(\gamma')$ for every pair γ, γ' of clauses with $\gamma \neq \gamma'$.

We will show how to construct the signed incidence graph G' of φ with the operations in the definition of signed cliquewidth along the tree T. In a first step, we label every variable v of φ with a color $\mu(x)$ from $\{1, \ldots, k+1\}$ such that in every bag B_t there are no two variables with the same label $\mu(v)$. This can be done similarly to the first step of the proof of Theorem 6 by descending from the root to the leaves and labeling the variables in the bags along this way. The label $\mu(v)$ will be the label that the variable gets when it is created in the construction of G'. As in the proof of Theorem 6, the only renamings of labels that we will perform will be forget operations, i.e., renaming a label to a dummy label d.

For the construction of G', we iteratively construct for every $t \in \mathcal{V}(T)$ a graph G_t that contains all variables in $S_t = \bigcup_{t'\in\mathcal{V}(T_t)} B_t$ where T_t is the subtree of T rooted in t. Moreover, G_t contains all clauses such that $\lambda(\gamma)$ lies in T_t and all signed edges connecting them to their variables.

If t is a leaf, then we create all variables in B_t and if there is a clause γ with $\lambda(\gamma) = t$, we introduce it with color $k + 2$. Since all variables of γ have different colors, we can then introduce all signed edges individually. This completes the construction for the leaf case.

Let now t be an internal node with children $t_1, \ldots, t_l$. By assumption, we have already constructed $G_{t_1}, \ldots, G_{t_l}$. Note that for every i the variables in G_{t_i} that are not in B_t are by construction already connected to all their clauses in G_{t_i}, so we can safely forget their label in a first step. Now we take the disjoint union of all G_{t_i}. Note that this union is in fact disjoint, because, since we start from a special tree decomposition, no node appears in more than one G_{t_i}. Now we create the variables which appear in B_t but not in any G_{t_i}. Note that at this point the vertices with non-dummy labels are exactly those in B_t. If there is no clause γ with $\lambda(\gamma) = t$, we are done. Otherwise, we create γ and connect it to all its variables by signed edges as in the leaf case. This completes the construction of G_t.

For the root r of T we have $G_r = G'$ by definition. Moreover, we have used at most $k + 2$ labels. This completes the proof.

□

With Lemma 6, we can give a version of Lemma 3 for signed incidence cliquewidth easily.

Lemma 7

Let f be a Boolean function in n variables that is computed by a structured DNNF D of width k. Then f has a clausal encoding φ of signed incidence cliquewidth and primal special treewidth $\mathcal{O}(\log(k))$ with $\mathcal{O}(n \log(k))$ variables and $\mathcal{O}(nk^3)$ clauses. Moreover, if D is deterministic then φ has dependent auxiliary variables.

Proof. We only have to observe that in fact the tree decomposition in the proof of Lemma 3 is special and apply Lemma 6.

□

Corollary 32

Let f be a function with a CNF representation of primal treewidth k. Then f has a clausal encoding of signed incidence cliquewidth and special treewidth $\mathcal{O}(k)$.

We can now state our main result.

Theorem 9

Let $A = \{\mathsf{tw_p}, \mathsf{tw_d}, \mathsf{tw_i}, \mathsf{scw}\}$ and $B = \{\mathsf{mtw}, \mathsf{cw}, \mathsf{mimw}\}$. Let f be a Boolean function in n variables.

a. Let $w_1 \in A$ and $w_2 \in B$. Then, there are constants c_1 and c_2 such that the following holds: let φ_1 and φ_2 be CNF representations for f with minimal w_1-width and w_2-width, respectively. Then, if $w_1(\varphi_1) \leq k \log(n)$, then $w_2(\varphi_2) \leq c_1 k$, and if $w_2(\varphi_2) \leq k$, then $w_1(\varphi_1) \leq c_2 k \log(n)$.
b. Let $w_1 \in A$ and $w_2 \in A$ or $w_1 \in B$ and $w_2 \in B$. Then, there are constants c_1 and c_2 such that the following holds: let φ_1 and φ_2 be CNF representations for f with minimal w_1-width and w_2-width, respectively. Then, $w_1(\varphi_1) \leq k \Rightarrow w_2(\varphi_2) \leq c_1 k$ and $w_2(\varphi_2) \leq k \Rightarrow w_1(\varphi_1) \leq c_2 k$.

Proof. Assume first that $w_1 = \mathsf{tw_p}$. For a, we get the second statement directly from Theorem 7 a. For cw and mtw, we get the first statement by Theorem 6. For mimw, it follows by the fact that for every graph G, $\mathsf{mimw}(G) \leq c \cdot \mathsf{cw}(G)$ for some constant c [Vat12, Section 4].

For b, the second statement is Theorem 7 b. Since for every formula φ we have $\mathsf{tw_i}(\varphi) \leq \mathsf{tw_p}(\varphi) + 1$ by [FMR08], the first statement for $\mathsf{tw_i}$ is immediate. For scw it is shown in Corollary 32, while for $\mathsf{tw_d}$ it has been shown by [SS10b]. All other combinations of w_1 and w_2 can now be shown by an intermediate step using $\mathsf{tw_p}$.

□

3.5 Some Applications

We now consider some applications of the results of this chapter to give lower bounds for different encodings. To this end, we will need the following notations.

Definition 89 ($\Omega(\cdot)$)

Given a function $f : \mathbb{N} \to \mathbb{N}$ and a function $g : \mathbb{N} \to \mathbb{N}$, we say that f is in $\Omega(g(n))$, denoted $f(n) = \Omega(g(n))$, if there exists $n_0 \in \mathbb{N}$ and a constant $c \in \mathbb{N}$ such that, for any $n \geq n_0$, $cg(n) \leq f(n)$.

Definition 90 ($\Theta(\cdot)$)

Given a function $f : \mathbb{N} \to \mathbb{N}$ and a function $g : \mathbb{N} \to \mathbb{N}$, we say that f is in $\Theta(g(n))$, denoted $f(n) = \Theta(g(n))$, if both $f(n) = \mathcal{O}(g(n))$ and $f(n) = \Omega(g(n))$.

3.5.1 Cardinality Constraints

In this section, we focus on *cardinality constraints*. Without loss of generality, we consider here cardinality constraints over positive literals, i.e. constraints of the form $\sum_{i=1}^{n} x_i \geq \delta$, in which all x_i are propositional variables. Let us denote this cardinality constraint by κ_n^δ.

In Chapter 2, we have studied the properties of cardinality constraints through the representation language CARD (see Definition 67). We also note that many CNF encodings are known for representing such constraints [Sin05]. Here we add another perspective on cardinality constraint encodings by determining their optimal treewidth. We start with the following easy observation.

Observation 3

κ_n^δ has an encoding of primal treewidth $\mathcal{O}(\log(\min(\delta, n - \delta)))$.

Proof. First, assume that $\delta < n/2$. We iteratively compute the partial sums of S_j given by $\sum_{i=1}^{j} x_i$ and encode their values in $\log(\delta) + 1$ bits $y_1^j, \dots, y_{\log(\delta)+1}^j$, denoted by Y^j. We cut these sums off at δ (if we have seen at least δ variables set to 1, this is sufficient to compute the output). In the end we encode a comparator comparing the last sum S_n to δ. Since the computation of S_{j+1} can be done from S_j and x_{j+1}, we can compute the partial sums with clauses containing only the variables in $Y^j \cup Y^{j+1} \cup \{x_{j+1}\}$, so $\mathcal{O}(\log(\delta))$ variables. The resulting CNF formula can easily be seen to be of treewidth $\mathcal{O}(\log(\delta))$.

If $\delta > n/2$, we proceed similarly but count variables assigned to 0 instead of those set to 1.

□

We remark that our construction is the "basic approach" described in [BHvMW09, Section 8.6.7]. It has some similarity with the sequential counter introduced in [Sin05]. The main difference is that we encode the partial sums S_j in binary whereas in the sequential counter, they are encoded in unary. This choice is based on the fact that our encoding has smaller treewidth, which is the parameter we are optimizing for. We did not empirically evaluate how this encoding performs in practice as this work is purely theoretical, but doing so could provide another point of view on this work. We now show that Observation 3 is essentially optimal.

Proposition 31

Let $\delta < n/2$. Then $\mathsf{cc}^{1/3}_{\mathsf{best}}(\kappa^\delta_n) = \Omega(\log(\min(\delta, n/3)))$.

Proof. Let $s = \min(\delta, n/3)$. Consider an arbitrary partition Y, Z with $\frac{n}{3} \leq |Y| \leq \frac{2n}{3}$. We show that every rectangle cover of κ^δ_n must have s rectangles. To this end, choose assignments $(I_0, I'_0), \dots, (I_s, I'_s)$ such that $I_i : Y \to \{0, 1\}$ assigns $\delta - i$ variables to 1 and $I'_i : Z \to \{0, 1\}$ assigns i variables to 1. Note that every (I_i, I'_i) satisfies κ^δ_n. We claim that no rectangle $r_1(Y) \wedge r_2(Z)$ in a rectangle cover of κ^δ_n can have models (I_i, I'_i) and (I_j, I'_j) for $i \neq j$. To see this, assume that such a model exists and that $i < j$. Then, the assignment (I_j, I'_i) is also a model of the rectangle since I_j satisfies $r_1(Y)$ and I'_i satisfies $r_2(Z)$. But (I_j, I'_i) contains strictly less than δ variables assigned to 1, so the rectangle $r_1(Y) \wedge r_2(Z)$ cannot appear in a rectangle cover of κ^δ_n. Thus, every rectangle cover of κ^δ_n must have a different rectangle for every model (I_i, I'_i) and thus at least s rectangles. This completes the proof. $\square$

A symmetric argument shows that for $\delta > n/2$ we have the lower bound $\mathsf{cc}^{1/3}_{\mathsf{best}}(\kappa^\delta_n) = \Omega(\log(\min(n - \delta, n/3)))$. Observing that $\delta < n$ for non-trivial cardinality constraints, we get the following from Theorem 4.

Corollary 33

CNF encodings of smallest primal treewidth for κ^δ_n have primal treewidth

$$\Theta(\log(\min(\delta, n - \delta)))$$

The same statement is true for dual and incidence treewidth and signed incidence cliquewidth. For incidence cliquewidth, modular treewidth and mim-width, there are CNF encodings of κ^δ_n of constant width.

3.5.2 The Permutation Function

We now consider the permutation function perm which has the n^2 input variables $X_n = \{x_{i,j} \mid i, j \in \{1, \dots, n\}\}$ thought of as a matrix in these variables. The function perm evaluates to 1 on an input I if and only if I is a permutation matrix, i.e., in every row and in every column of I there is exactly one 1.

Example 39

The function perm_2 has the variables $x_{1,1}, x_{1,2}, x_{2,1}, x_{2,2}$ which we interpret organized as the matrix $\begin{pmatrix} x_{1,1} & x_{1,2} \\ x_{2,1} & x_{2,2} \end{pmatrix}$. The only inputs on which perm_2 evaluates to 1 are $\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$ and $\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$. Inputs on which perm_2 evaluates to 0 are for example $\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$ (the first row has more than one 1-entry) and $\begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}$ (the first column has no 1-entry).

The function perm is known to be hard in several versions of branching programs [Weg00]. In [BKM11], it is shown that CNF encodings of perm require treewidth $\Omega(n/\log(n))$. We here give an improvement by a logarithmic factor.

Lemma 8

For every v-tree T on variables X_n, there is a node t of T such that $\mathsf{cc}(\mathsf{perm}, Y, Z) = \Omega(n)$ where $Y = \mathsf{var}(T_t)$ and $Z = X \backslash Y$.

Proof. The proof is a variation of arguments used in [BKM11, Kra88, Weg00, Section 4.12]. Since all models of perm assign exactly n variables to 1, for every model M of perm, there is a node t_M in T such that T_M contains between $n/3$ and $2n/3$ variables assigned to 1 by M. Since T has n internal nodes and perm has $n!$ models, there must be a node t such that for at least $(n-1)!$ of the models M we have $t = t_M$. We will show in the remainder that t has the desired property.

Denote by $\mathcal{M}$ the set of models M of perm for which $t_M = t$. Let $Y = \mathsf{var}(T_t)$ and $Z = X_n \setminus Y$ as in the statement of the lemma. Every model M of perm corresponds to a permutation π_M on $\{1, \ldots, n\}$ that assigns every $i \in \{1, \ldots, n\}$ to the j such that $M(x_{i,j}) = 1$. Note that because of the properties of M, π_M is well-defined and indeed a permutation.

Let $R(X) = r_1(Y) \wedge r_2(Z)$ be a rectangle in a rectangle cover of perm with partition (Y, Z). We will show that $R(X)$ contains few models from $\mathcal{M}$. To this end, fix a model $M \in \mathcal{M}$ of $R(X)$ and define $I(M) = \{i \mid x_{i,\pi_M(i)} \in Y\}$. Observe that, if we denote $|I(M)|$ by k, then k is the number of variables in Y that are assigned to 1 by M and thus $n/3 \leq k \leq 2n/3$. Let M' be another model of $R(X)$. Then $I(M') = I(M)$ because otherwise $M|_Y \cup M'|_Z$ does not encode a model, where $M|_Y$ denotes the restriction of M to Y and $M'|_Z$ that of M' to Z. Letting $I'(M) = \{\pi_M(i) \mid i \in I(M)\}$, we get similarly that for all models M' of $R(X)$ we have $I'(M) = I'(M')$. It follows that the models of $r_1(Y)$ are all bijections between $I(M)$ and $I'(M)$ and thus $r_1(Y)$ has at most $k!$ models.

By a symmetric argument, one sees that $r_2(Z)$ has at most $(n-k)!$ models. Thus, the number of models of R is bounded by $k!(n-k)! \leq \left(\frac{n}{3}\right)! \left(\frac{2n}{3}\right)!$. As a consequence, to cover all $(n-1)!$ models in $\mathcal{M}$, one needs at least

$$\frac{(n-1)!}{\left(\frac{n}{3}\right)! \left(\frac{2n}{3}\right)!} = \frac{1}{n}\binom{n}{\frac{n}{3}} \geq \frac{1}{n}\left(\frac{n}{\frac{n}{3}}\right)^{\frac{n}{3}} = \frac{1}{n}\sqrt[3]{3}^n$$

rectangles, which completes the proof.

□

As a consequence of Lemma 8, we get an asymptotically tight treewidth bound for encodings of perm.

Corollary 34

CNF encodings of smallest primal treewidth for perm_n have primal treewidth $\Theta(n)$.

Proof (sketch). The lower bound follows by using Lemma 8 and Proposition 30 and then arguing as in the proof of Theorem 5.

For the upper bound, observe that checking if out of n variables exactly one has the value 1 can easily be done with n variables. We apply this for every row in a bag of a tree decomposition. We perform these checks for one row after the other and additionally use variables for the columns that remember if in a column we have seen a variable assigned 1 so far. Overall, to implement this, one needs $\mathcal{O}(n^2)$ auxiliary variables and gets a formula of treewidth $\mathcal{O}(n)$.

□

From Corollary 34 we get the following bound by applying Theorem 6. This answers an open problem from [BKM11], who showed only conditional lower bounds for the incidence cliquewidth of encodings of perm.

Corollary 35

CNF encodings of smallest incidence cliquewidth for perm_n have width $\Theta(n/log(n))$.

In this chapter, we have shown several results on the expressiveness of CNF encodings with restricted underlying graphs. In particular, we have seen that many graph width measures from the literature put strong restrictions on the expressiveness of encodings. We have also seen that, contrary to the case of representations by CNF formulae, in the case where auxiliary variables are allowed, all width measures we have considered are strongly related to primal treewidth and never differ by more than a logarithmic factor. Moreover, most of our results are also true while maintaining dependence of auxiliary variables.

From a practical standpoint, one point of our results might be that formulae solved with width-based algorithms as those from the theoretical literature can likely only deal with quite simple formulae. Otherwise, for example if formulae contain big cardinality constraints or pseudo-Boolean constraints, the width of the formulae might be infeasibly high. This is because all those algorithms are at least exponential in the width of the input. An implementation of such algorithms would thus likely have to implement heuristics and optimizations not presented in the theory literature. For example, [FHZ19] showed that one can use parallelism of GPUs to improve the efficiency of treewidth-based counting and thus scale to higher treewidth.

Conclusion

We have studied properties of representation languages based on pseudo-Boolean constraints. In particular, we have shown that pseudo-Boolean constraints offer a succinct alternative to clauses, that they generalize. Interestingly, all the tractable queries available for CNF remain tractable when considering pseudo-Boolean constraints. However, this is not the case for the transformations that are offered by CNF, as some of them, such as forgetting and bounded disjunction, become intractable in general when dealing with pseudo-Boolean constraints. Thus, from a knowledge representation perspective, pseudo-Boolean languages cannot be considered as actual *compilation languages*, since important queries (especially, regarding consistency) cannot be computed in polynomial time, unless $\mathsf{P} = \mathsf{NP}$.

We have also studied properties of CNF encodings, as those used to encode pseudo-Boolean formulae into CNF formulae of reasonable size, thanks to auxiliary variables. We have shown that, if the width of such encodings is bounded, their expressiveness may be drastically limited, restricting the formulae to those of low communication complexity.

The main advantage of pseudo-Boolean constraints, when considered as a representation language, is thus their succinctness. Another benefit of using pseudo-Boolean constraints instead of clauses is that it is possible to reason on them using a stronger framework, based on the cutting planes proof system [Gom58, Hoo88]. This reasoning aspect of pseudo-Boolean constraints is studied in the following part of the document.

Part II

Pseudo-Boolean Solving

Introduction

During the last decades, many improvements have been made in satisfiability solving. In practice, so-called “modern” SAT solvers are particularly efficient, and can now solve problems with million of variables and clauses, while doing so was considered out of reach in the early 1990s [JLRS12]. This “SAT revolution” has been made possible by the development of the *conflict-driven clause learning architecture* (CDCL) [MS99] and of efficient data structures and heuristics [MMZ$^+$01, ES04].

However, some instances remain hard to solve for current SAT solvers. This is particularly true for inconsistent formulae for which only exponential-sized unsatisfiability proofs exist in the resolution proof system on which they are based. For instance, this is the case for *pigeonhole principle* formulae that state that it is not possible to put n pigeons in $n - 1$ holes [Hak85]. Many of those hard formulae require the solvers to be able to either detect and break symmetries or to “count” so as to generate short proofs [BS94, DBBD16, MBK19].

This was an important motivation for the development of pseudo-Boolean reasoning [RM09a], which benefits from the expressiveness of pseudo-Boolean constraints and from the strength of the *cutting planes* proof system [Gom58, Hoo88, Nor15]. This proof system is in theory strictly stronger than the resolution proof system used in SAT solvers, as the former p-simulates the latter [CCT87]: any resolution proof can be simulated by a cutting planes proof of polynomial size with respect to the size of the original proof.

In practice, however, none of the current pseudo-Boolean solvers uses the full power of the cutting planes proof system [VEG$^+$18]. Indeed, most of them exploit a subset of cutting planes which can be viewed as a generalization of resolution [Hoo88]. This allows to extend clausal inference to pseudo-Boolean inference, inheriting many of the techniques used in SAT solving [DG02, CK05]. In particular, the conflict-driven clause learning architecture has been extended to pseudo-Boolean problems, and many solvers have been developed in this direction [DG02, CK05, SS06, LP10, EN18].

Disappointingly, pseudo-Boolean solvers fail to keep in practice the promises of theory. While they perform generally well on specific classes of benchmarks, they fail to run uniformly well on all benchmarks [EGNV18]. This is partly due to the complexity of deciding when to use the rules of the cutting planes proof system, and of implementing their application efficiently. The initial trend has been to replace the application of the resolution rules by the generalized resolution rules [Hoo88] during conflict analysis. However, this approach is not satisfactory because it is equivalent to resolution when applied to clauses, and requires a specific preprocessing to derive cardinality constraints [BLLM14, EN20].

In the following, we study the resolution of the decision problem consisting in determining whether a pseudo-Boolean formula is consistent. To this end, Chapter 4 surveys the current state of pseudo-Boolean solving. In particular, we introduce the CDCL architecture of modern SAT solvers and its extension to pseudo-Boolean solving. We also briefly describe how resolution-based SAT solvers can be used to solve pseudo-Boolean problems.

In Chapter 5, we consider the problem of irrelevant literals produced by pseudo-Boolean solvers during their conflict analyses [LMMW20]. Such literals have no effect on the truth value of the constraints in which they appear. However, if such a constraint is used during conflict analysis to derive new constraints, the resulting constraints may be weaker than they could be if irrelevant literals were not there in the first place. They may thus impact the performance of the solver, as weaker constraints may lead to longer unsatisfiability proofs.

Chapter 6 investigates different weakening strategies that can be applied by pseudo-Boolean solvers to preserve the conflict during conflict analysis [LMW20]. In particular, we show that one can weaken so-called *ineffective* literals to also remove irrelevant literals, while doing so forces to only derive clauses, and thus weak constraints. We also study how to implement in a solver based on *RoundingSat* [EN18] a less aggressive weakening strategy that improves the performance of the solver.

Chapter 7 finally extends different strategies inspired by those used in resolution-based CDCL SAT solvers to pseudo-Boolean solving. More precisely, we consider different branching heuristics, restart policies, and learned constraint deletion strategies which are well-known to play a key role in the efficiency of modern SAT solvers. Our extensions of these strategies take advantage of the specific properties of pseudo-Boolean constraints to improve the performance of pseudo-Boolean solvers. Preliminary results of this contribution have been presented in a workshop [Wal20] and in a seminar [Nor20]. The feedback received allowed us to improve the reporting and analysis of our experimental results.

Chapter 4

State of Pseudo-Boolean Solving

Compared to CNF formulae, pseudo-Boolean formulae have a number of advantages. In addition to being more succinct, they are also a more natural language to encode a wide variety of problems, such as the well-known *subset sum* or *knapsack* problems (see, e.g., [CLRS09]). Over the years, several approaches have thus been proposed for pseudo-Boolean problems, following the development of SAT solvers [FM09]. For instance, the Davis-Putnam procedure [DP60] has been adapted into *opbdp* [Bar95] for handling pseudo-Boolean optimization. Different pseudo-Boolean solvers based on the SAT solver *GRASP* [MS99] have also been developed, such as *bsolo* [MS97], which integrates a *branch-and-bound* procedure, or *SATIRE* [WS01], which extends its conflict-driven clause learning architecture to natively support pseudo-Boolean constraints (while still performing the conflict analysis on clauses). Local search incomplete algorithms have also been proposed for solving such problems [Wal97]. Since the "SAT revolution" initiated by *Chaff* [MMZ$^+$01], most SAT solvers have adopted the CDCL architecture, especially the implementation proposed by *Minisat* [ES04]. This is also true for pseudo-Boolean solvers, either by directly using a SAT solver on a CNF encoding of the pseudo-Boolean formula, or by natively supporting pseudo-Boolean reasoning through the cutting planes proof system [RM09a]. In this chapter, we focus on such current pseudo-Boolean solvers, and survey the existing techniques for solving pseudo-Boolean constraints. We also discuss the pros and cons of different techniques.

4.1 Practical SAT Solving

Despite the NP-completeness of the SAT problem [Coo71], so-called "modern" SAT solvers are currently often able to solve industrial CNF instances with millions of variables and clauses [JLRS12]. This practical efficiency is mostly due to the combination of different building blocks, such as the *conflict-driven clause learning* architecture [MS99], and the use of efficient data structures and heuristics [MMZ$^+$01, ES04]. These building blocks are studied in this section.

4.1.1 Exploring the Search Space

In order to find a model of an input formula or to prove its inconsistency, SAT solvers try different (partial) assignments of the variables of this formula. This allows either to eliminate falsifying assignments or to find a satisfying assignment (if any). This exploration of the search space is done by making *decisions* and *propagating* literals.

Definition 91 (Decision)

Making a *decision* is the process of selecting an unassigned variable of the input formula and assigning it to a chosen truth value.

Remark 18

Note that Definition 91 does not specify *how* to select the variable to assign, and neither it describes how to choose *which* truth value to assign. For now, we only assume that these choices are performed *heuristically*. Some heuristics are described later in this chapter.

When a variable has been assigned (e.g., after a decision), the formula may be simplified. In particular, clauses that contain the corresponding satisfied literal may be removed from the formula, and clauses containing the corresponding falsified literal may be shortened on this literal, as in the following example.

Example 40

Consider a CNF formula Σ and a clause γ of Σ defined by $a \vee \neg b \vee c$. If c becomes falsified, the clause γ can be shortened into $a \vee \neg b$. If b is now falsified, the literal $\neg b$ becomes satisfied, which also satisfies the clause. In this latter case, γ can thus be removed from Σ.

Typically, when making a decision, we eliminate the (possibly) satisfying assignment in which the opposite decision is taken. However, if the literal is *pure*, this does not happen.

Definition 92 (Pure Literal)

Given a CNF formula Σ and a literal $\ell \in \mathsf{lit}(\Sigma)$, ℓ is said to be *pure* when $\neg\ell \notin \mathsf{lit}(\Sigma)$.

Example 41

Consider a CNF formula Σ defined by $(a \vee \neg b \vee c) \wedge (b \vee \neg c)$. In this formula, a is pure, while b and c are not.

Pure literals are useful because they allow removing clauses from the input formula while avoiding to shorten, and thus strengthen, other clauses. In this context, the strengthening of the shortened clauses is due to the fact that making a decision eliminates all (possibly) satisfying assignments in which the opposite decision is made, which may also be the only satisfying assignments of the considered formula. On the contrary, the (in)consistency of the formula is preserved when satisfying pure literals and simplifying the input formula accordingly. However, the obtained formula is not equivalent to the original formula in general (the formulae are only equisatisfiable). On the contrary, equivalence is preserved when satisfying *unit literals*.

Definition 93 (Unit Clause)

A *unit clause* is a clause that contains exactly one literal. This unique literal is called a *unit literal*.

Example 42

The clause $\neg b$ is unit while the clause $b \vee \neg c$ is not.

Unit clauses are particularly useful during the exploration of the search space, because they *force* their unit literal to be satisfied, as it is the only way to satisfy such a clause. When unit clauses are detected, the corresponding literals are thus *propagated* to true.

Definition 94 (Unit Propagation)

Unit propagation (also known as *Boolean Constraint Propagation* (*BCP*)) is the process of satisfying all unit literals appearing in a CNF formula, simplifying the formula, and repeating the operation until no more unit literals appear in the formula.

Notation 10

When a unit clause γ propagates a literal ℓ to true, we we say that γ is the *reason* for ℓ, which is denoted by $\mathsf{reason}(\ell) = \gamma$.

This latter notation illustrates the main difference between *decisions* and *propagations*: while they both assign literals, the former does not have a clear *reason* for making the assignment, while the latter has been forced to a truth value because of a unit clause.

As mentioned above, any assignment, no matter if it is a decision or a propagation, may trigger simplifications on the input formula. In practice, most current SAT solvers only perform the simplification of the formula *implicitly*: clauses are not actually shortened nor removed from the formula. Instead, SAT solvers keep track of the current assignment of their literals. In particular, they maintain for each literal the *decision level* of its assignment (if any).

Definition 95 (Decision Level)

Let v be a propositional variable. The *decision level* of v is the number $\mathsf{dl}(v)$ such that:

- either v is the $\mathsf{dl}(v)$-th variable on which the solver has made a decision, or
- the truth value of the variable v has been unit propagated after the $\mathsf{dl}(v)$-th decision and before the $(\mathsf{dl}(v) + 1)$-th decision.

The decision level of a literal ℓ is defined as the decision level of the corresponding variable, i.e., $\mathsf{dl}(\ell) = \mathsf{dl}(\mathsf{var}(\ell))$.

Notation 11

We denote by $\mathsf{val}(v)$ the value assigned by the solver to the variable v.

Notation 12

$v(b@d)$ denotes that the variable v is assigned the Boolean value $b \in \{0, 1\}$ at decision level d. $v(?@?)$ denotes that v is not assigned under the current partial assignment.
$\ell(b@d)$ denotes that the literal ℓ is assigned the Boolean value $b \in \{0, 1\}$ at decision level d. $\ell(?@?)$ denotes that ℓ is not assigned under the current partial assignment.
In the following, we use either the variable notation or the literal notation, depending on which is more convenient in the context. In particular, we always use the literal notation when dealing with clauses or constraints.

Remark 19

If a literal ℓ is propagated *before* any decision is made, we say that the literal is propagated at decision level 0, and thus write $\ell(1@0)$.

In order to take into account the implicit simplifications made by the solver when it assigns a variable, let us consider a new definition of *unit clause*.

Definition 96 (Unit Clause Under a Partial Assignment)

A clause is said to be *unit under a partial assignment* when it contains exactly one unassigned literal, while all its other literals are falsified by the current partial assignment.

Example 43

The clause $a(?@?) \vee \neg b(0@1) \vee c(0@2)$ is unit under the current partial assignment. The clauses $a(?@?) \vee \neg b(?@?) \vee c(0@2)$, $a(1@2) \vee \neg b(0@1) \vee c(0@2)$ and $a(0@2) \vee \neg b(0@1) \vee c(0@2)$ are not unit under the current partial assignment.

In the following, unit clauses are always considered unit under the current partial assignment. Unit clauses play a key role in modern SAT solvers: it has been estimated that almost 80% of the runtime of these solvers is spent performing unit propagation [MMZ+01]. An efficient algorithm for detecting unit clauses is thus required.

A first approach for detecting whether a clause is unit is to maintain a counter of its falsified literals, so as to know when only one literal remains unassigned. In practice, this technique requires a precise vision of the formula to update the counter each time the assignment of a literal changes, which is not appropriate in the architecture of modern SAT solvers, based on lazy data structures. This has motivated the development of more sophisticated algorithms and lazy data structures for detecting unit clauses, based on the following observation.

Observation 4

If a clause contains at least two non-falsified literals, it cannot be unit.

Intuitively, it is thus enough to keep track, for each clause, of two non-falsified literals. A first approach for doing so is the *head-tail* approach described in [ZS96, ZS00]. In this case, the two literals being considered are the first (*head*) and last (*tail*) non-falsified literals of the clause. These two literals are marked with a pointer that is moved when the head or tail literals become falsified. When it is not possible to update the pointer, the clause is falsified and, when the first and last non-falsified literals are identical, this literal must be propagated as in the following example.

Example 44

Consider the clause $a(?@?) \vee \neg b(?@?) \vee c(?@?)$. Initially, the *head* literal is a and the *tail* literal is c. If b is assigned to 0, nothing changes with respect to the *head* and *tail* literals. Now, if a becomes falsified, a new *head* literal must be identified. The next non-falsified literal in the clause is c, which becomes the new *head* literal. As it is also the *tail* literal, it must be propagated.

While being more efficient than the counter-based approach, as assignments made between the *head* and the *tail* literals do not alter these literals, this approach is not completely satisfactory. In particular, when literals are being unassigned during the exploration of the search space, it may be necessary to update the *head* and *tail* literals. For instance, in the example above, if a becomes unassigned again, the *head* pointer must be updated to point to this literal.

To improve the performance of unit propagation, current solvers mostly use another approach, introduced in [MMZ$^+$01] and known as *watched literals*. This data structure always ensures that the two first literals of the clause are unassigned: whenever one of these watched literals becomes falsified, the solver searches for another non-falsified literal in the clause, which takes the position of the falsified watched literal [MMZ$^+$01, Gel02, ES04]. If the clause does not contain a non-falsified literal that can replace this literal, then the other watched literal must be propagated. This procedure is described in Algorithm 1.

Remark 20

In practice and as described in the algorithm, watched literals are maintained so that the first literal of the clause is always the one to propagate.

Example 45

Consider again the clause $a(?@?) \vee \neg b(?@?) \vee c(?@?)$. Initially, the two watched literals are a and b. If b is assigned to 0, it is replaced by c, giving the clause $a(?@?) \vee c(?@?) \vee \neg b(0@1)$. Now, if a becomes falsified, its position is first switched with that of c, giving $c(?@?) \vee a(0@2) \vee \neg b(0@1)$. Then, the solver searches for a new watched literal to replace a. As no such literal exists, the first literal of the clause (i.e., the first watched literal) must be propagated.

Algorithm 1: updateWatchedLiterals

```
Input  : A clause $\gamma$ with watched literals $\ell_1$ and $\ell_2$ and a falsified literal $\ell$
Output: The literal to propagate, if any
1  if $\ell \neq \ell_1$ and $\ell \neq \ell_2$ then
2      return null
3  end
4  if $\ell = \ell_1$ then
5      Swap $\ell_1$ and $\ell_2$ in $\gamma$
6  end
7  foreach literal $\ell'$ of $\gamma$ that is not watched do
8      if $\ell'$ is not falsified then
9          Swap $\ell_2$ and $\ell'$ in $\gamma$
10         return null
11     end
12 end
13 return $\ell_1$
```

The main advantage of using watched literals is that they are a *lazy* data structure: these literals must be updated only when one of them becomes falsified, and are independent of the assignment of the other literals. In particular, they do not need to be updated when literals become unassigned in the clause (for instance, in the example above, if either a or b become unassigned, the current watched literals, c and a, remain correct).

Remark 21

The *head-tail* procedure described above can be improved by moving the *head* and *tail* literals to the first and last position of the clause, respectively, instead of moving pointers in the clause. Doing so, the *head-tail* and *watched literals* approaches have similar performances.

During unit propagation, it may happen that the same literal is propagated to both true and false at the same decision level. In this case, we say that a *conflict* has occurred. Intuitively, a conflict is due to a sequence of decisions made at some point while exploring the search space. In order to identify the sequence that is responsible for the conflict, the solver performs a *conflict analysis* so as to not run into the same conflict again.

4.1.2 Intelligent Backtracking

During the early development of SAT solvers, several algorithms have been designed to decide the satisfiability of a CNF formula. Among these algorithms is the one by Davis and Putnam [DP60], best known as *DP*, which makes an heavy use of the *resolution* proof system. This proof system is composed of the two following rules.

$$\frac{v \vee \bigvee_{i=1}^{n} \ell_i \qquad \bar{v} \vee \bigvee_{j=1}^{m} \ell'_j}{\bigvee_{i=1}^{n} \ell_i \vee \bigvee_{j=1}^{m} \ell'_j} \text{ (resolution)} \qquad\qquad \frac{\ell \vee \ell \vee \bigvee_{i=1}^{n} \ell_i}{\ell \vee \bigvee_{i=1}^{n} \ell_i} \text{ (merge)}$$

Notation 13

Let us describe the notation used for these rules. The clauses above the horizontal bars are called *premises*. From the premises, we *deduce* (or *derive*) the clauses that is below the horizontal bar. For instance, the first rules reads as: "if we have $v \vee \bigvee_{i=1}^{n} \ell_i$ and $\bar{v} \vee \bigvee_{j=1}^{m} \ell'_j$, then we can infer $\bigvee_{i=1}^{n} \ell_i \vee \bigvee_{j=1}^{m} \ell'_j$". From a logical viewpoint, this first rule can also be read as:

$$\left(v \vee \bigvee_{i=1}^{n} \ell_i\right) \wedge \left(\bar{v} \vee \bigvee_{j=1}^{m} \ell'_j\right) \models \bigvee_{i=1}^{n} \ell_i \vee \bigvee_{j=1}^{m} \ell'_j$$

Remark 22

Note that one can reuse the derived clause as a premise for the application of another rule.

In the following, we call the application of the resolution rule, followed by a systematic application of the *merge* rule, when needed, a *resolution* step. In practice, clauses are often represented as sets of literals, so that the merge rule is is applied implicitly.

Remark 23

Observe that, in the resolution rule above, if more than one literal appear with an opposite sign in the two clauses being resolved, one gets a tautology, as in the following example:

$$\frac{a \vee b \vee c \qquad \neg a \vee \neg b \vee \neg d}{b \vee \neg b \vee c \vee \neg d}$$

The clause derived here is equivalent to $\top$, as it contains both b and $\neg b$, and that every complete assignment satisfies exactly one of these literals.

Notation 14

Let γ and γ' be two clauses containing ℓ and $\neg\ell$, respectively. We denote $\gamma \boxplus \gamma'$ the result of the application of the resolution rule followed by the merge rule (if needed) on the clauses γ and γ'. The resulting clause is called the *resolvent* of γ and γ', and ℓ is called the *pivot* of the resolution.

The resolution proof system is particularly adapted to SAT solving, as it is both *sound* and *refutation complete*.

Definition 97 (Soundness)

A proof system is said to be *sound* if and only if, all formulae derived by this proof system are logical consequences of the conjunction of the original formulae.

Definition 98 (Refutation Completeness)

A proof system is said to be *refutation complete* if and only if, for any conjunction of inconsistent formulae, the rules of this proof system allow to derive $\bot$ from the original formulae. We call the combination of the rules and clauses that yields $\bot$ a *refutation proof* or *unsatisfiability proof*.

It is thus possible to decide the satisfiability of a CNF formula by finding an unsatisfiability proof from its clauses. This can be achieved by computing the forgetting (see Definition 61) of all the variables of the formula. If at some point, an empty clause is derived, the formula is unsatisfiable, otherwise it is satisfiable. This procedure is described by Algorithm 2 below.

Algorithm 2: DP

```
   Input  : A CNF formula $\Sigma$
   Output: Whether $\Sigma$ is satisfiable
1  $\Sigma \leftarrow$ unitPropagate($\Sigma$)
2  if $\Sigma$ contains the empty clause then
3      return UNSATISFIABLE
4  end
5  $\Sigma \leftarrow$ satisfyPureLiterals($\Sigma$)
6  if $\Sigma$ is empty then
7      return SATISFIABLE
8  end
9  $v \leftarrow$ chooseVariable($\Sigma$)
10 foreach clause $\gamma$ in $\Sigma$ containing $v$ do
11     foreach clause $\gamma'$ in $\Sigma$ containing $\neg v$ do
12         $\gamma_r \leftarrow \gamma \boxplus \gamma'$
13         if $\gamma_r \not\equiv \top$ then
14             add $\gamma_r$ to $\Sigma$
15         end
16     end
17 end
18 remove all clauses from $\Sigma$ containing $v$ or $\neg v$
19 return DP($\Sigma$)
```

The algorithm first applies unit propagation on the input formula, and returns that the formula is unsatisfiable when the empty clause is derived during this process, which (implicitly) simplifies the formula. Then, all pure literals are satisfied, so as to simplify the formula. If all clauses are satisfied, then the formula is consistent. Otherwise, a variable v is chosen among those appearing in Σ. In [DP60], the first variable of the shortest clause in Σ is chosen. All possible resolutions using v as pivot are then applied, and the corresponding resolvents are added to Σ. All clauses containing the variable v are then removed from Σ, giving the forgetting of v in Σ. A recursive call to the algorithm repeats all these operations, so as to forget all the variables of Σ, which guarantees the soundness of the algorithm.

While it allows to decide the satisfiability of a CNF formula, the DP algorithm has several drawbacks. First, computing all possible resolvents is both time and space consuming, making the algorithm inefficient in practice, and not scalable to large instances. Second, when the input formula is satisfiable, DP is not able to provide a model of the formula. This has motivated the development of another algorithm, known as *DPLL* [DLL62], which is given by Algorithm 3 below.

Algorithm 3: DPLL

```
Input  : A CNF formula $\Sigma$
Output: Whether $\Sigma$ is satisfiable
1  $\Sigma \leftarrow$ unitPropagate($\Sigma$)
2  if $\Sigma$ contains the empty clause then
3      return UNSATISFIABLE
4  end
5  $\Sigma \leftarrow$ satisfyPureLiterals($\Sigma$)
6  if $\Sigma$ is empty then
7      return SATISFIABLE
8  end
9  $\ell \leftarrow$ chooseLiteral($\Sigma$)
10 if DPLL($\Sigma \wedge \ell$) = SATISFIABLE) then
11     return SATISFIABLE
12 end
13 return DPLL($\Sigma \wedge \neg\ell$)
```

Contrary to *DP*, the *DPLL* procedure does not apply the resolution rule, but relies on so-called *backtrack search*. Once a literal ℓ is chosen, it is first assigned true (note that this is done in the algorithm by applying unit propagation on $\Sigma \wedge \ell$ during the recursive call). If the simplified formula is satisfiable, then the original formula is satisfiable as well (and ℓ appears as satisfied in the model built by the solver), otherwise the literal is assigned false, and a similar check is performed on the resulting formula. This algorithm is less space consuming than DP, as only the current partial assignment changes from one call to another, but also allows to identify a model (actually, an implicant) of the formula, which corresponds to the partial assignment currently stored by the solver when the formula is identified as satisfiable.

Despite these improvements, SAT solvers using the DPLL procedure are still too slow in practice to solve instances containing tens of thousands of variables. One of the main limitations of DPLL is that it can only produce tree-like proofs (while DAG-like proofs, for instance, may be shorter). Moreover, DPLL can only performs *chronological backtracking* (or simply *backtracking*). In this context, when a conflict is due to decisions made at the very beginning of the algorithm, these decisions can only be reconsidered after having explored the whole search space rooted at these decisions, so that the solver may run into the same conflict again and again. In order to identify these decisions earlier, the *Conflict-Driven Clause Learning* (*CDCL*) architecture [MS99] has been developed.

Recall that a conflict occurs when a literal is propagated to both 0 and 1 at the same decision level, or, in practice, when a clause becomes falsified. This may occur in two different contexts. If the conflict is detected at decision level 0, i.e., without having made any decision, then the formula is trivially unsatisfiable. Otherwise, a decision needs to be reconsidered at some point. The purpose of *conflict analysis* is to identify such a decision. To do so, the solver needs to keep track of all decisions and propagations that have produced this conflict with an *implication graph*.

Definition 99 (Implication Graph)

An *implication graph* is a directed acyclic graph such that:

- the vertices of the graph may either be $\top$ or an assignment of a variable $v(\mathsf{val}(v)@\mathsf{dl}(v))$,
- the predecessors of a vertex $v(\mathsf{val}(v)@\mathsf{dl}(v))$ are the assignments (decisions and propagations) that triggered the unit propagation of the value $\mathsf{val}(v)$ for the variable v, or $\top$ if the unit propagation was triggered by a unit clause of the original formula, and
- each edge is labeled by the clause that has triggered the unit propagation it represents.

Remark 24

In practice, the implication graph is only maintained *implicitly* by the solver. The solver exploits an *assignment stack*, also known as *trail*, in which each assignment (decision or propagation) is pushed, together with its reason (if any).

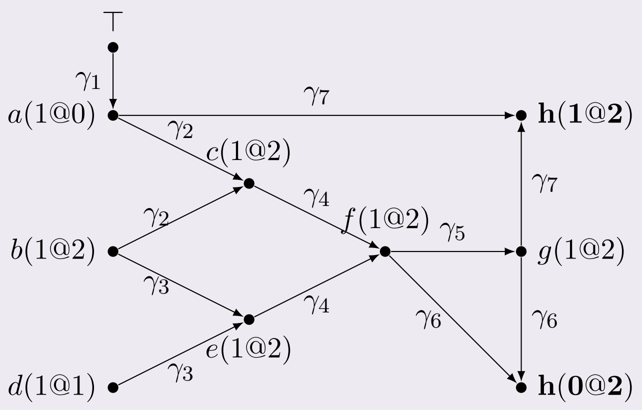
A conflict occurs in the implication graph when two vertices $v(0@\mathsf{dl}(v))$ and $v(1@\mathsf{dl}(v))$ are simultaneously present in the graph, as illustrated in the following example.

Example 46

Consider the CNF formula containing the following clauses:

- $\gamma_1 : a$
- $\gamma_2 : \neg a \vee \neg b \vee c$
- $\gamma_3 : \neg b \vee \neg d \vee e$
- $\gamma_4 : \neg c \vee \neg e \vee f$
- $\gamma_5 : \neg f \vee g$
- $\gamma_6 : \neg a \vee \neg g \vee h$
- $\gamma_7 : \neg f \vee \neg g \vee \neg h$

First, observe that the literal a is propagated at decision level 0, before any assignment is made. Then, suppose that the decisions $d(1@1)$ and $b(1@2)$ are taken. A decision level 2, some unit propagations are triggered, as illustrated in the implication graph below.



Observe that there is a conflict in the graph, as it contains both vertices $h(0@2)$ and $h(1@2)$.

When a conflict is identified, the analysis starts by choosing a clause γ_0 that is falsified by the current assignment. This clause is also referred to as the *conflict*.

Remark 25

From a theoretical viewpoint, the conflict is the propagation of a literal to both 0 and 1 at the same decision level. In practice, however, the conflict is only identified when the (watched) literal to be propagated has already been falsified during unit propagation, meaning that the clause itself is also falsified. This is why we consider here a falsified clause as the conflict.

A conflict analysis is then performed by following a path in the implication graph in a bottom-up fashion, starting from the conflict. The path is chosen by considering the reason for the last propagated variable, by popping the latest assignment from the trail (see Remark 24). A resolution operation is then applied between the conflict and this reason, to produce a new conflicting clause.

Remark 26

By construction, the resolution performed during conflict analysis can never produce tautologies. Indeed, as the conflict only contains falsified literals and the reason contains only one satisfied literal, it is clear that only one literal appears with an opposite sign in the clauses being resolved.

The resolution operations above are applied until the clause that is derived is *assertive*.

Definition 100 (Assertive Clause)

Given a set of decision levels $\{d_0, \ldots, d_n\}$, a clause is said to be *assertive* at decision level d_i $(0 \leq i < n)$ if and only if it is unit under the partial assignment given at decision level d_i.

A mostly accepted approach for producing an assertive clause in current implementation of SAT solvers is to construct a clause that is a *unique implication point* [MS99].

Definition 101 (Unique Implication Point)

Consider a clause γ that is conflicting under the current partial assignment, and a set of decision levels $\{d_0, \ldots, d_n\}$. The clause γ is a *Unique Implication Point* if and only if it contains a single literal assigned at decision level d_n.
In this case, γ is assertive at the decision level d_i such that all the literals in γ are assigned, but the one that is assigned at decision level d_n.

When a unique implication point is reached, it allows to cancel *all* decisions made after the decision level d_i at which the inferred clause is assertive, and thus to perform *non-chronological* backtracking, also known as *backjumping*.

The conflict analysis procedure used to derive a unique implication point is given by Algorithm 4 below.

Algorithm 4: findUIP

```
Input  : A conflicting clause $\gamma_0$ and an implication graph $\Gamma$
Output: A clause that is a unique implication point

1 $\gamma \leftarrow \gamma_0$
2 while $\gamma$ contains more than one literal assigned at the current decision level do
3     choose a literal $\ell$ assigned at the last decision level in $\gamma$
4     $\gamma' \leftarrow \mathsf{reason}(\neg\ell)$
5     $\gamma \leftarrow \gamma \boxplus \gamma'$
6 end
```

Remark 27

A decision is always a UIP, so the procedure stops in the worst case at the last decision that has been taken.

When this algorithm is applied to a conflicting clause γ_0, the clause it returns is guaranteed to be a unique implication point (UIP). In particular, it is the *first-unique implication point*, or *1-UIP*, in the sense that no other UIP could have been produced before. There exists other algorithms, aiming at deriving different UIPs, for instance the *Decision-UIP*, *Last-UIP*, *2-UIP*, *3-UIP* or *All-UIP* [MMZ+01, ZMMM01]. We do not consider these variants here, as most modern SAT solvers implement the *1-UIP* scheme, which is also known to yield the highest possible backjump level (in the sense that it allows to undo a maximum number of decisions) [ABH+08]. This UIP clause, also known as *no-good*, is added to the clause database of the solver: we say that the solver *learns* this clause.

The main advantage of clause learning is that it allows to perform backjumps which, compared to chronological backtracking, make it possible to reconsider decisions that were made early during the exploration of the search space, without having to explore the whole search space they define, as they may allow to undo multiple decisions at the same time. This is illustrated in the example below.

Example 47 (Example 46 cont'd)

In Example 46, the clause γ_7 is falsified. Let us consider this clause as γ_0 in the algorithm. γ_0 is not assertive, as the literals $\neg g$ and $\neg h$ are assigned at the current decision level. We thus apply the resolution rule until only one literal in the conflicting clause is assigned at decision level 2. First, we resolve the conflicting clause γ_0 with the reason for h, i.e., γ_6.

$$\frac{\gamma_0 \qquad \gamma_6}{\neg a(0@0) \vee \neg f(0@2) \vee \neg g(0@2)}$$

The resolvent we obtain is a new clause, that is still conflicting. It is not assertive yet, so we apply a resolution step between this new conflict and the reason for g, i.e., γ_5.

$$\frac{\neg a(0@0) \vee \neg f(0@2) \vee \neg g(0@2) \qquad \gamma_5}{\neg a(0@0) \vee \neg f(0@2)}$$

The clause $\neg a(0@0) \lor \neg f(0@2)$ that is derived is now assertive, as it only contains one literal that is falsified at the current decision level, namely $\neg f$. This literal is thus propagated at decision level 0 (i.e., the one at which $\neg a$ has been falsified). Hence, this clause is learned, and a backjump to decision level 0 is performed (i.e., all literals assigned at decision level 1 and 2 become now unassigned).

Algorithm 5 summarizes the algorithm used to determine the satisfiability of a CNF formula using the CDCL approach.

Algorithm 5: CDCL

```
Input  : A CNF formula $\Sigma$
Output: Whether $\Sigma$ is satisfiable
decisionLevel ← 0
while true do
    $\Sigma$ ← unitPropagate($\Sigma$)
    if $\Sigma$ contains a conflicting clause $\gamma_0$ then
        if decisionLevel = 0 then
            return UNSATISFIABLE
        else
            $\gamma_l$ ← findUIP($\gamma_0$)
            $\Sigma$ ← $\Sigma \land \gamma_l$
            backtrackLevel ← assertionLevel($\gamma_l$)
            backjumpTo(backtrackLevel)
            decisionLevel ← backtrackLevel
        end
    else
        $\ell$ ← chooseLiteral($\Sigma$)
        if $\ell$ = null then
            return SATISFIABLE
        end
        decisionLevel ← decisionLevel + 1
        $\ell$ ← 1
    end
end
```

Algorithm 5 works as follows. First, unit propagation is applied to the input formula. If a conflict is detected at level 0, then the formula is clearly unsatisfiable, as no decision has been taken. Otherwise, if a conflict is detected at another decision level, it is analyzed to derive a UIP clause (see Algorithm 4) that is added to the clause database. A backjump to the decision level at which the UIP clause is assertive is then performed, so as to forget all decisions taken after this decision level. Then, the loop continues. Note that, as the clause is assertive, the first unit propagation necessarily propagates a literal which was not propagated before at this decision level. This guarantees that the solver does not explore the same part of the search space. If no conflict is encountered during unit propagation, then a new literal is assigned to true to explore a subpart of the search space (unless all literals are already assigned, in which case a model of the formula has been found). Note that the literal is not assigned to 0 in case of a failure of the decision: instead, the CDCL algorithm relies on the learned clauses to explore another part of the search space.

In practice, the conflict analysis procedure described above is not enough to make CDCL SAT solvers efficient. Indeed, modern SAT solvers also use a number of internal heuristics and features that guide the search towards a solution or an unsatisfiability proof.

4.1.3 Guiding the Search

In addition to clause learning, the CDCL architecture comes with many strategies that help the solver find its way through the search space. It is commonly accepted that, without these strategies, modern SAT solvers become very bad [EGG$^+$18]. Particularly important are the decision heuristic, but also other features like learned clause deletion and restarts. This section describes their purposes and their implementations in SAT solvers.

Branching Heuristics

An important component of a SAT solver is its *branching heuristic*: to efficiently find a solution or an unsatisfiability proof, the solver has to choose the *right* variables on which to make decisions. With the initial development of DPLL-based solvers, a number of heuristics considering the number of occurrences of a literal have been designed. This is for example the case of the *Bohm*'s heuristic [BB92], the *maximum occurrences on clauses of minimum size* (*MOM's*) heuristics [DABC93, Fre95] or its variant known as *Jeroslow-Wang* [JW90], and also many others described, for instance, in [Mar99]. These heuristics require a *complete* view of the input formula, as they need to know the assignment of each literal and the clauses that are satisfied. As such, they do not fit well in modern SAT solvers, which mostly rely on *lazy* data structures.

To be efficient, SAT solvers thus need *lazy* branching heuristics. This has motivated the development of the *variable state independent decaying sum* (VSIDS) heuristic [MMZ$^+$01], on which many heuristics implemented in modern SAT solvers are based. In this heuristic, each variable is assigned a *score* that is incremented each time a new clause containing this variable is learned. Additionally, variable scores are regularly divided by 2 (in practice, every 256 conflicts), so as to favor variables appearing in the most recent learned clauses (this operation is called *rescoring*). When it comes to selecting a variable, the solver chooses the variable with the highest score.

The most popular variant of VSIDS is *exponential VSIDS* (EVSIDS), introduced in *MiniSat* [ES04]. In this heuristic, a value g is chosen between 1.01 and 1.2 at the beginning of the execution of the solver. When a variable is encountered during the analysis of the i-th conflict, the score of this variable is updated by adding g^i to its current score. Such an update is called *variable bumping*. It preserves the property of favoring variables appearing in recent conflicts while avoiding the cost of a regular rescoring. Moreover, modern implementations of VSIDS not only update the score of variables appearing in the learned clauses, but also that of variables appearing in all clauses used to produce them. This approach aims to favor the selection of variables that are *involved* in recent conflicts. To this aim, some solvers such as *Sat4j* [LP10] even bump all literals encountered during conflict analysis *each time* they are encountered, while *MiniSat* [ES04] for instance bumps them only once per conflict.

Another heuristic aiming to favor recently used variables is known as *variable move to front* (VMTF) [Rya04]. Here, the score of a variable becomes the index of the latest conflict in which the variable was involved.

Over the years, many other heuristics based on VSIDS have been developed (see, e.g., [BF15] for an overview). More recently, new heuristics have been introduced, such as that known as *learning rate branching* (LRB) [LGPC16]. This heuristic uses machine learning techniques (concretely, multi-armed bandits) so as to predict which variables will be present in more learnt clauses, and to branch on such variables.

The branching heuristics described above allow to decide on which *variable* the next decision should be taken. In order to decide which *truth value* should be assigned to this variable, most SAT solvers use the so-called *phase saving* heuristic. Typically, the solver assigns the variable to the last truth value (phase) to which it has been propagated [PD07]. More recently, it has been proposed to consider instead the *trend* of the phases propagated for a literal, by aggregating all propagations instead of only keeping the latest. This is achieved through the *decaying polarity score* (DPS) and the generalization of VSIDS to literals with the *literal state independent decaying sum* (LSIDS) [SM20].

Deleting Learned Clauses

Each time a conflict is encountered in a SAT solver, a new clause is learned. This allows the solver to be efficient in practice, but it has also a drawback: many conflicts may occur during the search, so that the number of clauses in the clause database may become very large, which may both increase memory usage and slow down unit propagation, as literals may be watched in many clauses. This is why, since the early development of CDCL solvers, the management of learned clauses has been considered. In particular, different techniques have been developed to *delete* clauses that have been learned.

First, one needs to consider *when* to delete clauses. A possibility is not to keep the clause at all, and simply use it to perform the backjump, as for instance in [Gin93] or as *Lingeling* [Bie16] does for long learned clauses. Typically, in such cases, the clause is just used as a reason for the propagation of the literal after the backjump, but not added to the clause database, so that it can only be used during conflict analysis and not for propagating other literals later on. Another approach is limiting the number of clauses in the database, and to delete some of them when the limit is reached, while potentially changing this limit during the execution of the solver [MMZ$^+$01, ES04, Bie16].

Furthermore, one also needs to decide *which* clauses should be deleted. A first approach is to consider the size of the learned clauses. Intuitively long clauses containing many literals take much space and are weak, especially from a propagation viewpoint: many literals need to be falsified before the clause can trigger a unit propagation. However, deleting such clauses may also prevent the solver from finding an unsatisfiability proof: long clauses may be required to derive inconsistency [BJ10].

Another approach is considering the age of the learned clauses, i.e., remove first the clauses that were the first to be learned. This strategy is however not satisfactory, as it does not take into account the role played by the clause in recent conflicts. This is why an activity based approach has been designed in *MiniSat* [ES04]: similarly to the variables, each learned clause has a score that is incremented each time the clause is encountered during conflict analysis. When removing clauses, clauses with high activity, i.e., those that were involved in recent conflicts, are preferred over those with low activity, so the latter are deleted first.

Later on, a quality measure known as *literal block distance* (LBD) [AS09] has been introduced.

Definition 102 (Literal Block Distance)

Consider a clause γ and the current assignment of its literals. Let π be a partition of these literals, such that literals are partitioned with respect to their decision levels. The LBD of γ is the number of elements in π.

The LBD is first computed when the clause is learned, and is then updated each time the clause propagates a literal. When clause deletion is performed, clauses having a high LBD are removed before those with low LBD.

Restarting the Search

Restarting the search is a feature that was first introduced in [GSK98] to avoid the so-called *heavy-tail phenomenon*, i.e., the non-negligible probability that exploring a subpart of the search space may take exponentially more time than the exploration of all previously explored subparts. Intuitively, restarts allow avoiding that decisions taken at the beginning of the search keep the solver stuck in a part of the search space. Modern SAT solvers all implement restarts, and it is commonly accepted that this feature is a major component without which the solver has poor performances [EGG+18, PD11, VEG+18, AFT11]. However, the notion of restarts is not currently completely understood.

In practice, restarting consists in forgetting all decisions made by the solver, then going back to the root decision level, as a backjump to decision level 0. We call the execution of the solver between two restarts a *run*. It is worth noting that the solver does not completely reset itself when performing a restart as learned clauses, variable scores and saved phases are kept from one run to another. This ensures that restarting the search will allow to explore another subpart of the search space. However, when a solver implements both restart and clause deletion, it may theoretically never terminate for certain instances, especially if run lengths are not guaranteed to increase arbitrarily.

A key aspect when implementing restarts is to detect *when* the solver should perform a restart. A first approach are static restarts, in which restarts are generally triggered after a given number of conflicts, that fixes the *length* of a run. This number may either be fixed once for the whole execution of the solver, or be modified during the execution. This is for example the case of *MiniSat* [ES04], which performs restarts based on a *geometric function*: the first length is fixed to a number N and the following lengths are increased by a factor of 1.5.

The main disadvantage of this restart policy is that it only triggers few restarts, while it is often preferred to make *frequent* restarts. This has motivated the use of another static restart policy, based on the notion of *reluctant doubling* and the Luby series [LSZ93, Hua07]:

$$\text{luby}(i) = \begin{cases} 2^{k-1} & \text{if } i = 2^k - 1 \\ \text{luby}(i - 2^{k-1} + 1) & \text{if } 2^{k-1} \leq i < 2^k - 1 \end{cases}$$

This sequence is better described by Donald E. Knuth [Knu16, Section 7.2.2.2] as follows:

> “ *The elements of this sequence are all powers of 2. Furthermore we have luby*$(i+1) = 2 \times$ *luby*(i) *if the number luby*(i) *has already occurred an even number of times, otherwise luby*$(i) = 1$. ”
>
> Donald E. Knuth

Similarly to the geometric approach described above, this restart policy starts by making a run of length N, and then makes runs of length $N \times \text{luby}(i)$, where i is the index of the current run.

Another static restart policy that is designed to trigger frequent restarts is that in the original implementation of *PicoSAT* [Bie08b]. This strategy performs so-called *inner* restarts, following a geometric function on the number of encountered conflict. In addition to inner restarts, *outer* restarts are performed after a certain number of restarts (following a geometric function of the number of restarts) to reset the number of conflicts before triggering a restart to the initial one.

More recently, *dynamic* restart policies have been developed, in order to consider the current state of the solver to decide whether a restart should be performed. This is for example the case of the restart policy implemented in a more recent version of *PicoSAT*, based on the *average number of recently flipped assignments* (ANRFA) [Bie08a]. In this case, a global measure is used to evaluate the agility of the solver, based on the number of recent *flips*. A *flip* is defined as the propagation of a truth value to a

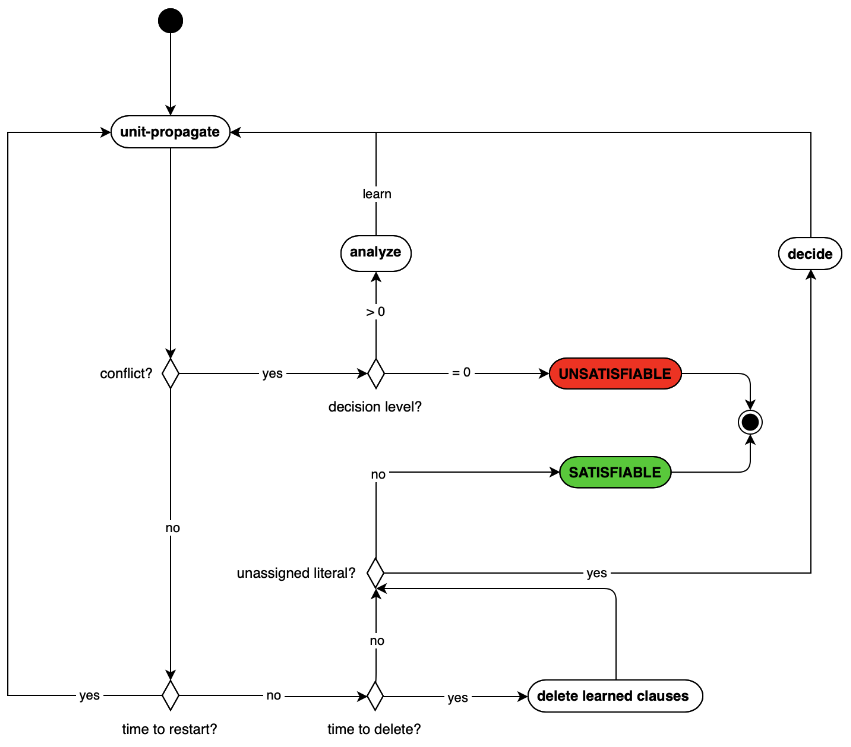


Figure 4.1: Activity diagram representing the CDCL algorithm.

variable that is the opposite of the previous propagated value for this variable. Intuitively, the agility is high when many flips occur. When the agility becomes too low, a restart is performed.

Another dynamic restart policy is the one introduced by *Glucose* [AS12], in which the decision of whether a restart should be performed depends on the *quality* of the constraints that are currently being learned: when this quality decreases, the solver is most likely exploring the wrong search space. In Glucose, the learned clause quality is measured with their LBD (see Definition 102). To measure the decrease in the quality of learned clauses, the average LBD is computed over the most recent learned clauses (in practice, the last 100 clauses). Whenever this average is greater than 70% of the average LBD computed over all learned clauses, a restart should be performed. Glucose also implements a feature known as *restart blocking*: whenever the solver *seems* close to find a complete satisfying, no restart can be triggered to improve the performance of the solver on satisfiable instances. This feature is designed so that, similarly to the way restarts are triggered, when the number of currently assigned variables is above the average number of assigned variables when the previous restarts were performed, restarts are blocked.

Combined with the deletion of learned clauses we presented above, the restart feature is integrated in the CDCL architecture to get the algorithm that is summarized in Figure 4.1.

Even though current SAT solvers allow to exploit more power of the resolution proof system compared to their older implementations, they are not always powerful enough to find short unsatisfiability proofs. In particular, instances that are hard for resolution, as for instance pigeonhole-principle formulae [Hak85], are necessarily hard for resolution-based SAT solvers (simply denoted as SAT solvers in the following). For such problems, considering a different proof system may allow to improve the performance of the solver.

4.2 Pseudo-Boolean Solving Based on Cutting Planes

As an alternative to the weak resolution proof system, pseudo-Boolean solvers may use the stronger *cutting planes* proof system to derive new constraints during conflict analysis. In particular, unsatisfiability proofs of the cutting planes proof system can be exponentially shorter than their resolution counterpart. This section studies how the cutting planes proof system may be implemented in pseudo-Boolean SAT solvers (or simply pseudo-Boolean solvers, as opposed to the SAT solvers based on the resolution proof system described in the previous section) to extend the CDCL architecture so as to deal with pseudo-Boolean constraints.

4.2.1 Detecting Propagations

As in SAT solvers, pseudo-Boolean solvers explore the search space by taking decisions and propagating assignments. However, propagations in pseudo-Boolean and cardinality constraints are slightly different compared to those in clauses. In particular, such constraints may propagate multiple literals at the same decision level, as in the following example.

Example 48

The cardinality constraint $a(?@?) + b(?@?) + c(0@1) \geq 2$ propagates both a and b under the current partial assignment.

Moreover, pseudo-Boolean constraints may trigger propagations multiple times, and even when some other literals are not assigned yet, as illustrated below. Additionally, a pseudo-Boolean constraint may also become conflicting after having triggered a propagation at an earlier decision level.

Example 49

The pseudo-Boolean constraint $8a(?@?) + 2b(?@?) + c(?@?) + d(?@?) \geq 10$ propagates the literal a at decision level 0 (i.e., before making any decision). Later on, if d becomes falsified, say at decision level 3, then the constraint $8a(1@0) + 2b(?@?) + c(?@?) + d(0@3) \geq 10$ propagates b under the current partial assignment.

These observations make the detection of propagations in pseudo-Boolean constraints harder, so that we need a new definition of assertivity, based on the *slack* of a pseudo-Boolean constraint [CK05, DG02].

Definition 103 (Slack)

Let χ be the pseudo-Boolean constraint given by $\sum_{i=1}^{n} \alpha_i \ell_i \geq \delta$, the *slack* of χ under the current partial assignment is the value:

$$\mathsf{slack}(\chi) = \sum_{i=1, \ell_i \neq 0}^{n} \alpha_i - \delta$$

Remark 28

The slack of a pseudo-Boolean constraint is also referred to as *poss* or *possible* in the literature (e.g., in [DG02]).

Remark 29

In the definition above, it is possible to consider the slack independently of the current assignment, by computing the sum of the coefficients of all literals. In this case, we talk about the *absolute* slack.

Example 50 (Example 49 cont'd)

The slack of the constraint $8a(1@0) + 2b(?@?) + c(?@?) + d(0@3) \geq 10$ is 1. The absolute slack of this constraint is 2.

Thanks to the slack of a constraint, it is in particular possible to detect whether a constraint is assertive or conflicting, based on the following observation.

Observation 5

Let χ be a pseudo-Boolean constraint. If $\mathsf{slack}(\chi) < 0$, then the constraint χ is conflicting under the current partial assignment.

The observation above is quite clear: intuitively, the slack is negative when the sum of the coefficients of all non-falsified literals is lower than the degree, and in this case, the constraint cannot be satisfied. More generally, this observation also allows to provide a new definition of assertivity, which takes into account the case of pseudo-Boolean constraints.

Definition 104 (Assertive Constraint)

A pseudo-Boolean constraint χ is said to be *assertive* if it contains an unassigned literal ℓ with coefficient α such that $\alpha > \mathsf{slack}(\chi)$. In this case, the literal ℓ is propagated by the constraint under the current partial assignment.

The notion of assertivity above is well-defined: indeed, observe that, when $\mathsf{slack}(\chi) < \alpha$, then $\mathsf{slack}(\chi) - \alpha < 0$, so that the constraint becomes falsified if the literal ℓ is falsified. In this case, ℓ must thus be propagated to prevent the constraint from being falsified. The following example illustrates how the definition above can be applied to detect propagations.

Example 51 (Example 50 cont'd)

As the constraint $8a(?@?) + 2b(?@?) + c(?@?) + d(?@?) \geq 10$ has slack 2, it propagates a since this literal is unassigned and has coefficient $8 > 2$.
If we now consider $8a(1@0) + 2b(?@?) + c(?@?) + d(0@3) \geq 10$, this constraint has slack 1, so that it propagates b as this literal is unassigned and has coefficient $2 > 1$. Note that a is not propagated as it is already satisfied.

Thanks to the slack of a constraint, it is thus possible to detect when a constraint becomes conflicting. This is for example the default approach used by *Sat4j* [LP10] for detecting propagations in pseudo-Boolean constraints, and also in solvers such as *Galena* [CK05] for instance. However, this approach is similar in the spirit to the counter-based algorithm for detecting propagations in clauses, since maintaining the slack requires an update each time a literal is assigned or unassigned in the constraint, for instance during backtracking. This is why different watched literal schemes extending that used in SAT solvers have been pointed out. In particular, the following observation illustrates how to implement watched literals for cardinality constraints.

Observation 6

A cardinality constraint of degree δ containing at least $\delta + 1$ unfalsified literals cannot be assertive.

Thanks to this observation, it is possible to generalize the notion of watched literals, so that any cardinality constraint of degree δ has $\delta + 1$ watched literals. Note that this is indeed a generalization of the watched literals in clauses, as a clause may be seen as a cardinality constraint of degree 1.

If we now consider a pseudo-Boolean constraint, it is also possible to design a lazy approach inspired by the watched literals used in clauses or cardinality constraints, but they are trickier to maintain. To design such a data structure, we can take advantage of the following observation.

Observation 7

If there exists a set W of literals in a pseudo-Boolean constraint $\sum_{i=1}^{n} \alpha_i \ell_i \geq \delta$ such that all the literals in W are non-falsified and the sum of their coefficients is greater than $\delta + \alpha_{max}$, where α_{max} denotes the greatest coefficient of an unassigned literal in the constraint, then the constraint is not assertive.

Intuitively, literals in W are those being watched, and their coefficients are used to compute a lower bound of the slack of the constraint. Based on this observation, we can design an algorithm to maintain a set of watched literals for a pseudo-Boolean constraint.

This algorithm is similar to Algorithm 1 used to update the watched literals of a clause, except that it requires to identify multiple watched literals to ensure that the sum of their coefficients guarantees that there is no literal to propagate. Moreover, observe that, each time a conflict is identified when looking for watched literals, the falsified literal is added back to the set of watched literals. This ensures to preserve the property that watched literals do not require to be updated during backjumps. In practice, the falsified literal may also be added back during backjumps, to ensure to watch as few literals as possible.

Algorithm 6: updateWatchedLiteralsPB

Input : A pseudo-Boolean constraint χ given by $\sum_{i=1}^{n} \alpha_i \ell_i \geq \delta$ with its set W of watched literals and a falsified literal $\ell \in W$

Output: A set of literals to propagate, or $CONFLICT$ if a conflict is detected.

```
1  $W \leftarrow W \backslash \{\ell\}$
2  $S \leftarrow \sum_{i=1, \ell_i \in W}^{n} \alpha_i$
3  $V \leftarrow \{\ell_i | \ell_i \neq 0$ and $\ell_i \notin W\}$
4  $\alpha_{max} \leftarrow \max\{\alpha_i | \ell_i$is unassigned$\}$
5  while $S < \delta + \alpha_{max}$ and $V \neq \emptyset$ do
6      $\alpha_s \leftarrow \max\{\alpha_i | \ell_i \in V\}$
7      $S \leftarrow S + \alpha_s$
8      $W \leftarrow W \cup \{\ell_s\}$
9      $V \leftarrow W \backslash \{\ell_s\}$
10 end
11 if $S < \delta$ then
12     $W \leftarrow \ell$
13     return CONFLICT
14 end
15 $P \leftarrow \emptyset$
16 $m \leftarrow S - \delta$
17 foreach literal $\ell_i$ in $W$ do
18     if $m < \alpha_i$ then
19         if $\ell_i$ is not falsified then
20             $P \leftarrow P \cup \{\ell_i\}$
21         end
22         else
23             $W \leftarrow \ell$
24             return CONFLICT
25         end
26     end
27 end
28 return $P$
```

Example 52

Let us consider the constraint $5a + 2b + 2c + 2d + 2e + f \geq 6$. At the beginning, α_{max} is equal to 5, so that the sum of the coefficients of the literals to watch must be at least equal to $5 + 6 = 11$. The watched literals are thus a, b, c and d.
Now, if a becomes falsified, the new α_{max} is equal to 2, and the sum of watched literals must be at least equal to 8. We thus now watch the literals b, c, d and e.
When b becomes falsified, α_{max} is still equal to 2, so we now watch c, d, e and f. However, when watching these literals, the sum of their coefficients is only equal to 7, while 8 is needed. As such, all literals having a coefficient that is greater than $8 - 6 = 2$ must be propagated, i.e., c, d and e.
Moreover, in order to make sure that the watched literals remain valid after a future backjump, the last removed watched literal (i.e., b) remains watched, even if it is currently falsified.

The main disadvantage of the procedure described in Algorithm 6 is that one constantly needs to retrieve the value of α_{max} when updating watched literals. This may be costly if the constraint contains many literals. To avoid the cost of recomputing α_{max}, a conservative approach has been proposed in both *Galena* [CK05] and *Pueblo* [SS06]. It consists in considering as α_{max} the largest coefficient of the constraint, independently of the current assignment.

Example 53

If we consider again the constraint $5a + 2b + 2c + 2d + 2e + f \geq 6$, the value of α_{max} is always equal to 5, so that the sum of the coefficients of the literals to watch must be at least equal to $5 + 6 = 11$ *whatever the current assignment*. The watched literals are initially set to a, b, c and d, as previously.
Now, if a becomes falsified, α_{max} is not updated, and we still need to find a set of watched literals so that the sum of their coefficients is at least 11. In this case, we thus need to watch all the literals b, c, d, e and f, for which the sum of the coefficients is equal to 9.
First, observe that more literals than needed are watched (actually, in this case, it would have been enough to only watch b, c, d and e, as in the previous example). Still, as $9 - 6 = 3$, there is no literal to propagate, since all literals have a coefficient that is strictly less than 3.
As before, the literal a remains watched to ensure that no update will be required on backtrack.

This approach may however require to watch more literals than needed, and thus to perform additional checks when looking for propagations, made in the loop that computes the set of propagated literals. If the exact α_{max} were used, the corresponding literal should be propagated, and α_{max} recomputed.

Let us compare the performance of the different algorithms used to detect propagations as presented in this section. To do so, we ran different configurations of the pseudo-Boolean solver *Sat4j* [LP10] on the whole set of decision benchmarks containing only "small" integers collected during all pseudo-Boolean evaluations since the first edition [MR06]. Experiments have been run on a cluster equipped with bi-processors quadcore Intel XEON X5550 (2.66 GHz, 8 MB cache) and 32 GB of RAM (for more details, see Appendix B). This experimental setting is referred to as the "usual setting" in the rest of this document. The results are given in Figure 4.2.

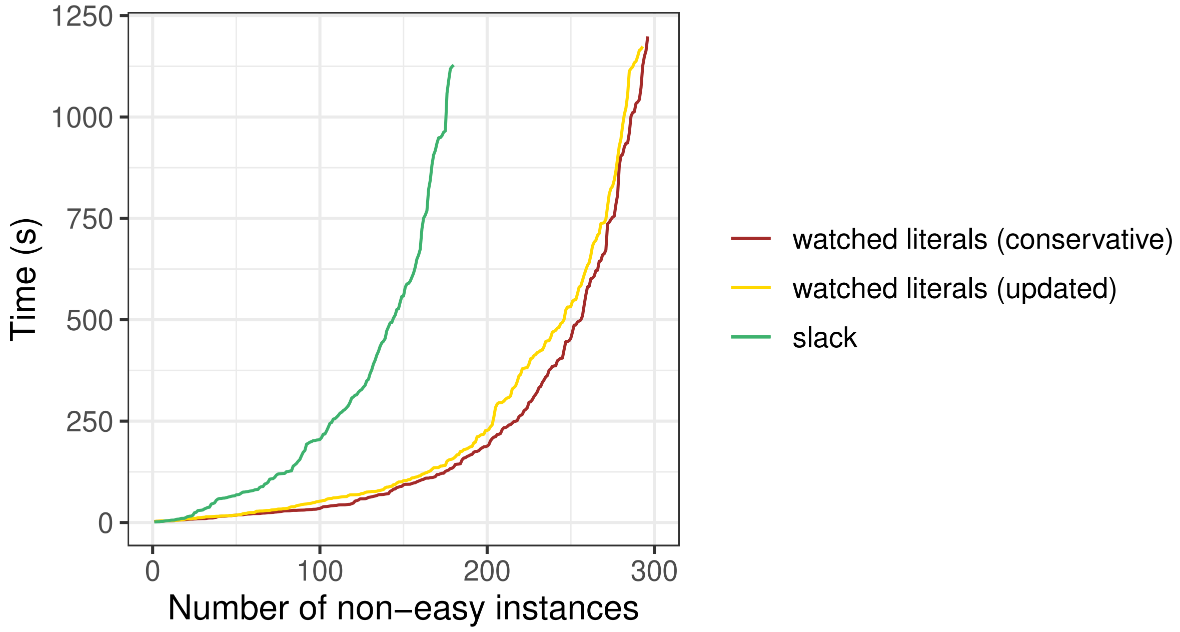


Figure 4.2: Cactus plot comparing the different algorithms for detecting propagations in the pseudo-Boolean solver *Sat4j-GeneralizedResolution*.

This figure is a so-called *cactus plot*: each line corresponds to a solver and represents the distribution of the runtime (in seconds) of this solver on the execution of all considered benchmarks. More precisely, for a fixed runtime (on the y axis), the line gives the number of instances (on the x axis) that are solved within this runtime. In particular, the more to the right the line in plot, the better the solver (in terms of number of solved instances). Moreover, for more readability, this figure only shows the results on *non-easy* instances, i.e., instances for which at least one of the approaches took more than 1 minute to get a result.

On the cactus plot, we can see that the approaches based on watched literals are clearly faster than that based on the slack in the case of *Sat4j-GeneralizedResolution*. In both cases, we note, as also mentioned in [SS06], that the difference of performance between the watched literal-based strategies is not really significant. In the case of *Sat4j* [LP10], the default approach for detecting propagations is the slack-based approach (as in [CK05]) for general pseudo-Boolean constraints, while the original *RoundingSat* [EN18] uses watched literals with a conservative value of α_{max}. For cardinality constraints and clauses, both solvers used watched literals.

In a more recent version of *RoundingSat* [Dev20], the watched literal scheme has been improved with the *opt-watch* algorithm. This approach relies on different optimizations. In particular, to avoid watching too many literals after backjumps, no replacement literals are looked for when an excess watched literal becomes falsified (recall that watched literals may contain more literals than necessary to avoid updates during backjumps). Moreover, the index of watched literals is kept, instead of having a set of watched literals, so as to make sure that coefficients remain sorted to allow an efficient detection of the literals to propagate.

Example 54 (Example 53 cont'd)

Consider again the constraint $5a + 2b + 2c + 2d + 2e + f \geq 6$. In the previous example, a was falsified and all the literals were watched. If a backtrack occurs and all literals in the constraint become unassigned, they all remain watched.
However, it would be enough to only watch a, b, c and d. As no update is performed on watched literals, all literals remain watched. Instead, if f is falsified then, the algorithm detects that enough literals are already watched (the sum of their coefficients is 9, and thus already greater than 8), and f is simply unwatched.

Whether they use a slack-based or an algorithm based on watched literals for detecting propagations, pseudo-Boolean solvers can make decisions and apply Boolean constraint propagation until they find a solution or identify a conflict. In the latter case, a conflict analysis procedure, similar to that implemented in modern SAT solvers, allows to learn new pseudo-Boolean constraints, in a conflict-driven *constraint* learning architecture.

4.2.2 Analyzing Conflicts with Cutting Planes

Similarly to the use of the resolution proof system in SAT solvers, pseudo-Boolean solvers use the *cutting planes* proof system during conflict analysis. This proof system has the following axioms:

$$\frac{}{0 \leq x \leq 1}\text{(bounds)} \qquad\qquad \frac{}{\bar{x} = 1 - x}\text{(negation)}$$

The cutting planes proof system also defines the following main inference rules:

$$\frac{\sum_{i=1}^{n} \alpha_i \ell_i \geq \delta \qquad \sum_{i=1}^{n'} \beta_i \ell'_i \geq \delta'}{\sum_{i=1}^{n} \alpha_i \ell_i + \sum_{i=1}^{n'} \beta_i \ell'_i \geq \delta + \delta'}\text{(addition)}$$

$$\frac{\sum_{i=1}^{n} \alpha_i \ell_i \geq \delta \qquad \lambda \in \mathbb{N}^*}{\sum_{i=1}^{n} \lambda \alpha_i \ell_i \geq \lambda \delta}\text{(multiplication)}$$

Notation 15

Given a pseudo-Boolean constraint χ and an integer λ, we denote by $\lambda\chi$ the result of multiplying the constraint χ by λ, as defined in the multiplication rule above.

Notation 16

Let $x \in \mathbb{R}$. The value $\lceil x \rceil$ denotes the unique integer r such that $r - 1 < x \leq r$.

$$\frac{\sum_{i=1}^{n} \alpha_i \ell_i \geq \delta \qquad \rho \in \mathbb{N}^*}{\sum_{i=1}^{n} \lceil \frac{\alpha_i}{\rho} \rceil \ell_i \geq \lceil \frac{\delta}{\rho} \rceil}\text{(division)}$$

Remark 30

The constraint that is derived in the division rule above is not equivalent to the original constraint in general. However, equivalence is preserved as long as ρ is a divisor of all the coefficients of the constraints. Note that, in this context, the degree does not need to be divisible by ρ.
This property is part of what makes cutting planes more powerful than resolution [CCT87]. For example, the constraint $2x + 2y + 2z \geq 3$ becomes $x + y + z \geq 2$, which has less rational solutions.

The rules defined above are often combined to define new rules used to form sound and refutation complete subsets of the cutting planes proof system.

A popular subset of cutting planes is the *generalized resolution* proof system [Hoo88], consisting of the two following rules: saturation and cancellation.

$$\frac{\sum_{i=1}^{n} \alpha_i \ell_i \geq \delta}{\sum_{i=1}^{n} \min(\alpha_i, \delta) \ell_i \geq \delta} \text{(saturation)}$$

Notation 17

Given a constraint χ, we denote by $\mathsf{saturation}(\chi)$ the application of the cancellation rule on χ.

$$\frac{\alpha\ell + \sum_{i=1}^{n} \alpha_i \ell_i \geq \delta \qquad \beta\bar{\ell} + \sum_{i=1}^{n'} \beta_i \ell'_i \geq \delta' \qquad \rho, \rho' \in \mathbb{N}^* \qquad \rho\alpha = \rho'\beta}{\sum_{i=1}^{n} \rho\alpha_i \ell_i + \sum_{i=1}^{n'} \rho'\beta_i \ell'_i \geq \rho\delta + \rho'\delta' - \rho\alpha} \text{(cancellation)}$$

Notation 18

Let χ and χ' be two pseudo-Boolean constraints containing ℓ and $\bar{\ell}$, respectively. We denote by $\rho\chi \boxplus \rho'\chi'$ the result of the application of the cancellation rule described above, followed by the saturation rule (if needed).

Note that, most of the time, the value of ρ and ρ' are chosen in the cancellation rule so as to minimize the value of $\rho\alpha = \rho'\beta$. In particular, we often choose $\rho = \text{lcm}(\alpha, \beta)/\alpha$ and $\rho' = \text{lcm}(\alpha, \beta)/\beta$, where lcm denotes the least common multiple of two integers.

An important observation is that the cancellation rule above is not enough to ensure the completeness of the proof system, which is why the saturation rule is required.

Example 55

Consider the inconsistent CNF formula $(a \vee b) \wedge (a \vee \neg b) \wedge (\neg a \vee b) \wedge (\neg a \vee \neg b)$, which is equivalent to the pseudo-Boolean formula:

$$(a + b \geq 1) \wedge (a + \bar{b} \geq 1) \wedge (\bar{a} + b \geq 1) \wedge (\bar{a} + \bar{b} \geq 1)$$

If the only allowed rule is the cancellation rule, then it is not possible to prove the inconsistency of this formula.

Remark 31

Intuitively, the reason why saturation is needed for completeness is that the cancellation rule is also sound over the reals. Saturation allows to take into account that variables are integral.

Two other useful rules, also obtained by combining the three main rules of the cutting planes proof system and the axioms, are the weakening rules, given below.

$$\frac{\alpha\ell + \sum_{i=1}^{n} \alpha_i \ell_i \geq \delta \qquad \alpha \in \mathbb{N}}{\sum_{i=1}^{n} \alpha_i \ell_i \geq \delta - \alpha} \text{(weakening)}$$

$$\frac{\alpha\ell + \sum_{i=1}^{n} \alpha_i \ell_i \geq \delta \qquad \varepsilon \in \mathbb{N} \qquad 0 \leq \varepsilon < \alpha}{(\alpha - \varepsilon)\ell + \sum_{i=1}^{n} \alpha_i \ell_i \geq \delta - \varepsilon} \text{(partial weakening)}$$

Notation 19

Given a constraint χ and a literal ℓ, we denote by $\mathsf{weaken}(\ell, \chi)$ the application of the weakening rule on χ that weakens away the literal ℓ.

The main advantage of using the cutting planes proof system is that it *p-simulates* the resolution proof system, but the converse does not hold. Intuitively, this means that for every unsatisfiability proof produced by the resolution proof system, there exists an unsatisfiability proof of polynomial size (in terms of number of derivation steps) in the cutting planes proof system. In this case, the resolution rule and the merge rule are particular cases of the cancellation rule and of the saturation rule, respectively.

$$\frac{\ell + \sum_{i=1}^{n} \ell_i \geq 1 \qquad \bar{\ell} + \sum_{i=1}^{n'} \ell'_i \geq 1}{\sum_{i=1}^{n} \ell_i + \sum_{i=1}^{n'} \ell'_i \geq 1 + 1 - 1 = 1} \text{(resolution)}$$

$$\frac{\sum_{i=1}^{n} \alpha_i \ell_i \geq 1}{\sum_{i=1}^{n} \min(\alpha_i, 1)\ell_i \geq 1} \text{(merge)}$$

In theory, the cutting planes proof system is thus strictly stronger than the resolution proof system, in the sense that the former allows to find shorter unsatisfiability proofs than the latter.

Example 56

Let us consider a pigeonhole principle formula, which states that n pigeons cannot fit into $n-1$ holes. There exists an unsatisfiability proof of linear size in the cutting planes proof system, while only exponential-sized proofs exist with the resolution proof system [Hak85, CR79, CCT87, Nor15]

Now that we have presented the cutting planes proof system, let us consider its use in the conflict analysis procedure. As in SAT solvers, this analysis is triggered in a pseudo-Boolean solver when a conflict is detected. In this case, a conflict occurs when a pseudo-Boolean constraint becomes falsified by the current assignment.

Similarly to SAT solvers, the analysis is performed by following a bottom-up path in the implication graph (implicitly) maintained by the pseudo-Boolean solver, while taking into account the following remark.

Remark 32

Contrary to the implication graph of a SAT solver, a given pseudo-Boolean constraint may trigger multiple propagations, at different decision levels. This constraint may thus appear at different places in the implication graph, and it may thus be used multiple times during conflict analysis.

During the conflict analysis procedure, the cancellation rule is applied between the conflicting constraint and the reasons that are encountered along the path in the implication graph. However, particular attention has to be paid to the preservation of the conflict when doing so: indeed, contrary to SAT solvers, applying the cancellation rule between a conflicting constraint and a reason does not guarantee deriving a new conflicting constraint. Instead of performing the cancellation rule and checking whether the derived constraint is conflicting, which would be too costly in practice, one can take advantage of the following proposition.

Proposition 32

The slack is *subadditive* with respect to the cancellation rule. That is, given two constraints χ and χ', $\mathsf{slack}(\rho\chi \boxplus \rho'\chi') \leq \rho\mathsf{slack}(\chi) + \rho'\mathsf{slack}(\chi')$.

Proof. To prove the results, we need two main claims enabling to simplify the proof, without losing generality.

Claim 17. *We can consider that $\rho = \rho' = 1$, and thus consider that the two constraints χ and χ' have the same coefficients for the pivot of the cancellation.*

Proof. It is easy to see that, if the two constraints χ and χ' do not have the same coefficient for the pivot, one may simply apply the multiplication rule to produce constraints that are equivalent to χ and χ' and having the same coefficient of the pivot. Moreover, it is easy to see that for any $\rho \in \mathbb{N}$, we have $\mathsf{slack}(\rho\chi) = \rho\mathsf{slack}(\chi)$, so that reasoning on the constraints obtained after multiplication is enough to prove the result.

□

Claim 18. *Computing the slack of a constraint χ under the current partial assignment is equivalent to computing the (absolute) slack of the constraint obtained by simplifying χ with respect to the current partial assignment.*

Proof. Let χ be the pseudo-Boolean constraint $\sum_{i=1}^{n} \alpha_i \ell_i \geq \delta$. Under the current partial assignment, we can write χ as

$$\sum_{i=1,\ell_i=0}^{n} \alpha_i \ell_i + \sum_{i=1,\ell_i=1}^{n} \alpha_i \ell_i + \sum_{i=1,\ell_i \notin \{0,1\}}^{n} \alpha_i \ell_i \geq \delta$$

After simplification with respect to the current assignment, we get the constraint χ_s given by

$$\sum_{i=1,\ell_i \notin \{0,1\}}^{n} \alpha_i \ell_i \geq \delta - \sum_{i=1,\ell_i=1}^{n} \alpha_i$$

Now, simply observe that, under the current partial assignment, $\mathsf{slack}(\chi) = \sum_{i=1,\ell_i=1}^{n} \alpha_i + \sum_{i=1,\ell_i \notin \{0,1\}}^{n} \alpha_i - \delta$ and $\mathsf{slack}(\chi_s) = \sum_{i=1,\ell_i \notin \{0,1\}}^{n} \alpha_i - \left(\delta - \sum_{i=1,\ell_i=1}^{n} \alpha_i\right)$, so that the two slacks are indeed equal.

□

Following the two claims above, let us consider the two pseudo-Boolean constraints χ and χ' given by $\alpha\ell + \sum_{i=1}^{n} \alpha_i \ell_i \geq \delta$ and $\alpha\bar{\ell} + \sum_{i=1}^{m} \alpha'_i \ell'_i \geq \delta'$, respectively, such that these two constraints contain only unassigned literals. Now, let us identify literals ℓ_i and ℓ_j such that $\ell_i = \bar{\ell}_j$ for some indices i and j. For the sake of simplicity, we may assume that these literals are identified with the same index i, for instance by rearranging the terms of the sum. For such literals, let us denote by $\hat{\ell}_i$ the literal ℓ_i if $\alpha_i \geq \alpha'_i$ and the literal ℓ'_i otherwise. Then, we have that $\chi \boxplus \chi'$ is given by

$$\sum_{\substack{i=1 \\ \bar{\ell}_i \notin \mathsf{lit}(\chi')}}^{n} \alpha_i \ell_i + \sum_{\substack{i=1 \\ \bar{\ell}'_i \notin \mathsf{lit}(\chi)}}^{m} \alpha'_i \ell'_i + \sum_{\substack{i=1 \\ \bar{\ell}_i \in \mathsf{lit}(\chi') \\ \bar{\ell}'_i \in \mathsf{lit}(\chi)}}^{\min(n,m)} (\max(\alpha_i, \alpha'_i) - \min(\alpha_i, \alpha'_i))\hat{\ell}_i \geq \delta + \delta' - \sum_{\substack{i=1 \\ \bar{\ell}_i \in \mathsf{lit}(\chi') \\ \bar{\ell}'_i \in \mathsf{lit}(\chi)}}^{\min(n,m)} \min(\alpha_i, \alpha'_i)$$

The slack of this constraint is given by

$$\sum_{\substack{i=1\\ \ell_i \notin \mathsf{lit}(\chi')}}^{n} \alpha_i + \sum_{\substack{i=1\\ \bar{\ell}'_i \notin \mathsf{lit}(\chi)}}^{m} \alpha'_i + \sum_{\substack{i=1\\ \bar{\ell}_i \in \mathsf{lit}(\chi')\\ \bar{\ell}'_i \in \mathsf{lit}(\chi)}}^{\min(n,m)} (\max(\alpha_i, \alpha'_i) - \min(\alpha_i, \alpha'_i)) - \delta - \delta' + \sum_{\substack{i=1\\ \bar{\ell}_i \in \mathsf{lit}(\chi')\\ \bar{\ell}'_i \in \mathsf{lit}(\chi)}}^{\min(n,m)} \min(\alpha_i, \alpha'_i)$$

Observing that the two occurrences of $\min(\alpha_i, \alpha'_i)$ cancel each other, we simplify the expression above and get that $\mathsf{slack}(\chi \boxplus \chi')$ is equal to

$$\sum_{\substack{i=1\\ \bar{\ell}_i \notin \mathsf{lit}(\chi')}}^{n} \alpha_i + \sum_{\substack{i=1\\ \bar{\ell}'_i \notin \mathsf{lit}(\chi)}}^{m} \alpha'_i + \sum_{\substack{i=1\\ \bar{\ell}_i \in \mathsf{lit}(\chi')\\ \bar{\ell}'_i \in \mathsf{lit}(\chi)}}^{\min(n,m)} \max(\alpha_i, \alpha'_i) - \delta - \delta'$$

Now, observe that, as $\min(\alpha_i, \alpha'_i) > 0$ by normalization of χ, we have

$$\mathsf{slack}(\chi \boxplus \chi') \leq \mathsf{slack}(\chi \boxplus \chi') + \sum_{\substack{i=1\\ \bar{\ell}_i \in \mathsf{lit}(\chi')\\ \bar{\ell}'_i \in \mathsf{lit}(\chi)}}^{\min(n,m)} \min(\alpha_i, \alpha'_i)$$

As the right-hand side of the inequation above contains both the sum of $\max(\alpha_i, \alpha'_i)$ and that of $\min(\alpha_i, \alpha'_i)$ for each i such that ℓ_i appears with an opposite polarity in χ and χ', we simplify this inequation as follows

$$\mathsf{slack}(\chi \boxplus \chi') \leq \left(\sum_{\substack{i=1\\ \bar{\ell}_i \notin \mathsf{lit}(\chi')}}^{n} \alpha_i + \sum_{\substack{i=1\\ \bar{\ell}_i \in \mathsf{lit}(\chi')\\ \bar{\ell}'_i \in \mathsf{lit}(\chi)}}^{\min(n,m)} \alpha_i - \delta \right) + \left(\sum_{\substack{i=1\\ \bar{\ell}'_i \notin \mathsf{lit}(\chi)}}^{m} \alpha'_i + \sum_{\substack{i=1\\ \bar{\ell}_i \in \mathsf{lit}(\chi')\\ \bar{\ell}'_i \in \mathsf{lit}(\chi)}}^{\min(n,m)} \alpha'_i - \delta' \right)$$

Now, observe that the sums in parentheses are actually the slacks of χ and χ', and thus

$$\mathsf{slack}(\chi \boxplus \chi') \leq \mathsf{slack}(\chi) + \mathsf{slack}(\chi')$$

To conclude, one just needs to observe that applying the saturation rule on the resulting constraint can only decrease the slack of this constraint, as it only reduces the coefficients of the literals and does not affect the degree.

□

Proposition 32 allows to estimate the slack of the constraint that will be obtained after applying the cancellation rule by computing a upper bound of its slack: whenever this upper bound is not negative, the constraint *may* be non-conflictual, as in the example below.

Example 57

Let us consider the three following pseudo-Boolean constraints:

- $\chi_1 : a + d + \bar{e} \geq 2$
- $\chi_2 : 6\bar{b} + 6c + 4e + f + g + h \geq 7$
- $\chi_3 : 5a + 4b + c + d \geq 6$

Suppose that the decisions $a(1@1)$ and $c(0@2)$ are taken. If the decision $d(0@3)$ is taken now, this triggers some propagations that are illustrated in the implication graph below.

$$\begin{tikzcd}
a(1@1) & & \mathbf{b(0@3)} \\
c(0@2) \arrow[urr, "\chi_2"] \arrow[r, "\chi_3"] & \mathbf{b(1@3)} & \\
d(0@3) \arrow[ur, "\chi_3"] \arrow[rr, "\chi_1"'] & & e(0@3) \arrow[uu, "\chi_2"']
\end{tikzcd}$$

Observe that a conflict has been produced: the graph contains both the vertices $b(0@3)$ and $b(1@3)$ (bold-faced in the graph).
In this case, the constraint χ_2, i.e., $6\bar{b}(1@3) + 6c(0@2) + 4e(0@3) + f(?@?) + g(?@?) + h(?@?) \geq 7$, is the reason for $\bar{b}$, while the constraint χ_3, i.e., $5a(1@1) + 4b(0@3) + c(0@2) + d(0@3) \geq 6$ is conflicting. The cancellation rule must thus be applied between these two constraints to eliminate b, similarly as in SAT solvers. However, $\mathsf{slack}(\chi_2) = 2$ and $\mathsf{slack}(\chi_3) = -1$, and the subadditivity of the slack gives that $\mathsf{slack}(2\chi_2 \boxplus 3\chi_3) \leq 2 \times 2 - 3 \times 1 = 1$. As a consequence, the constraint that will be derived by applying the cancellation rule is not guaranteed to be conflicting.

It is important to note that the subadditivity property of the slack illustrated in the example above is really a *heuristic* for determining whether the conflict will be preserved. In particular, if the upper bound that is obtained when estimating the slack is positive, this does not necessarily mean that the constraint will not be conflicting (this is why it is a *upper bound*). This is illustrated by the following example.

Example 58

Consider the reason for a given by $6a(1@2)+6b(0@2)+2c(0@1)+2d(?@?)+2e(?@?) \geq 6$, which has slack 4 and the conflicting constraint $6\bar{a}(0@2)+6f(0@2)+2g(0@1)+2\bar{d}(?@?)+2\bar{e}(?@?) \geq 6$, which has slack -2.
If we now compute the upper bound of the slack of the constraint obtained by applying the cancellation rule on these two constraints, we obtain $4-2=2$. This upper bound being positive, there is no guarantee that the constraint will be conflicting.
However, when applying the cancellation rule, we obtain the constraint $6b(0@2)+6c(0@2)+2d(0@1)+2g(0@1) \geq 2$, which is equivalent to the clause $b(0@2)+c(0@2)+d(0@1)+g(0@1) \geq 1$, which is still conflicting.

Whenever the estimated slack is not negative, non-falsified literals are successively weakened away from the constraint until the estimated slack becomes negative after the application of the saturation rule, following the procedure of Algorithm 7.

Algorithm 7: reduce

```
Input  : A constraint $\chi_r$ that is the reason for propagating a literal $\ell$ and a conflicting constraint
         $\chi_c$ containing $\bar{\ell}$
Output: The constraint $\chi_r$ reduced to preserve the conflict after the cancellation.
1 $\rho \leftarrow \mathsf{lcm}(\mathsf{coefficient}(\ell, \chi_r), \mathsf{coefficient}(\bar{\ell}, \chi_c))/\mathsf{coefficient}(\ell, \chi_r)$
2 $\rho' \leftarrow \mathsf{lcm}(\mathsf{coefficient}(\ell, \chi_r), \mathsf{coefficient}(\bar{\ell}, \chi_c))/\mathsf{coefficient}(\bar{\ell}, \chi_c)$
3 while $\rho\mathsf{slack}(\chi_r) + \rho'\mathsf{slack}(\chi_c) \geq 0$ do
4     $\ell' \leftarrow$ a non-falsified literal of $\chi_r$ such that $\ell \neq \ell'$
5     $\chi_r \leftarrow \mathsf{saturation}(\mathsf{weaken}(\ell, \chi_r))$
6     $\rho \leftarrow \mathsf{lcm}(\mathsf{coefficient}(\ell, \chi_r), \mathsf{coefficient}(\bar{\ell}, \chi_c))/\mathsf{coefficient}(\ell, \chi_r)$
7 end
8 return $\chi_r$
```

Remark 33

In Algorithm 7, it is possible to select any non-falsified literal from the constraint χ. In particular, it is possible to select first literals that are unassigned, satisfied, or any of both. In *Sat4j*, for instance, unassigned literals are selected first.

By applying the algorithm above, the reason for the propagation of a literal ℓ will be reduced into another pseudo-Boolean constraint, that is also a reason for the literal ℓ, but that ensures that the conflict will be preserved.

Proposition 33

Algorithm 7 eventually derives a reason that preserves the conflict being analyzed.

Proof. Consider the pseudo-Boolean constraint χ that is the reason for the propagation of ℓ defined by $\alpha\ell + \sum_{i=1}^{n} \alpha_i \ell_i \geq \delta$. Note that, in the worst case, Algorithm 7 weakens away all non-falsified literals ℓ_i to get the constraint χ' which still propagates ℓ: indeed, whatever the assignment of the literals that are weakened away, the literal ℓ must be satisfied by construction. Note that this literal is the only one to be non-falsified in χ', so that it must necessarily be equal to the degree of the constraint to satisfy it, and thus $\mathsf{slack}(\chi') = 0$.

As the conflicting constraint has (by definition) a slack that is negative, applying the cancellation rule thus necessarily yields a conflicting constraint, thanks to the subadditivity of the slack (Proposition 32). □

The following example illustrates how the reduction algorithm may be applied to ensure the preservation of the conflict during conflict analysis.

Example 59 (Example 57 cont'd)

When applying the cancellation rule between the reason χ_2 for $\bar{b}$, given by $6\bar{b}(1@3) + 6c(0@2) + 4e(0@3) + f(?@?) + g(?@?) + h(?@?) \geq 7$, and the conflicting constraint χ_3 given by $5a(1@1) + 4b(0@3) + c(0@2) + d(0@3) \geq 6$, the resulting constraint is not guaranteed to be conflicting.
The constraint χ_2 is thus weakened away on a non-falsified literal, say h, for instance, yielding the constraint χ_2' given by $6\bar{b}(1@3) + 6c(0@2) + 4e(0@3) + f(?@?) + g(?@?) \geq 6$, which has slack 2. Observe now that $\mathsf{slack}(2\chi_2' \boxplus 3\chi_3) \leq 2 \times 2 - 3 \times 1 = 1$, so that the constraint that will be obtained after the cancellation step is still not guaranteed to be conflicting.
It is thus necessary to weaken away another non-falsified literal from χ_2', say g, to get the constraint $6\bar{b}(1@3) + 6c(0@2) + 4e(0@3) + f(?@?) \geq 5$ which, after saturation, is equivalent to the constraint χ_2'' given by $5\bar{b}(1@3) + 6c(0@2) + 4e(0@3) + f(?@?) \geq 5$. In this case, $\mathsf{slack}(\chi_2'') = 1$, so that $\mathsf{slack}(4\chi_2'' \boxplus 5\chi_3) \leq 4 \times 1 - 5 \times 1 = -1$, and the conflict will be preserved as the slack of the produced constraint will be necessarily negative. Also, note that the coefficients have changed since the coefficient of the pivot has been reduced by the application of the saturation rule.

In addition to the subadditivity property of the slack, one may ensure that the conflict will be preserved using the following proposition.

Proposition 34 (From [Dix04, Proposition 4.3.6])

Let χ_1 and χ_2 be the two constraints $\alpha\ell + \sum_{i=1}^{n} \alpha_i \ell_i \geq \delta$ and $\bar{\ell} + \sum_{i=1}^{m} \beta_i \ell_i' \geq \delta'$, respectively. If χ_1 (resp. χ_2) propagates ℓ and χ_2 (resp. χ_1) is conflicting under the current partial assignment, then the result of applying the cancellation rule to these two constraints on ℓ is conflicting under the current partial assignment.

Proof. By the subadditivity of the slack, we have that $\mathsf{slack}(\chi_1 \boxplus \alpha\chi_2) \leq \mathsf{slack}(\chi_1) + \alpha\mathsf{slack}(\chi_2)$.

Suppose that χ_1 propagates ℓ and χ_2 is conflicting. We have that $\mathsf{slack}(\chi_1) < \alpha$ and $\mathsf{slack}(\chi_2) < 0$, i.e., $\mathsf{slack}(\chi_1) \leq \alpha - 1$ and $\mathsf{slack}(\chi_2) \leq -1$, so that $\mathsf{slack}(\chi_1 \boxplus \alpha\chi_2) \leq \alpha - 1 + \alpha \times -1 = -1$. The slack of $\chi_1 \boxplus \alpha\chi_2$ is negative, and the conflict is preserved.

Symmetrically, if χ_2 propagates ℓ and χ_1 is conflicting, we have that $\mathsf{slack}(\chi_1) \leq -1$ and $\mathsf{slack}(\chi_2) \leq 0$, so that $\mathsf{slack}(\chi_1 \boxplus \alpha\chi_2) \leq -1 + \alpha \times 0 = -1$. Once again, the slack of $\chi_1 \boxplus \alpha\chi_2$ is thus necessarily negative, hence the conflict is preserved.

$\square$

With Algorithm 7 above, we can now guarantee that a conflict is preserved when applying the cancellation rule. We can thus define a conflict analysis procedure similar to that of SAT solvers, given by Algorithm 8. Note that this algorithm, contrary to the one used in SAT solvers, may return $CONFLICT$: in this case, the inconsistency of the input formula is proven during conflict analysis.

Algorithm 8: learnPB

```
Input  : A conflicting constraint $\chi_0$ and an implication graph $\Gamma$
Output: An assertive constraint, or $CONFLICT$ if the unsatisfiability is proven during
        conflict analysis

1 $\chi \leftarrow \chi_0$
2 while there exists an assigned literal $\ell$ in $\chi$ do
3     $\chi' \leftarrow \mathsf{reduce}(\mathsf{reason}(\bar{\ell}))$
4     $\chi \leftarrow \chi \boxplus \chi'$
5     if $\chi$ is assertive then
6         return NO_CONFLICT
7     end
8 end
9 return CONFLICT
```

Remark 34

Similarly to the conflict analysis performed in SAT solvers, cancellations are typically applied on the literals in the reversed order of their assignment.

Since pseudo-Boolean constraints may propagate different literals at different decision levels (see Example 49), checking whether such a constraint is assertive and at which level it is is harder than when considering clauses or cardinality constraints. Indeed, in the case of a pseudo-Boolean constraint, one may compute, for each literal ℓ of the constraint, whether the constraint propagates a literal at the decision level at which ℓ had been assigned (based on the slack at this decision level, for instance). Moreover, the pseudo-Boolean constraint produced by the conflict analysis procedure may be assertive at different decision levels. In this case, the backjump must be performed at the first decision level at which the constraint is assertive, as the solver always performs the propagations as soon as they appear. This is achieved following Algorithm 9 below.

This algorithm iterates over all the literals of the constraint to identify a decision level at which the constraint is assertive. In this case, we consider only literals that are falsified in the constraint, as a propagation always occurs when a literal becomes falsified. The decision level that is returned is the highest one (i.e., the one having the lowest value, as we are talking about the depth of the search).

Algorithm 9: computeAssertionLevel

```
Input  : A conflicting constraint $\chi$ and the current decision level $d$
Output: The first decision level at which the constraint is assertive
1 $b \leftarrow d$
2 foreach assigned literal $\ell$ in $\chi$ do
3     $d' \leftarrow \mathsf{dl}(\ell)$
4     if $\ell$ is falsified and $d' < b$ and $\chi$ is assertive at decision level $d'$ then
5         $b \leftarrow d'$
6     end
7 end
```

Remark 35

Initially, in *Sat4j*, the decision level was computed by identifying the last literal to be *assigned* (instead of *falsified*) before the literal to propagate. While this approach is not incorrect, the resulting backjump level is not optimal, in the sense that considering falsified literals allows higher backjumps.
The following *scatter plot* compares the performance of *Sat4j* with the two approaches described above, on non-easy instances from the pseudo-Boolean evaluations (see Appendix B). Each point in the plot is an instance, and its coordinates give the runtime (in seconds and in logarithmic scale) of a solver executed on this particular instance. As such, points that are below the red line represents instances that are solved faster by the configuration on the y-axis, while points that are above the red line represent the instances that are solved faster by the solver on the x-axis. In this case, we can see that, in general, the approach considering falsified literals is more efficient.

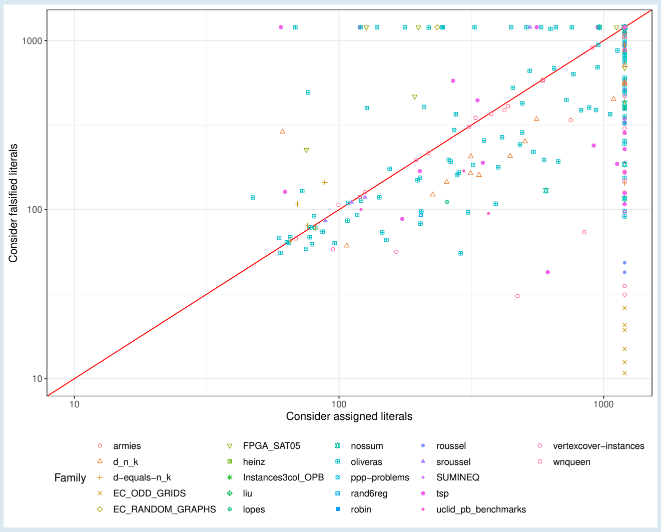


Figure 4.3: Scatter plot comparing the runtime (in seconds, logarithmic scale) of different ways of determining the assertion level of the learned constraint in *Sat4j-GeneralizedResolution*.

Thanks to the different algorithms we have presented here, we can now illustrate the conflict analysis procedure by the following example.

Example 60 (Example 59 cont'd)

After the reduction of the constraint χ_2 yielding $5\bar{b}(1@3)+6c(0@2)+4e(0@3)+f(?@?) \geq 5$, the cancellation rule is applied between this reason and the conflict $5a(1@1) + 4b(0@3) + c(0@2) + d(0@3) \geq 6$ to eliminate b. This operation produces the conflicting constraint $25a(1@1) + 25c(0@2) + 16e(0@3) + 5d(0@3) + 4f(?@?) \geq 30$. This constraint is not assertive yet, so a new cancellation step must be done.
The cancellation rule is then applied between this latter constraint and the reason for $\bar{e}$, namely $a(1@1) + d(0@3) + \bar{e}(1@3) \geq 2$. Observe that the coefficient of $\bar{e}$ is 1 in this constraint, so that the conflict is guaranteed to be preserved and the constraint does not need to be reduced. The constraint that is obtained here is $41a(1@1) + 25c(0@2) + 21d(0@3) + 4f(?@?) \geq 46$, which is both conflicting and assertive: it propagates d at decision level 2. This constraint is thus learned, and a backjump is performed to decision level 2.

Let us now discuss an important difference between the conflict analysis procedure implemented in pseudo-Boolean solvers compared to SAT solvers: the need to apply *arithmetic* operations. Even though this is the main strength of pseudo-Boolean solvers (they are able to "count"), it is also a weakness. Indeed, when applying the cancellation rule, the coefficients appearing in the constraints that are derived may grow significantly. For instance, observe the growth of the coefficients in few cancellation steps in Example 60: in practice, there may be many such steps during conflict analysis, and the learned constraints will be reused later on, so that coefficients will continue to grow, requiring to use arbitrary precision encodings for them. Dealing with such an encoding slows down the solver, as it cannot use the arithmetic operations provided on fixed precision encodings (typically, on primitive types such as `int` or `long int`).

Different solutions have been proposed to limit the size of the coefficients, such as the reduction of the derived constraints to cardinality constraints [CK05], following Algorithm 10.

The algorithm has two main steps. The first one identifies the minimum number of literals that must be satisfied to satisfy the constraint. This is achieved by greedily choosing the literals with the highest coefficients, until the sum of their coefficients becomes greater than the degree. The second part of the algorithm removes the literals with a low coefficient so as to strengthen the derived cardinality constraint. Intuitively, such literals can be removed when their weakening from the original constraint does not change the minimum number of literals that must be satisfied to satisfy the constraint, as in the example below.

Algorithm 10: reduceToCardinality

```
Input  : A pseudo-Boolean constraint $\chi$ given by $\sum_{i=1}^{n} \alpha_i \ell_i \geq \delta$
Output: A cardinality constraint $\kappa$ entailed by $\chi$
1  $s \leftarrow 0$
2  $d \leftarrow 0$
3  $\alpha_{last} \leftarrow 0$
4  foreach literal $\ell_i$ in $\chi$ by descending coefficient do
5      $s \leftarrow s + \alpha_i$
6      $\alpha_{last} \leftarrow \alpha_i$
7      $d \leftarrow d + 1$
8      if $s \geq \delta$ then
9          break
10     end
11 end
12 $L \leftarrow \mathsf{lit}(\chi)$
13 $\mu \leftarrow \delta - (s - \alpha_{last})$
14 while $\min(\{\alpha_i | \ell_i \in L\}) < \mu$ do
15     $\alpha_{min} \leftarrow \min(\{\alpha_i | \ell_i \in L\})$
16     $L \leftarrow L \setminus \{\ell_{min}\}$
17     $\mu \leftarrow \mu - \alpha_{min}$
18 end
19 return $\sum_{\ell \in L} \ell \geq d$
```

Example 61

Consider the constraint χ given by $8a + 2b + c + d \geq 10$. First, observe that this constraint requires at least 2 literals to be satisfied. Indeed, if we consider the coefficients in decreasing order, the first literal (a) has coefficient 8, which is not enough to satisfy the constraint. We thus consider the next literal (b), which has coefficient 2. Together with a, these two literals are enough to satisfy the constraint, so the degree of the cardinality constraint is 2. We could stop it and get the cardinality constraint $a + b + c + d \geq 2$. Now observe that we can weaken away the literal d from χ, to get $8a+2b+c \geq 9$. This constraint is implied by χ, and allows to derive the cardinality constraint $a + b + c \geq 2$, which is stronger than the one derived before, as we removed d from this constraint. To detect that d can be removed, observe that here the value of μ, as defined in the algorithm, is equal to 2, and that $\alpha_{min} = 1 < 2$.
Similarly, if we consider the constraint $6a + 6b + 2c + 2d + 2e \geq 6$, we first derive the cardinality constraint $a + b + c + d + e \geq 1$. Now, we observe that $\mu = 6$, so that d and e can successively be removed to get the cardinality constraint $a + b + c \geq 1$.
If we now consider the constraint $6\bar{b} + 6c + 4e + f + g + h \geq 7$, we derive the cardinality constraint $\bar{b} + c + e + f + g + h \geq 2$. No literal can be removed, as $\mu = 1$.

Remark 36

When Algorithm 10 is applied to a learned constraint, the resulting cardinality constraint may not be conflicting anymore, and thus may not propagate any literal. To prevent this from happening, the implementation in *Galena* [CK05] weakens away all non-falsified literals before applying the reduction above when the conflict is not guaranteed to be preserved.

Another advantage of the reduction to cardinality constraints, as observed in [CK05], is that propagations can easily be detected on such constraints, as watched literals generalize directly for such constraints. Limiting constraint learning to cardinality constraint learning allows thus to improve the efficiency of unit propagation and of the conflict analysis, as fixed precision arithmetic may be used. The simplicity of handling cardinality constraints compared to general pseudo-Boolean constraints explains why solvers dedicated to problems encoded with cardinality constraints, such as *MiniCARD* [LM12], have been developed. However, reasoning only with cardinality constraints may drastically reduce the strength of the constraints learned by the solver. For instance, Algorithm 10 requires to weaken the derived constraints to make sure they remain cardinality constraints. To maintain the inference of strong constraints, solvers taking advantage of the *division* rule have been developed.

To limit the growth of the coefficients during conflict analysis, *RoundingSat* [EN18] introduced an aggressive use of the weakening and *division* rules, applying Algorithm 11 to reduce the constraint before applying the cancellation rule, on both the conflict and the reason side.

Algorithm 11: roundingSatReduce

```
Input  : A pseudo-Boolean constraint $\chi$ and a literal $\ell$ of $\chi$
Output: The constraint $\chi$ after reduction

$\alpha \leftarrow \mathsf{coefficient}(\ell, \chi)$
foreach literal $\ell'$ in $\chi$ do
    $\alpha' \leftarrow \mathsf{coefficient}(\ell', \chi)$
    if $\ell'$ is not currently falsified then
        if $\alpha'$ is not divisible by $\alpha$ then
            // The literal $\ell'$ is weakened away
            $\mathsf{degree}(\chi) \leftarrow \mathsf{degree}(\chi) - \alpha'$
            $\alpha' \leftarrow 0$
        end
    end
    // Dividing the coefficient: $\alpha'$ is always divisible by $\alpha$ here
    $\mathsf{coefficient}(\ell', \chi) \leftarrow \alpha'/\alpha$
end
return $\chi$
```

When a conflict occurs, both the conflict and the reason are weakened so as to remove all literals not falsified by the current assignment and having a coefficient not divisible by the weight of the literal used as pivot for the cancellation step, before being divided by this weight. This ensures that the pivot has a weight equal to 1, which guarantees that the result of the cancellation step will be conflictual, thanks to Proposition 34.

Example 62 (Example 60 cont'd)

The weakening operation is applied on the reason for $\bar{b}$, given by $6\bar{b}(1@3) + 6c(0@2) + 4e(0@3) + f(?@?) + g(?@?) + h(?@?) \geq 7$. All non-falsified literals having a coefficient not divisible by 6 are weakened away, giving $6\bar{b}(1@3) + 6c(0@2) + 4e(0@3) \geq 4$. This constraint is then divided by 6, to get $\bar{b}(1@3) + c(0@2) + e(0@3) \geq 1$.
Similarly, the weakening operation is applied on the conflict given by $5a(1@1) + 4b(0@3) + c(0@2) + d(0@3) \geq 6$. All non-falsified literals having a coefficient not divisible by 4 are weakened away, giving $4b(0@3) + c(0@2) + d(0@3) \geq 1$. This constraint is then divided by 4, to get $b(0@3) + c(0@2) + d(0@3) \geq 1$
Applying the cancellation rule on these two constraints gives $2c(0@2)+d(0@3)+e(0@3) \geq 1$ (which is not saturated, as *RoundingSat* does not use the saturation rule). This constraint is not assertive, so a new cancellation step is performed between this constraint and the reason for e, i.e., $a(1@1) + d(0@3) + \bar{e}(1@3) \geq 2$.
In this case, the coefficient of the literals in e is 1 in both constraints, so there is no need to apply any weakening or division operation (in practice, the reduction is a no-op on both constraints). The cancellation rule is thus applied to the two constraints, giving $2c(0@2) + 2d(0@3) + a(1@1) \geq 2$, which propagates the literal d at decision level 2.

RoundingSat's approach allows both to keep coefficients small in practice and to preserve the strength of the cutting planes proof system. However, some constraints inferred by *RoundingSat* may be weaker than those inferred using generalized resolution (compare the constraints derived in Examples 60 and 62).

Let us compare the performances of the two main cutting planes proof systems, namely generalized resolution, as used in *Sat4j* [LP10], and *RoundingSat*'s proof system [EN18]. To this end, we ran these two solvers on the same experimental settings as in the previous section. The timeout was set to 1200 seconds and the memory limit to 32 GB. The results are given in Figures 4.4, 4.5, 4.6, 4.7.

The scatter plot in Figure 4.4 compares the implementation of the classical generalized resolution algorithm implemented in *Sat4j* (named *Sat4j-GeneralizedResolution*) to the implementation of *RoundingSat* in *Sat4j* (named *Sat4j-RoundingSat*). On most instances, it is clear that the *RoundingSat*-based algorithm performs better than the other algorithm, even though some instances are faster solved by the generalized resolution-based approach.

It is however important to note that *Sat4j-RoundingSat* does *not* use exactly the same algorithm as *RoundingSat* (see Appendix A for more details). In particular, *RoundingSat* is much faster than *Sat4j-RoundingSat*, as shown by the scatter plot in Figure 4.5 (partially because the former is written in Java, while the latter is written in C++). More precisely, *RoundingSat* solves 4442 instances, while *Sat4j-RoundingSat* only solves 3843 instances. The difference is less marked when considering *RoundingSat2*, as this solver solves 4178 with `gmp` (Figure 4.6) and 4216 instances without `gmp` (Figure 4.7).

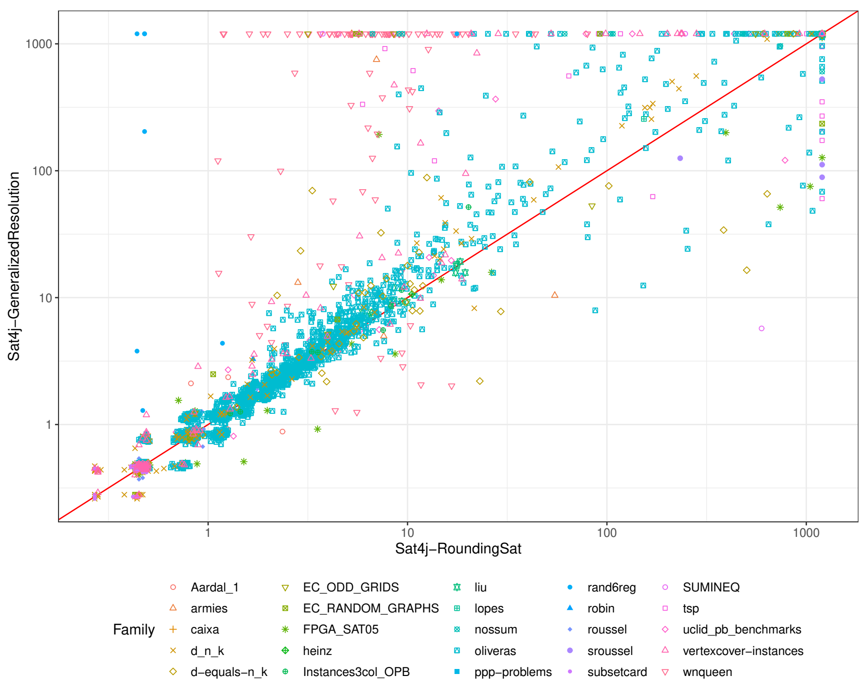


Figure 4.4: Scatter plot comparing the runtime (in seconds) of *Sat4j-GeneralizedResolution* and *Sat4j-RoundingSat*, in logarithmic scale.

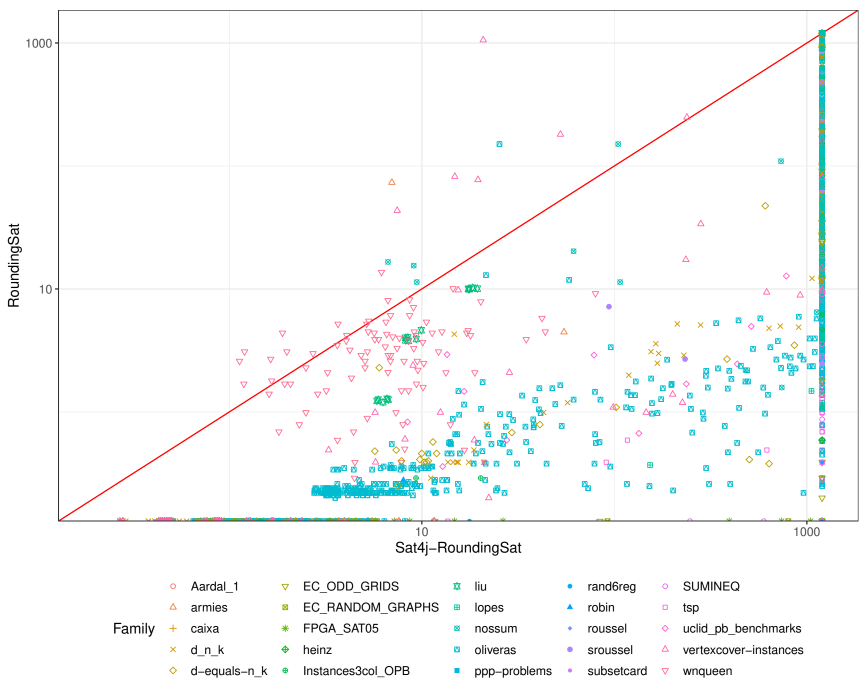


Figure 4.5: Scatter plot comparing the runtime (in seconds) of *RoundingSat*and *Sat4j-RoundingSat* (logarithmic scale).

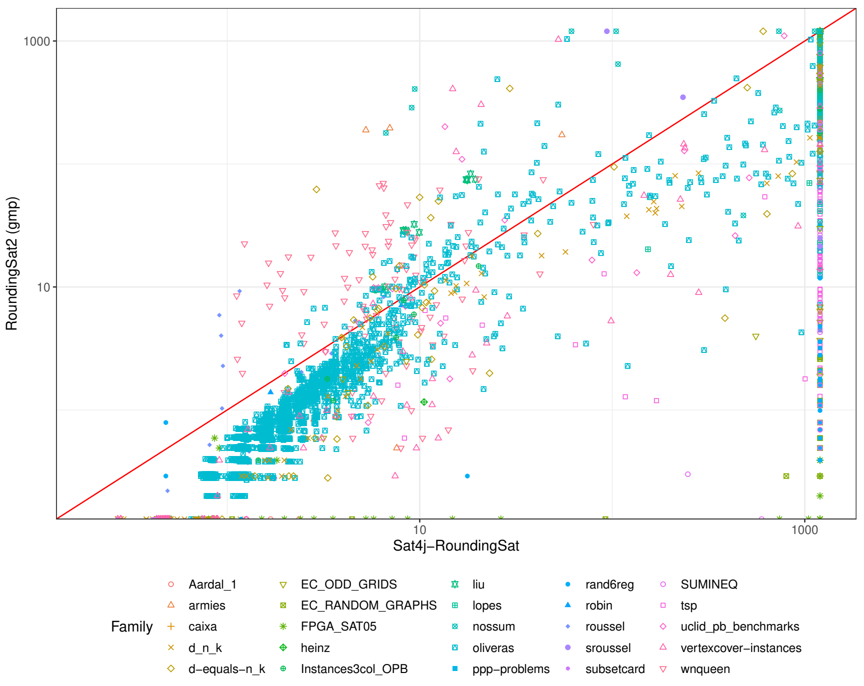


Figure 4.6: Scatter plot comparing the runtime (in seconds) of *RoundingSat2 (gmp)* and *Sat4j-RoundingSat* (logarithmic scale).

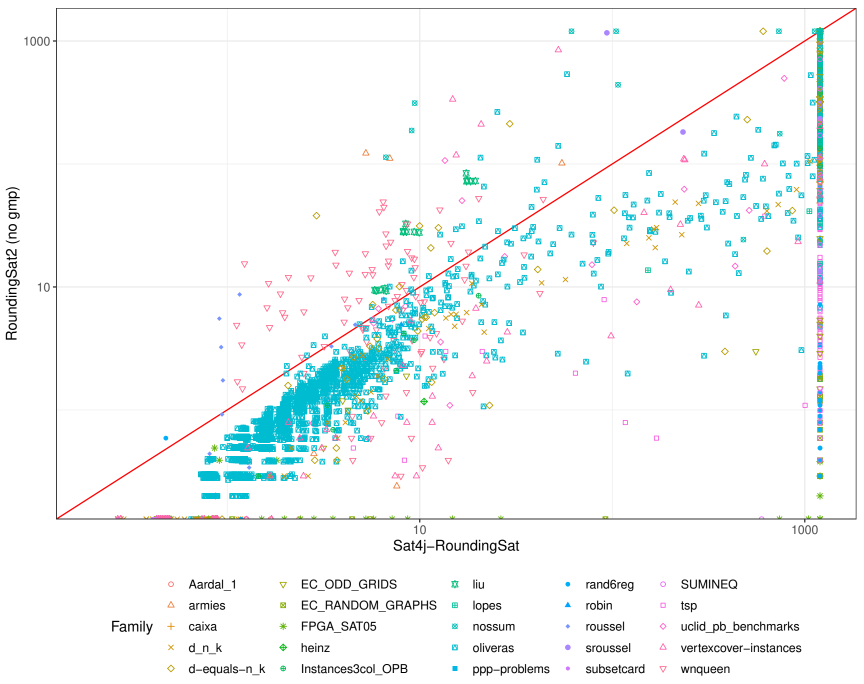


Figure 4.7: Scatter plot comparing the runtime (in seconds) of *RoundingSat2 (no gmp)* and *Sat4j-RoundingSat* (logarithmic scale).

4.2.3 Guiding the Search in a Pseudo-Boolean Solver

As in SAT solvers, the CDCL algorithm implemented in pseudo-Boolean solvers also relies on additional features allowing to improve the performance of the solver. Mostly inherited from SAT solvers, these features may also be adapted to the pseudo-Boolean case, for instance by taking into account the coefficients of the constraints being considered. In practice, however, current pseudo-Boolean solvers often implement strategies similar to those used in classical SAT solvers, as described in this section.

Pseudo-Boolean Branching Heuristic

Current pseudo-Boolean solvers rely on the VSIDS heuristic (or one of its variants) to decide which variable should be assigned next. In practice, this heuristic may be used as is by pseudo-Boolean solvers, even though doing so does not allow to take into account all the information given by a pseudo-Boolean constraint, as observed in [CK05] (which, however, does not explicitly provide a more suitable heuristic). This is why different variants of this heuristic have been proposed.

In [Dix04, Section 4.5], it is proposed to add, for each variable appearing in a cardinality constraint *of the original problem* (i.e., not for *learned* constraints) the degree of this constraint to the *initial* score of the corresponding variables. This approach actually counts the occurrences of the variable in the clauses that are represented by the cardinality constraint.

Example 63 (Example from [Dix04, Section 4.5])

If the cardinality constraint $a + b + c \geq 2$ is present in the original constraint database, the score of each of its variables is increased by 2. Indeed, this constraint is equivalent to the conjunction of the clauses $a + b \geq 1$, $a + c \geq 1$ and $b + c \geq 1$. If this constraint is learned, the corresponding scores are only increased by 1.

Despite providing a more specific heuristic than the original VSIDS heuristic when considering pseudo-Boolean problems, this heuristic is not completely satisfactory, as pointed out by the following observation.

Observation 8

The VSIDS implementation proposed in [Dix04, Section 4.5] does not fit well in modern implementations of VSIDS, and especially of EVSIDS.
First, as only the original constraints are considered, the heuristic does not bring any improvement over the classical implementation of the heuristic, which essentially relies on the bumping of variables involved in recent conflicts.
Second, the particular form of general pseudo-Boolean constraints is not taken into account by this heuristic. The main reason for only considering cardinality constraints in this case is that computing the number of clauses in which a literal of a pseudo-Boolean constraint appears is hard in general.

Another alternative, implemented in *Pueblo* [SS06], consists in estimating the relative importance of a literal in a constraint, by computing the ratio of its coefficient by the degree of the constraint. This value is then added to the VSIDS score of the variable.

Example 64

When bumping the variable a from the constraint $5a + 5b + c + d + e + f \geq 6$, the score is augmented by $5/6$.

On the contrary, *Sat4j* [LP10] and *RoundingSat* [EN18] both implement a more classical EVSIDS heuristic. However, some implementation details are worth noting for these two solvers. In particular, *Sat4j* bumps each literal encountered during conflict analysis *each time it appears in a reason*, while *RoundingSat*, bumps all these literals only once (as in *MiniSat*), except if the literal is used as pivot during conflict analysis, in which case it is bumped *twice*.

Learned Constraint Deletion

For reasons similar to SAT solvers, pseudo-Boolean solvers regularly delete constraints from their database. Recall that doing so allows to limit the memory usage, and to prevent Boolean constraint propagation from slowing down.

In current pseudo-Boolean solvers, this feature is mostly inherited directly from SAT solvers. For instance, both *Pueblo* [SS06] and *Sat4j* [LP10] use *MiniSat*'s learned constraint deletion, based on their activity (the less active constraints are removed first), while *RoundingSat* considers a hybrid approach, based on both the LBD and the activity measures (the latter is used as a tie-break rule when the former gives identical measures).

Observation 9

The Literal Block Distance, as described in Definition 102, is not well-defined for pseudo-Boolean constraints, as some literal may be unassigned in such constraints when they are conflicting or assertive. The workaround chosen by *RoundingSat* is to compute the LBD over *assigned* literals only.

In other solvers, such as *pbChaff* [DG02] and *Galena* [CK05], the learned constraint deletion in use (if any) is not documented. In [CK05], a perspective is however mentioned, which consists in weakening the learned constraints instead of removing them.

A possible explanation for the few implementations of learned constraint deletion strategies dedicated to pseudo-Boolean solving is that this feature is not as essential as in SAT solvers. Indeed, pseudo-Boolean solvers are slower in practice than SAT solvers, especially because the operations they need to perform, such as detecting propagations and applying the cancellation rule, are more complex than their counterpart in SAT solvers. In practice, this means that the number of conflicts per second in a pseudo-Boolean solver is lower than that in a SAT solver, and so is the number of learned constraints. As a consequence, a pseudo-Boolean solver does not need to clean its learned constraint database as regularly as a SAT solver, and can even avoid doing so. Figures 4.8 and 4.9 show that never deleting learned constraint may even provide better performances, in this case compared to the activity-based deletion strategy (which is the default strategy in *Sat4j*).

The scatter plots show that, for instances that are solved in less than about 50 seconds, there is very little difference: on these inputs, the solver does not learn enough constraints to need to delete any of them. For the rest of the instances, we can indeed see that there are more instances that are solved faster without constraint deletion than when this feature is enabled. In practice, when all the constraints are

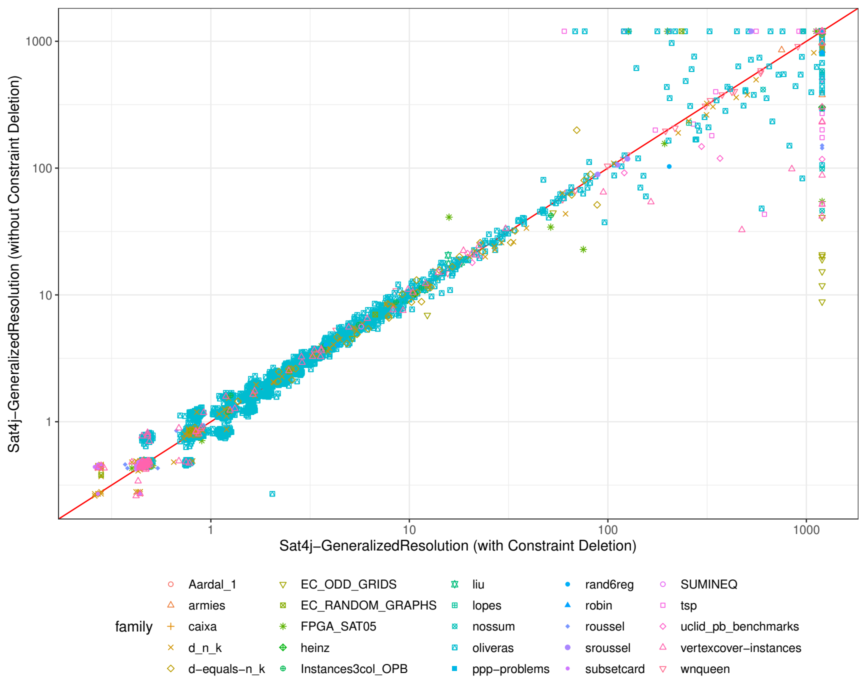


Figure 4.8: Comparison of the runtime (in seconds) of *Sat4j-GeneralizedResolution* with and without learned constraint deletion enabled (logarithmic scale).

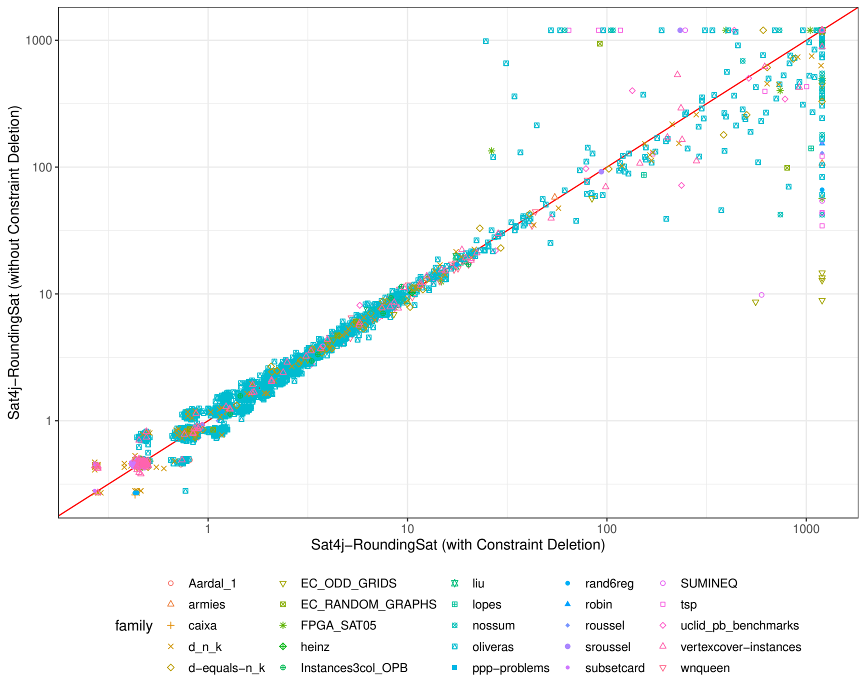


Figure 4.9: Comparison of the runtime (in seconds) of *Sat4j-RoundingSat* with and without learned constraint deletion enabled (logarithmic scale).

kept, this allows to improve the propagation power of the formula being considered by the solver, and there are not enough constraints to actually slow down these propagations.

Restarts

Another feature that is implemented in pseudo-Boolean solvers are restarts. For instance, *Sat4j* implements *PicoSAT*'s static and aggressive restart scheme [Bie08b] and *RoundingSat* [EN18] uses a Luby-based restart policy [LSZ93, Hua07]. Note that a common point to these two strategies is that they do not take into account the constraints that are being considered, as they are both static policies. They may thus be reused since they are independent from the type of the constraints being considered.

In older solvers, it is not clear whether restarts are implemented or not, as there is no mention of this feature in *pbChaff* [DG02] and in *Galena* [CK05]. As *Pueblo* [SS06] is heavily based on *MiniSat* [ES04], it is most likely to inherit its restart policy, even though no mention of this feature is made in [SS06] either.

Yet, as in SAT solvers, restarts are an important feature of pseudo-Boolean solvers. They allow together with learned constraint deletion, the EVSIDS branching heuristic and the conflict analysis procedure based on the cutting planes proof system, to efficiently solve pseudo-Boolean problems. Yet, solvers based on cutting planes are slower in practice than classical SAT solvers, which explains why resolution-based pseudo-Boolean solvers have been developed.

4.3 Pseudo-Boolean Solving Based on Resolution

Current implementations of the cutting planes proof system in CDCL pseudo-Boolean solvers fail to keep the promises of the theory. In particular, these solvers do not implement the full power of the cutting planes proof system [VEG$^+$18], mostly because finding which rules to apply and when is not easy. In this context, a number of pseudo-Boolean solvers relying on the resolution proof system for their conflict analysis have been developed [WS01, ES06, SN15, MML14]. Typically, these solvers use internally a classical SAT solver, which is given as input the original pseudo-Boolean formula, converted into a CNF formula.

The main difference between such solvers relates on *how* they encode the original input. Undeniably, considering the CNF representation of the original pseudo-Boolean formula, i.e., a strictly equivalent representation of this formula, is too costly in practice. Indeed, such representations may be exponentially larger than the original pseudo-Boolean representation (Proposition 3), and thus cannot scale on large instances. Instead, resolution-based solvers use other approaches, as for instance those based on CNF encodings that are *equisatisfiable* to the original pseudo-Boolean formulae (see Definition 26).

The main advantage of using such encodings is that, thanks to the introduction of auxiliary variables, the size of the CNF encoding that is produced from a pseudo-Boolean formula is often much smaller in practice than the CNF representation of the same formula. However, to make sure that a resolution-based pseudo-Boolean solver is efficient in practice, the encoding must also be clever enough to counterbalance the relative weakness of the resolution proof system compared to cutting planes.

An important criterion, for instance, is to preserve the propagations of the original formula (we also say that the encoding is *arc-consistent*). Intuitively, this means that the same literals are propagated under the same partial assignment in both the original formula and its encoding. This is for instance the case of the BDD-based encodings used by *MiniSat+* [ES06] or by *NaPS* [SN15] (the latter actually uses ROBDDs to represent pseudo-Boolean constraints). This is also the case of the *ladder* encoding [AM05, AS14], the *sequential* encoding [HMS12] and the *cardinality network* [ANORC11] implemented in *Open-WBO* [MML14].

Some other encodings are *not* arc-consistent, but provide smaller encodings, such as the *adder* encoding and the *sorting networks* also implemented in *MiniSat+* [ES06]. Many other encodings have been proposed (see, e.g., [PS15]). We do not study all of them, as this thesis mostly focuses on *native* pseudo-Boolean reasoning.

Another approach for using resolution-based reasoning on pseudo-Boolean problems consists in lazily inferring clauses during conflict analysis, as in *SATIRE* [WS01] or *Sat4j-Resolution* [LP10]. In such an approach, the pseudo-Boolean constraints are considered as they are, in particular for detecting propagations and conflicts. However, during conflict analysis, conflicting and assertive constraints are turned into clauses, so as to perform a resolution step between the two obtained clauses (and thus derive a new clause), similarly to what is done in constraint programming with the so-called *lazy clause generation* [Stu10]. Algorithm 12 describes the procedure for deriving a clause from a pseudo-Boolean constraint during conflict analysis.

Algorithm 12: inferClause

```
Input  : A (conflicting or assertive) pseudo-Boolean constraint $\chi$ and a threshold $\theta$
Output: A set of falsified literals
1  $\sigma \leftarrow \mathsf{sumOfCoefficients}(\chi) - \mathsf{degree}(\chi)$
2  $F \leftarrow \emptyset$
3  foreach literal $\ell$ appearing with a coefficient $\alpha$ in $\chi$ do
4      if $\ell$ is falsified under the current assignment then
5          $F \leftarrow F \cup \{\ell\}$
6          $\sigma \leftarrow \sigma - \alpha$
7          if $\sigma < \theta$ then
8              break
9          end
10     end
11 end
12 return $F$
```

Intuitively, the algorithm allows to select as many falsified literals as needed to preserve the conflict or the propagation. To ensure this preservation, the slack is incrementally computed while selecting variables, and the selection stops when the slack is below the threshold. This threshold is set to 0 in case of a conflict and to the coefficient of the propagated literal in case of a propagation, so that the set of selected literals produced by the algorithm can either be interpreted as a conflicting clause or a reason for the propagation of a literal.

Remark 37

In the algorithm above, it is possible to iterate over the constraint in any order. However, in order to get shorter clauses, it is commonly preferred to perform the iteration greedily, i.e., by descending coefficients.

The algorithm above allows to lazily infer clauses during conflict analysis from the considered pseudo-Boolean constraints, and thus to apply classical resolution, as illustrated in the following example.

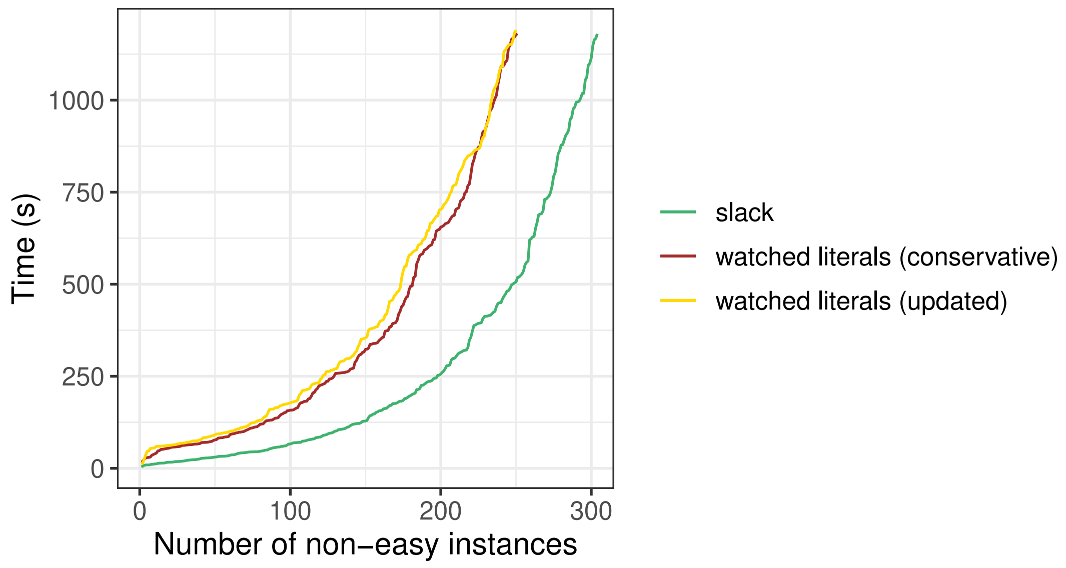


Figure 4.10: Cactus plot comparing the different algorithms for detecting propagations in the pseudo-Boolean solver *Sat4j-Resolution* on non-easy instances.

Example 65

Consider the constraint $5a(0@3)+5b(?@?)+c(?@?)+d(?@?)+e(0@1)+f(1@2) \geq 6$. This constraint propagates b under the current partial assignment, so that the threshold is set to 5 in the algorithm, as this is the coefficient of b in the constraint. The value σ is here initialized to 8. The first falsified literal to be considered is a, which has coefficient 5. This literal is added to the set F, and σ is updated and is now assigned 3. As this value is below the threshold, the algorithm stops. The reason for b is thus the clause $a(0@3) + b(?@?) \geq 1$.
After the propagation of b, the constraint $2a(0@3)+\bar{b}(0@3)+c(?@?)+e(0@1) \geq 2$ becomes conflicting. In this case, the threshold of the algorithm is set to 0 and the value of σ is initialized to 3. In this case, all falsified literals are added to F, to get $\sigma = -1$. The conflicting clause is thus $a(0@3) + \bar{b}(0@3) + e(0@1) \geq 1$.
The clause $a(0@3)+e(0@1) \geq 1$ is then obtained by applying the resolution rule between the two clauses derived above.

The main advantage of this approach is that it allows to apply the efficient CDCL algorithm of classical SAT solvers, while considering a succinct (and implicit) CNF representation of the original pseudo-Boolean formula.

Let us now empirically evaluate the different state-of-the-art approaches presented in this chapter for solving pseudo-Boolean problems. The following experiments have been executed in the same experimental setting as above, and described in Appendix B. The timeout was set to 1200 seconds and the memory limit to 32 GB.

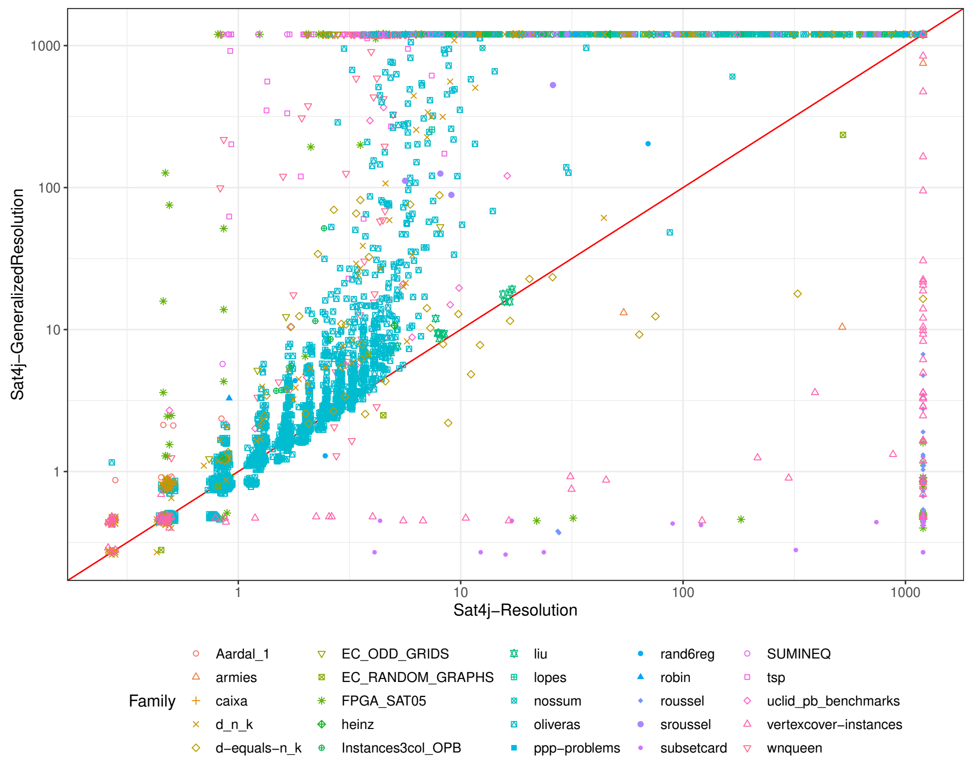


Figure 4.11: Scatter plot comparing the runtime (in seconds) of *Sat4j-Resolution* and *Sat4j-GeneralizedResolution*.

First, an interesting observation regarding *Sat4j-Resolution* is that for this solver, as shown in the cactus plot in Figure 4.10, the slack-based approach for detecting propagations in pseudo-Boolean constraints is more efficient than the one based on watched literals. This is the opposite observation as that we made for *Sat4j-GeneralizedResolution* (see Figure 4.2). This can be explained by the fact that *Sat4j-Resolution* only derives clauses: the cost of maintaining watched literals in pseudo-Boolean constraints is not amortized by their number. It is important to note that this observation only takes into account the impact of how propagations are detected in *general* pseudo-Boolean constraints: propagations in clauses and cardinality constraints are always detected using watched literals, as it is easy to maintain them for such constraints.

Now, let us evaluate the benefits of *Sat4j-Resolution* over *Sat4j-GeneralizedResolution*. A comparison of these two solvers is presented in Figure 4.11. Clearly, the resolution-based approach is more efficient, but there remain instances (e.g., from the `vertexcover` or `subsetcard` families) for which generalized resolution is faster: these instances are hard for the resolution proof system (they have only exponential-sized refutation proofs). This illustrates the complementarity of the two approaches, which is leveraged by solvers such as *Sat4j-Both*. This solver runs both approaches in parallel (one CPU core per approach), and outputs the result obtained by the faster approach. Results are shown in Figures 4.12 and 4.13. A variant of this solver, *Sat4j-Both (sober)*, only runs *Sat4j-GeneralizedResolution* for one minute, and let *Sat4j-Resolution* run the rest of the time.

As shown by these scatter plots, *Sat4j-Both* benefits from both approaches and, despite a slight overhead due to the multi-threaded environment in which the solvers are executed, the solver is able to solve quite efficiently most of the considered instances.

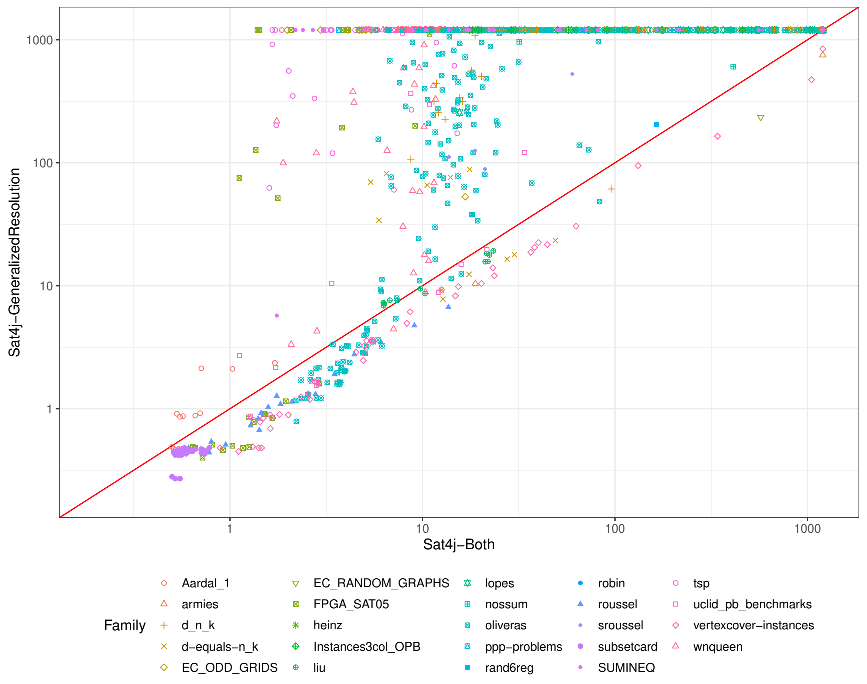


Figure 4.12: Scatter plot comparing the runtime (in seconds) of *Sat4j-Both* and *Sat4j-GeneralizedResolution*.

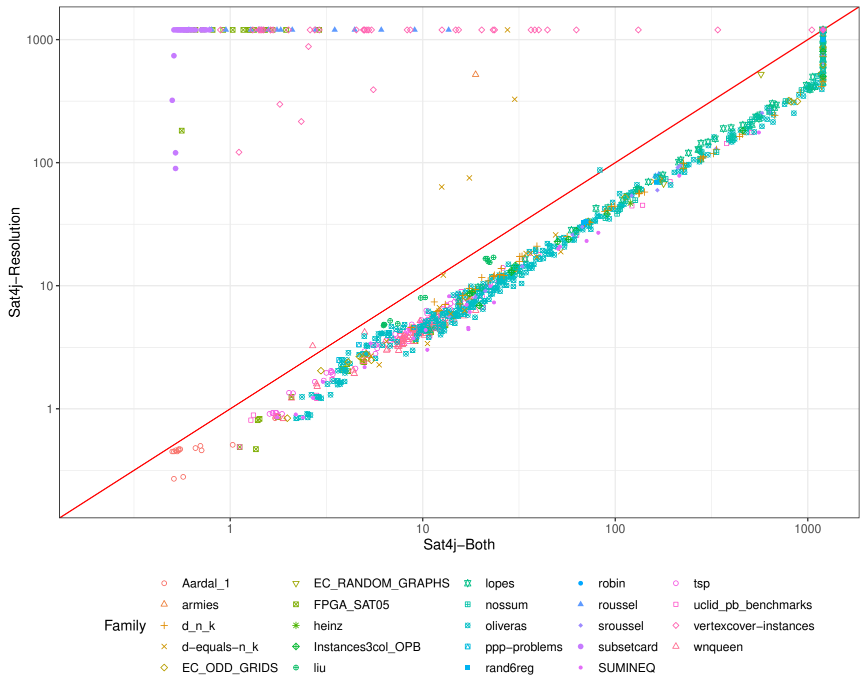


Figure 4.13: Scatter plot comparing the runtime (in seconds) of *Sat4j-Both* and *Sat4j-Resolution*.

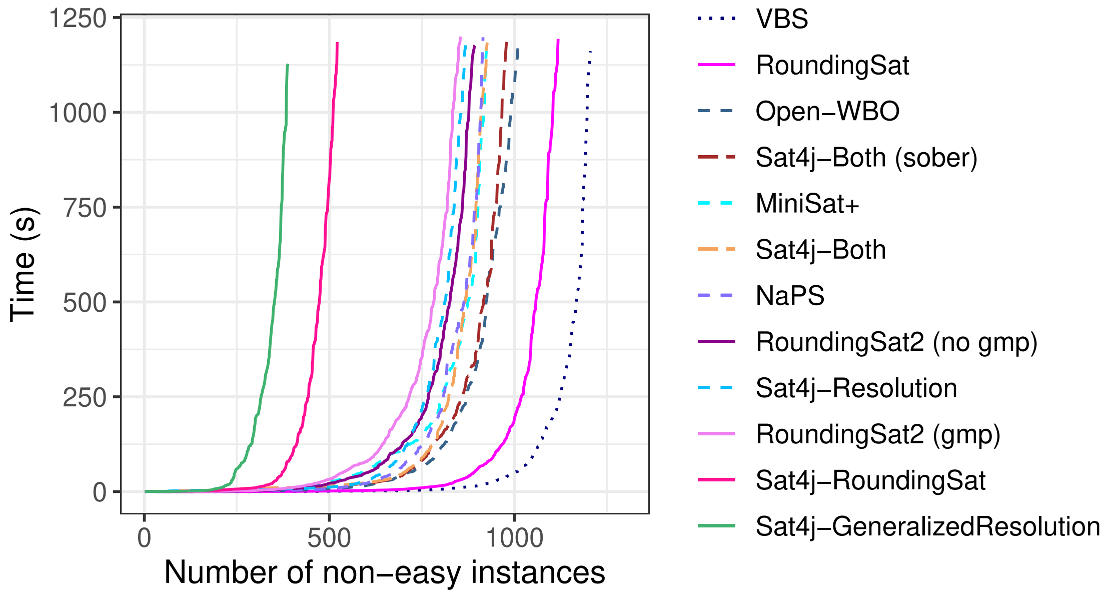


Figure 4.14: Cactus plot comparing the runtime (in seconds) of different state-of-the-art pseudo-Boolean solvers.

Figure 4.14 also compares the performance of these solvers with other state-of-the-art pseudo-Boolean solvers (the experimental settings is the same as above). In this cactus plot we can see that resolution-based solvers have good performance, even though *RoundingSat* (which, at the time of writing, has not yet participated in any pseudo-Boolean competition), is the best of all solvers studied here. We can observe that the performance of *RoundingSat* is quite impressive, especially because it is close to that of the *Virtual Best Solver (VBS)*. This solver is obtained by choosing, for each instance, the runtime of the solver that is the fastest to solve the instance.

However, one can also observe that more recent versions of this solver (denoted as *RoundingSat2*) are not as efficient as the first version of *RoundingSat*. In particular, *RoundingSat2* now uses arbitrary precision arithmetic to represent big coefficients, which adds a supplementary cost during conflict analysis (while *Sat4j* has always used arbitrary precision arithmetic). One can note that, in our experiments, the version of *RoundingSat2* that does not use `gmp` to represent large numbers (and which actually uses `boost`) is slightly more efficient than that using `gmp`.

Also, note that all solvers here are written in C++, except those based on *Sat4j*, which are written in Java: this impacts the efficiency of these latter solvers, as Java-based programs are roughly three times slower than their C++ equivalent. Moreover, in Java, it is not possible to finely tune memory usage, while in C++ programs it is possible to do so in order to optimize the performance of the solver. For instance, compare in Figure 4.15 the performance of *MiniSat 2.2.0*, written in C++, and *jMiniSat*, which is a Java port of *MiniSat 2.2*. This comparison is made from the results of the two solvers during the SAT Competition 2011 [JLBR11].

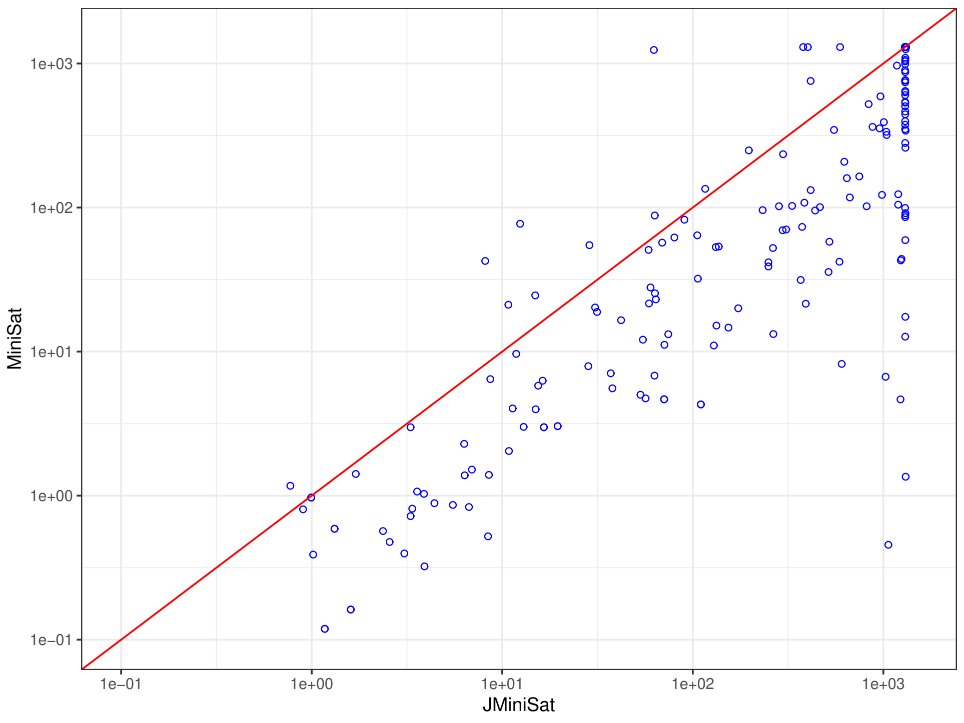


Figure 4.15: Scatter plot comparing the runtime of *MiniSat* and *JMiniSat* on the instances of “Phase 1” of the SAT Competition 2011.

Chapter 5

On Irrelevant Literals in Pseudo-Boolean Constraint Learning

In order to get more benefits from the conflict analysis procedure implemented in pseudo-Boolean solvers, the constraints that are learned must be as strong as possible. Indeed, strong constraints allow to eliminate a larger part of the search space, and thus to find more efficiently a solution or an unsatisfiability proof. In practice, however, current implementations of the cutting planes proof system in pseudo-Boolean solvers do not always allow the derivation of the strongest possible constraints. In this chapter, we study a specific problem arising with pseudo-Boolean constraints but not with clauses or cardinality constraints that are neither tautological nor contradictory: the presence of *irrelevant literals* [LMMW20].

Definition 105 (Irrelevant Literal)

A literal ℓ is said to be *irrelevant* with respect to a constraint χ when $\chi|\ell \equiv \chi|\bar{\ell}$, where $\chi|\ell$ denotes the conditioning of χ by ℓ (see Definition 59). Otherwise, ℓ is said to be *relevant* with respect to χ. In this case, we also say that χ *depends* on ℓ.

Example 66

In the constraint $10a + 5b + 5c + 2d + e + f \geq 15$, the three literals d, e and f are irrelevant.

In the following, when there is no ambiguity about which constraint is considered, we omit the constraint and simply say that ℓ is relevant or irrelevant.

5.1 Characterization of Irrelevant Literals

Let us start our study of irrelevant literals with a characterization of irrelevant literals, in order to identify them in the pseudo-Boolean constraints that are handled by pseudo-Boolean solvers. First, the following theorem provides a useful alternative definition for irrelevant literals.

Theorem 10

Let χ be a pseudo-Boolean constraint and ℓ be a literal of this constraint. ℓ is irrelevant in χ if and only if, for any model M of χ, flipping the value of ℓ in M does not make it a counter-model of χ.

Proof. Let χ be a pseudo-Boolean constraint containing a literal ℓ.

First, suppose that the literal ℓ is relevant in χ. Then, $\chi|\ell \not\equiv \chi|\bar{\ell}$. As we always have $\chi|\bar{\ell} \models \chi|\ell$ (the two constraints only differ on the degree, and the latter is smaller), necessarily $\chi|\ell \not\models \chi|\bar{\ell}$, and there is a model M of $\chi|\ell$ that is not a model of $\chi|\bar{\ell}$. Now, observe that $\chi \equiv ((\chi|\ell) \wedge \ell) \vee ((\chi|\bar{\ell}) \wedge \bar{\ell})$. So, if M satisfies ℓ, it is also a model of $(\chi|\ell) \wedge \ell$, and thus of χ. On the contrary, if M falsifies ℓ, it is neither a model of $(\chi|\ell) \wedge \ell$ nor of $((\chi|\bar{\ell}) \wedge \bar{\ell})$, and is thus a counter-model of χ. Hence, changing the value of ℓ in M can make it either a model or a counter-model of χ.

Now, suppose that ℓ is irrelevant in χ. Let us start by proving the following claim.

Claim 19. *Let χ be a pseudo-Boolean constraint and ℓ be a literal of this constraint. If ℓ is irrelevant in χ, then $\chi \equiv \chi|\ell \equiv \chi|\bar{\ell}$.*

Proof. First, observe that, clearly, $\chi \models \chi|\ell$ and $\chi|\bar{\ell} \models \chi$. Now, as ℓ is irrelevant, we also have that $\chi|\ell \equiv \chi|\bar{\ell}$, and in particular, $\chi|\ell \models \chi|\bar{\ell}$. By transitivity of $\models$, we conclude that the three constraints χ, $\chi|\ell$ and $\chi|\bar{\ell}$ are logically equivalent.

□

If ℓ is irrelevant, we thus have that any model M of χ is also a model of $\chi|\bar{\ell}$. Now, observe that, in this case, the fact that M is a model of $\chi|\bar{\ell}$ does not depend on the value it assigns to ℓ (as this constraint neither contains ℓ nor $\bar{\ell}$). Thus, flipping the value of ℓ in M does not affect the fact that it is a model $\chi|\bar{\ell}$, and thus of χ, which ends the proof.

□

An easy consequence of Theorem 10 is that, in a sense, literal relevance is a monotonic property (with respect to the coefficients of the literals in the constraint).

Proposition 35

Let χ be the pseudo-Boolean constraint $\alpha\ell + \sum_{i=1}^{n} \alpha_i \ell_i \geq \delta$ such that ℓ is irrelevant. All literals ℓ_i having a coefficient $\alpha_i \leq \alpha$ in χ are also irrelevant.

Proof. Consider a constraint χ as in the proposition. Let $i_0 \in \{1, \ldots, n\}$ such that $\alpha_{i_0} \leq \alpha$. Towards a contradiction, let us suppose that ℓ_{i_0} is relevant.

By Theorem 10, there exists a model M of χ such that M satisfies ℓ_{i_0} and flipping its value makes M a counter-model of χ. Let us denote by M' this counter-model. As ℓ is irrelevant, we can assume without loss of generality that it is falsified by M, and thus by M'. M satisfies the constraint $\chi|(\bar{\ell} \wedge \ell') \equiv \sum_{i=1}^{n} \alpha_i \ell_i \geq \delta - \alpha'$ (1), and so it is for M', because M and M' coincide on ℓ_i. As ℓ is irrelevant, flipping its value cannot make M' a model of χ. Thus, M' does not satisfy $\chi|(\ell \wedge \bar{\ell}') \equiv \sum_{i=1}^{n} \alpha_i \ell_i \geq \delta - \alpha$ (2). However, because $\alpha' \leq \alpha$, we have (1) $\models$ (2), which is incompatible with the fact that $M' \models$ (1).

□

This proposition is particularly useful for detecting irrelevant literals in a pseudo-Boolean constraint, without having to consider *all* literals in the constraint, as illustrated in the following example.

Example 67 (Example 66 continued)

In the constraint $10a + 5b + 5c + 2d + e + f \geq 15$, one can observe that d is irrelevant. As a consequence, so are e and f, which have lower coefficients. Symmetrically, as c is relevant, so are a and b, which have coefficients that are greater or equal to that of c.

Another consequence of the definitions is given by the following proposition.

Proposition 36

Let χ be an assertive constraint propagating a literal ℓ. In χ, the literal ℓ is necessarily relevant.

Proof. The result is straightforward: by definition, as χ propagates ℓ, then χ becomes falsified if ℓ is falsified, while it can still be satisfied if ℓ is satisfied. By Theorem 10, ℓ is thus relevant. □

The different characteristics of irrelevant literals identified in this section are particularly useful, especially if we want to detect such literals in the constraints handled by a solver.

5.2 Irrelevant Literals in Pseudo-Boolean Solvers

There are two types of constraints in which irrelevant literals may occur in a pseudo-Boolean solver: either in the original constraints (i.e., those of the input formula), or in the constraints learned during conflict analysis. In the latter case, irrelevant literals may be introduced by the application of cutting planes rules, which may infer constraints containing irrelevant literals as long as they do not preserve equivalence. This may happen even if the constraints used to produce them do not contain any such literal, as shown below.

The weakening rule applied to weaken away the literal d from the constraint $3a + 3b + c + d \geq 4$ produces the constraint $3a + 3b + c \geq 3$, which does not depend on c anymore.

The division rule applied to the constraint $6a + 5b + c \geq 6$ by dividing it by 2 leads to the inference of the same constraint as above, i.e., $3a + 3b + c \geq 3$, in which c is irrelevant.

The addition rule applied to the two constraints $4a + 3b + 3c \geq 6$ and $3b + 2a + 2d \geq 3$, produces the constraint $6a + 6b + 3c + 2d \geq 9$, which does not depend on d.

The cancellation rule applied to the two constraints $4b + 3\bar{e} + 3c + 2a \geq 6$ and $4a + 3e + 2b + 2d \geq 6$, to cancel e out, yields the constraint $6a + 6b + 3c + 2d \geq 9$, in which d is irrelevant.

As discussed in Section 4.2, these rules are widely used by pseudo-Boolean solvers during their conflict analysis, so solvers have to deal with constraints containing irrelevant literals (note that, for instance, the literals f and a are irrelevant in Examples 60 and 62, respectively). In particular, the main issue arises when cutting planes rules are applied to these constraints: these rules may cause irrelevant literals to become relevant in the newly inferred constraint. When this occurs, we say that the literal has become *artificially* relevant.

Definition 106 (Artificially Relevant Literal)

Consider the pseudo-Boolean constraints χ and $\chi_1, \ldots, \chi_n$ such that, for some cutting planes rule r, we have:

$$\frac{\chi_1 \qquad \ldots \qquad \chi_n}{\chi}\ (r)$$

A literal ℓ is *artificially relevant* in χ if it is relevant in χ but irrelevant in all the constraints among $\chi_1, \ldots, \chi_n$ in which it appears.

Artificially relevant literals may be produced in different circumstances by pseudo-Boolean solvers, as they accumulate in the constraints derived during conflict analysis. The following example shows how this may happen in a generalized resolution-based solver, such as *Sat4j* [LP10].

Example 68

Suppose that, during the search, a conflict occurs on the constraint χ_1 given by $4a + 4b + 3\bar{e} + 3g + 3h + 2i + 2j \geq 16$, and suppose that e was propagated by $6a + 6b + 4c + 3d + 3e + 2f \geq 10$. The conflict analysis is performed by applying the cancellation rule on e between these two constraints.
Now, suppose that, to preserve the conflict, the solver needs to apply the weakening rule on the reason for e, e.g., on c. This produces the constraint χ_2 given by $6a + 6b + 3d + 3e + 2f \geq 6$, in which f is irrelevant.
Applying the cancellation rule between the conflicting constraint χ_1 and χ_2 produces the constraint $\chi_3 = 10a + 10b + 3d + 3g + 3h + 2f + 2i + 2j \geq 19$, in which f has become artificially relevant.

In division-based solvers, such as *RoundingSat* [EN18], applying the weakening and division rules may also produce artificially relevant literals, as illustrated by the following example.

Example 69

Let us consider the constraint $17a + 17b + 8c + 4d + 2e + 2f \geq 23$ in which all literals are relevant. Suppose that, during the search performed by *RoundingSat*, c and f are satisfied and all other literals are falsified by some propagations. The constraint is now conflictual: to analyze the conflict, *RoundingSat* resolves it against the reason for one of its falsified literals, e.g., the reason for $\bar{d}$.
RoundingSat weakens the constraint on f, as it is not falsified and its coefficient (2) is not divisible by the coefficient of d (4), giving the constraint $\chi_4 = 17a + 17b + 8c + 4d + 2e \geq 21$. Observe that e is now irrelevant.
When *RoundingSat* applies the division by 4, the constraint we obtain is $\chi_5 = 5a + 5b + 2c + d + e \geq 6$, in which all literals are relevant: e has thus become artificially relevant.

Remark 38

In *RoundingSat*, irrelevant literals produced after weakening a reason are always falsified by the current assignment.[4] Indeed, suppose that the literal ℓ it propagates has coefficient α. By construction, all remaining satisfied and unassigned literals have a coefficient that is divisible by α, and thus that is at least equal to α. As ℓ is propagated, it is relevant by Proposition 36, and Proposition 35 tells us that this is also the case for these literals.

As irrelevant literals do not impact the semantics of a pseudo-Boolean constraint, it may seem at first sight that they have little impact on the strength of the reasoning performed by the solver. However, when they become artificially relevant, irrelevant literals may cause the derived constraint to be weaker than it could be, if the irrelevant literals were not there in the first place. This becomes particularly clear when these literals are removed.

5.3 Removing Irrelevant Literals

Given a pseudo-Boolean constraint χ, the main characteristic of any irrelevant literal ℓ that appears in χ is that it can be removed from χ while preserving equivalence. By Definition 105, this can be achieved in two ways: either by locally assigning ℓ to 1 (i.e., computing $\chi|\ell$) or to 0 (i.e., computing $\chi|\bar{\ell}$).

[4] Many thanks to an anonymous reviewer of IJCAI 2020 who pointed out this observation.

Example 70 (Example 66 cont'd)

Let χ be the constraint $10a + 5b + 5c + 2d + e + f \geq 15$, in which d, e and f are irrelevant. The constraint χ is logically equivalent to:

- $\chi|\bar{d}\bar{e}\bar{f} \equiv 10a + 5b + 5c \geq 15 \equiv \chi|def \equiv 10a + 5b + 5c \geq 11$
- $\chi|\bar{d}\bar{f} \equiv 10a + 5b + 5c + e \geq 15 \equiv \chi|df \equiv 10a + 5b + 5c + e \geq 12$
- $\chi|\bar{d}\bar{e} \equiv 10a + 5b + 5c + f \geq 15 \equiv \chi|de \equiv 10a + 5b + 5c + f \geq 12$
- $\chi|\bar{e}\bar{f} \equiv 10a + 5b + 5c + 2d \geq 15 \equiv \chi|ef \equiv 10a + 5b + 5c + 2d \geq 13$
- $\chi|\bar{d} \equiv 10a + 5b + 5c + e + f \geq 15 \equiv \chi|d \equiv 10a + 5b + 5c + e + f \geq 13$
- $\chi|\bar{f} \equiv 10a + 5b + 5c + 2d + e \geq 15 \equiv \chi|f \equiv 10a + 5b + 5c + 2d + e \geq 14$
- $\chi|\bar{e} \equiv 10a + 5b + 5c + 2d + f \geq 15 \equiv \chi|e \equiv 10a + 5b + 5c + 2d + f \geq 14$

However, observe that these constraints are not equivalent *over the reals*, in which case the constraint $10a + 5b + 5c \geq 15$ is the strongest.

5.3.1 Removal by Weakening

A first approach consists in locally assigning the considered irrelevant literals to 1, which is equivalent to weakening away these irrelevant literals. The main advantage of this strategy is that it may sometimes trigger the saturation rule, allowing to maintain small coefficients and thus making arithmetic operations more efficient. The following example shows how this approach may allow to derive stronger constraints, in this case in a generalized resolution-based solver.

Example 71 (Example 68 cont'd)

Recall that f is irrelevant in the constraint $\chi_2 = 6a + 6b + 3d + 3e + 2f \geq 6$. If f is weakened away from χ_2, this constraint becomes $6a + 6b + 3d + 3e \geq 4$, which can be saturated into $\chi_2' = 4a + 4b + 3d + 3e \geq 4$. If this constraint is used in place of χ_2 when applying the cancellation with $\chi_1 = 4a + 4b + 3\bar{e} + 3g + 3h + 2i + 2j \geq 16$, one gets $\chi_3' = 8a + 8b + 3d + 3g + 3h + 2i + 2j \geq 17$, which is strictly stronger than the constraint $\chi_3 = 10a + 10b + 3d + 3g + 3h + 2f + 2i + 2j \geq 19$ we obtained before, i.e., $\chi_3' \models \chi_3$. For instance, $a \wedge b$ is not an implicant of the constraint any longer.

Yet, the weakening-based strategy does not always allow to infer stronger constraints, as illustrated in the following example.

Example 72 (Example 69 cont'd)

If the irrelevant literal e is weakened away from $\chi_4 = 17a + 17b + 8c + 4d + 2e \geq 21$, we get the constraint $\chi_4' = 17a + 17b + 8c + 4d \geq 19$. When the division by 4 is applied on this constraint, it becomes $\chi_5' = 5a + 5b + 2c + d \geq 5$. Observe that c and d are now irrelevant: the constraint is equivalent to $a + b \geq 1$, which is strictly weaker than the constraint $\chi_5 = 5a + 5b + 2c + d + e \geq 6$ that was obtained before, i.e. $\chi_5 \models \chi_5'$.

5.3.2 Simple Removal

As the weakening of irrelevant literals is not completely satisfactory, let us consider their *simple* removal. The second approach is assigning irrelevant literals to 0, i.e., removing them without modifying anything else on the constraint. This approach allows to strengthen the constraint over the reals, although it remains equivalent over the Booleans. Doing so allows to fix the weaker constraint derived in Example 72, as shown below.

Example 73 (Example 72 cont'd)

From $\chi_4 = 17a+17b+8c+4d+2e \geq 21$, removing e gives $\chi_4'' = 17a+17b+8c+4d \geq 21$. When the division by 4 is applied on this new constraint, we get the constraint $\chi_5'' = 5a+5b+2c+d \geq 6$, which is stronger than both the constraints $\chi_5 = 5a+5b+2c+d+e \geq 6$ and $\chi_5' = 5a+5b+2c+d \geq 5$, as we have $\chi_5'' \models \chi_5 \models \chi_5'$.

This approach, however, fails to infer a constraint that is as strong as that of Example 71.

Example 74 (Example 71 cont'd)

When f is removed from $\chi_2 = 6a+6b+3d+3e+2f \geq 6$, we get the constraint $\chi_2'' = 6a+6b+3d+3e \geq 6$. When applying the cancellation rule between this constraint and $\chi_1 = 4a+4b+3\bar{e}+3g+3h+2i+2j \geq 16$, one gets $\chi_3'' = 10a+10b+3d+3g+3h+2i+2j \geq 19$, which is stronger than $\chi_3 = 10a+10b+3d+3g+3h+2f+2i+2j \geq 19$, but weaker than $\chi_3' = 8a+8b+3d+3g+3h+2i+2j \geq 17$, so that $\chi_3' \models \chi_3'' \models \chi_3$. In this case, the weakening-based approach is better.

Consequently, it appears that none of the two approaches for removing irrelevant literals is better than the other for inferring strong constraints in all cases. We thus need a heuristic to determine, given a pseudo-Boolean constraint, whether its irrelevant literals should be removed using the weakening or the simple removal strategy.

5.3.3 Slack-Based Approach

The strength of a pseudo-Boolean constraint can heuristically be approached as the *slack* of this constraint (see Definition 103). Thus, in the case of irrelevant literals, we may consider the slack to decide which removal strategy should be applied. Indeed, recall that the slack is *subadditive* (see Proposition 32). In this context, choosing the strategy that minimizes the slack of the constraint being considered allows to put a tighter upper bound on the slack of the constraints that will be derived later on.

The following examples shows how to apply a case-by-case approach for removing irrelevant literals in the different kinds of solvers we have studied above.

Example 75 (Example 74 cont'd)

Let us consider the slack of the different constraints we obtained after the removal of irrelevant literals in Examples 71 and 74:

- The constraint $\chi_2 = 6a + 6b + 3d + 3e + 2f \geq 6$, obtained without removing irrelevant literals, has slack 14.
- The constraint $\chi_2' = 4a + 4b + 3d + 3e \geq 4$, obtained by weakening away irrelevant literals, has slack 10.
- The constraint $\chi_2'' = 6a + 6b + 3d + 3e \geq 6$, obtained by simply removing irrelevant literals, has slack 12.

The constraint to choose is thus χ_2', as its slack is lower than the others.

Example 76 (Example 73 cont'd)

Let us consider the slack of the different constraints we obtained after the removal of irrelevant literals in Examples 72 and 73:

- The constraint $\chi_4 = 17a + 17b + 8c + 4d + 2e \geq 21$, obtained without removing irrelevant literals, has slack 27.
- The constraint $\chi_4' = 17a + 17b + 8c + 4d \geq 19$, obtained by weakening away irrelevant literals, has slack 27.
- The constraint $\chi_4'' = 17a + 17b + 8c + 4d \geq 21$, obtained by simply removing irrelevant literals, has slack 25.

The constraint to choose is thus χ_4'', as its slack is lower than the others.

Now that we have identified how to remove irrelevant literals to infer stronger constraints, we need an efficient algorithm to detect these literals. Unfortunately, checking whether a literal is relevant in a pseudo-Boolean constraint is NP-complete [CLH11, Theorem 9.26]. Still, as shown in the following section, incomplete approaches can be used to identify *some* irrelevant literals produced by pseudo-Boolean solvers.

5.4 Detecting Irrelevant Literals

To evaluate the impact of irrelevant literals in pseudo-Boolean solvers, we designed an approach for identifying and removing them from pseudo-Boolean constraints. In practice, the NP-completeness of the relevance check makes it unrealistic to systematically remove *all* irrelevant literals. We thus need to find efficient ways for detecting these literals heuristically. This starts by carefully choosing *when* irrelevant literals should be looked for, so as to make as few checks as possible.

5.4.1 When to Detect Irrelevant Literals

As relevance checks are costly, our goal is here to minimize the number of checks we need to perform. First, an easy optimization is to take advantage of Proposition 35: only one check per coefficient is required, and once a relevant literal is identified, all literals having a coefficient greater than that of the relevant literals are also relevant.

Then, we must apply the relevance check *after* irrelevant literals have been introduced and *before* they become artificially relevant. For instance, we know that irrelevant literals may be introduced by the weakening rule. The following proposition ensures that this rule cannot make any irrelevant literal artificially relevant.

Proposition 37

Let χ be a pseudo-Boolean constraint containing a literal ℓ. If ℓ is irrelevant, then ℓ is irrelevant in any constraint χ' obtained by weakening χ on any literal $\ell' \neq \ell$.

Proof. Let χ be the pseudo-Boolean constraint $\alpha\ell + \alpha'\ell' + \sum_{i=1}^{n} \alpha_i \ell_i \geq \delta$. Suppose that ℓ is irrelevant in χ. Let χ' be the pseudo-Boolean constraint obtained by weakening χ on ℓ', i.e., $\chi' = \alpha\ell + \sum_{i=1}^{n} \alpha_i \ell_i \geq \delta - \alpha'$. Towards a contradiction, suppose that ℓ is relevant in χ'.

By Theorem 10, there exists a model M of χ' such that flipping the value of ℓ in M makes it a counter-model of χ'. Observe that such a model necessarily satisfies ℓ. In other words, $M \models \sum_{i=1}^{n} \alpha_i \ell_i \geq \delta - \alpha' - \alpha$ and $M \not\models \sum_{i=1}^{n} \alpha_i \ell_i \geq \delta - \alpha'$. As ℓ' does not appear in these constraints, we can suppose that M satisfies ℓ' (otherwise, we can modify M without changing the statements above).

In this case, we have that $M \models \alpha'\ell' + \sum_{i=1}^{n} \alpha_i \ell_i \geq \delta - \alpha$ and $M \not\models \alpha'\ell' + \sum_{i=1}^{n} \alpha_i \ell_i \geq \delta$, i.e., M is a model of χ for which flipping the value of ℓ makes it a counter-model of χ, which contradicts that ℓ is irrelevant in χ.

□

Proposition 37 allows to detect irrelevant literals after having applied multiple weakening operations: it allows to perform the check once all weakening operations have been performed, while ensuring that no artificially relevant literals were introduced by these operations (otherwise, we would have needed to perform the check each time a single literal is weakened away). This is particularly useful in generalized resolution-based solvers, which successively apply the weakening rule. However, in this context, the removal of irrelevant literals may lose the conflict that was restored by the application of the weakening rule (and thus may require another weakening step), as shown in the following example.

Example 77

Consider the pseudo-Boolean constraint $11a(1@3) + 5b(0@3) + 5c(0@2) + 5d(?@?) + 2e(0@1) + 2f(?@?) \geq 11$, which has slack 7 and is the reason for a. Consider also the constraint $7\bar{a}(0@3) + 6h(0@3) + 3i(?@?) + 2j(?@?) \geq 9$, which has slack -4, and is thus conflicting.
If we estimate the slack of the constraint obtained when applying the cancellation rule, we get $7 \times 7 - 4 \times 11 = 5$, and one can verify that this is indeed the slack of the constraint we get. As a consequence, the conflict is not preserved, and f is weakened away from the reason, giving the constraint $9a(1@3) + 5b(0@3) + 5c(0@2) + 5d(?@?) + 2e(0@1) \geq 9$, which has slack 5. If we estimate the slack of the constraint we obtain now, we get the bound $5 \times 7 - 4 \times 9 = -1$, and one can verify that this is indeed the slack of the constraint we get. The cancellation applied to the two constraints will thus preserve the conflict. However, observe that e is irrelevant in the new reason, and can thus be weakened away, giving the constraint $7a(1@3) + 5b(0@3) + 5c(0@2) + 5d(?@?) + \geq 7$, which still has slack 5.

Now, if we estimate the slack of the constraint we obtained after having removed the irrelevant literal, we get $5 - 4 = 1$, and one can verify that this is indeed the slack of the constraint we get. In other words, the conflict is not preserved anymore after the removal of the irrelevant literals, so that a new literal must be weakened away to preserve the conflict.

In the case of solvers based on *RoundingSat* [EN18], we need to pay attention to two main properties of the proof system regarding irrelevant literals. The first one is that the division rule always turns irrelevant literals into artificially relevant literals when applied on the reason.

Proposition 38

In *RoundingSat*-based solvers, irrelevant literals produced after weakening a reason are always made artificially relevant by the division performed on this reason afterwards.

Proof. Observe that the division applied on the reason ensures that the coefficient α of the propagated literal ℓ is equal to 1 while preserving the propagation of ℓ. As such, by Proposition 36, ℓ remains relevant after the application of the division. Moreover, recall that the division rounds all coefficients that are smaller than α, including those of the irrelevant literals to 1. As a consequence, these literals are also relevant after the application of the division by Proposition 35.

□

This proposition forces to run the detection of irrelevant literals in the reason *after* the weakening operation, and *before* the division. On the conflict side, things are quite different. In particular, the following example shows that the pivot of the cancellation may become irrelevant in the conflicting constraint after its weakening.

Example 78

Let us consider the constraint $\bar{a}(?@?) + \bar{b}(?@?) + \bar{c}(?@?) + f(0@3) \geq 3$, which propagates $\bar{a}$, $\bar{b}$ and $\bar{c}$ under the current partial assignment. These propagations falsify the constraint $3a(0@3) + 3b(0@3) + 2c(0@3) + d(1@1) + e(1@2) \geq 5$.
If the cancellation rule is applied on the literal c, then the literals d and e are weakened away from the conflicting constraint, as these literals are satisfied and their coefficient (1) is not divisible by the coefficient of c (2), giving the constraint $3a(0@3) + 3b(0@3) + 2c(0@3) \geq 3$. In this new conflicting constraint, the pivot c has become irrelevant.

When this occurs, it seems natural to abort the cancellation step that is being performed, and move on to the next literal to cancel in the implication graph. Indeed, if the literal becomes irrelevant after the weakening, it does not play any role in the conflict, as weakening it preserves the conflict, so that resolving on it does not make much sense. However, as the weakening step has already been performed, we also need to rollback the weakening operation: this operation would have been made without any reason, while we prefer to keep the constraints as strong as possible. Moreover, the application of the division rule does not make sense in this case, as the pivot has now a coefficient equal to 0 (it has been removed from the constraint).

Considering the different observations made in this subsection and the cutting planes rules used in pseudo-Boolean solvers, the relevance check must be performed after the weakening step applied to preserve the conflict (and before the application of the division, if any) and also after the application of the cancellation rule. We now need to define *how* to perform this check.

5.4.2 SAT-Based Relevance Check

First, let us study how to identify whether a literal is relevant. For the sake of illustration, the following pseudo-Boolean constraint χ, in which we would like to decide whether ℓ is relevant, will be used as running example:

$$\chi = \alpha\ell + \sum_{i=1}^{n} \alpha_i \ell_i \geq \delta$$

Recall that ℓ is irrelevant if and only if $\chi|\ell \equiv \chi|\bar{\ell}$ or, said differently:

$$\sum_{i=1}^{n} \alpha_i \ell_i \geq \delta - \alpha \equiv \sum_{i=1}^{n} \alpha_i \ell_i \geq \delta$$

Note that, because $\delta > \delta - \alpha$ holds, the latter constraint trivially entails the former, so the only check to be performed is the following:

$$\sum_{i=1}^{n} \alpha_i \ell_i \geq \delta - \alpha \models \sum_{i=1}^{n} \alpha_i \ell_i \geq \delta$$

To check whether ℓ is relevant, we thus need to check whether the statement above holds, which can be achieved by determining the unsatisfiability of this conjunction of constraints:

$$\sum_{i=1}^{n} \alpha_i \ell_i \geq \delta - \alpha \wedge \neg \left(\sum_{i=1}^{n} \alpha_i \ell_i \geq \delta \right) \equiv \sum_{i=1}^{n} \alpha_i \ell_i \geq \delta - \alpha \wedge \sum_{i=1}^{n} \alpha_i \ell_i < \delta$$

This conjunction is equivalent (after normalization) to the following conjunction of pseudo-Boolean constraints.

$$\sum_{i=1}^{n} \alpha_i \ell_i \geq \delta - \alpha \wedge \sum_{i=1}^{n} \alpha_i \bar{\ell}_i \geq \sum_{i=1}^{n} \alpha_i - \delta$$

This formula is unsatisfiable if and only if the literal ℓ is irrelevant. Thus, one can just use one's favorite pseudo-Boolean solver to detect irrelevant literals.

Remark 39

Note that the fact that there are only two constraints does not mean that this problem is easy to solve (subset-sum is basically a two-normalized-constraint problem, see, e.g., [RM09a, Section 22.4]).

Let us evaluate the impact of the removal of irrelevant literals described above in the pseudo-Boolean solver *Sat4j* [LP10]. In practice, there is no guarantee about the time needed to solve the pseudo-Boolean problem used to detect irrelevant literals, especially considering the remark above. In order to remain efficient, a timeout of 5 seconds is set to each call to the solver, so that when it cannot find an answer within the time limit, the literal is assumed relevant. Thus, the approach remains sound while being incomplete. Table 5.1 shows, for each family, the number of instances solved by *Sat4j* with the removal of irrelevant literals turned on. Figures 5.1 and 5.2 show the number of irrelevant literals detected for each family when using the solvers *Sat4j-GeneralizedResolution* and *Sat4j-RoundingSat*, respectively, executed in the usual experimental setting (see Appendix B). The timeout was set to 1800 seconds and the memory limit to 32 GB.

Family	Number of instances in the family	*Sat4j GeneralizedResolution*	*Sat4j RoundingSat*
`Aardal_1`	14	14	14
`armies`	12	0	0
`caixa`	1	1	1
`d_n_k`	234	156	159
`d-equals-n_k`	70	27	29
`EC_ODD_GRIDS`	25	11	12
`EC_RANDOM_GRAPHS`	22	5	8
`FPGA_SAT05`	57	34	33
`heinz`	4	0	0
`Instances3col_OPB`	26	5	6
`liu`	20	16	16
`lopes`	193	0	0
`nossum`	180	0	0
`oliveras`	4080	2646	2630
`ppp-problems`	6	0	0
`rand6reg`	33	6	11
`robin`	6	2	3
`roussel`	40	22	22
`sroussel`	122	0	0
`subsetcard`	56	56	56
`SUMINEQ`	24	1	3
`tsp`	100	0	5
`uclid_pb_benchmarks`	50	30	35
`vertexcover-instances`	107	80	82
`wnqueen`	100	36	98

Table 5.1: Table summarizing, for both *Sat4j-GeneralizedResolution* and *Sat4j-RoundingSat*, the number of instances solved in each family by the solver when irrelevant literals are detected with a call to a SAT solver.

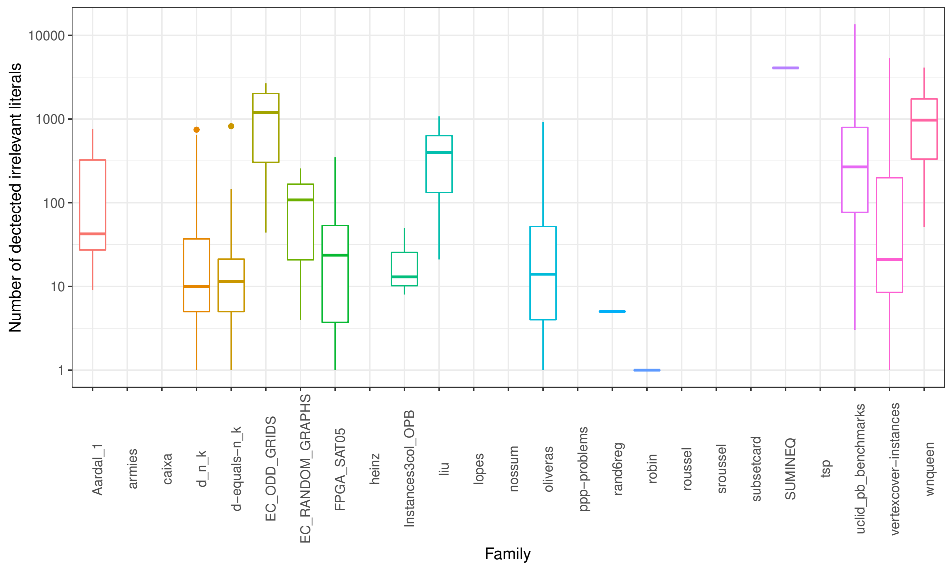


Figure 5.1: Boxplots of the number of irrelevant literals detected using the SAT-based algorithm in each family in *Sat4j-GeneralizedResolution*, (logarithmic scale). Each family has its own box, for which the horizontal bars represent the quartiles and the vertical bars the estimated minimum and maximum. Points represent outlier, i.e., instances for which the number of detected irrelevant literals is either below or above the estimated minimum or maximum, respectively.

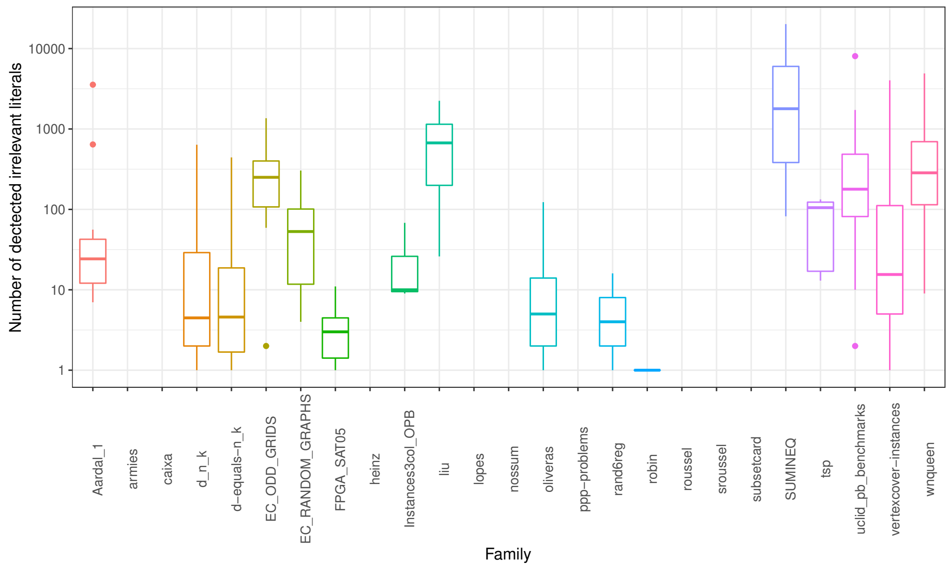


Figure 5.2: Boxplots of the number of irrelevant literals detected using the SAT-based algorithm in each family in *Sat4j-RoundingSat* (logarithmic scale). Each family has its own box, for which the horizontal bars represent the quartiles and the vertical bars the estimated minimum and maximum. Points represent outlier, i.e., instances for which the number of detected irrelevant literals is either below or above the estimated minimum or maximum, respectively.

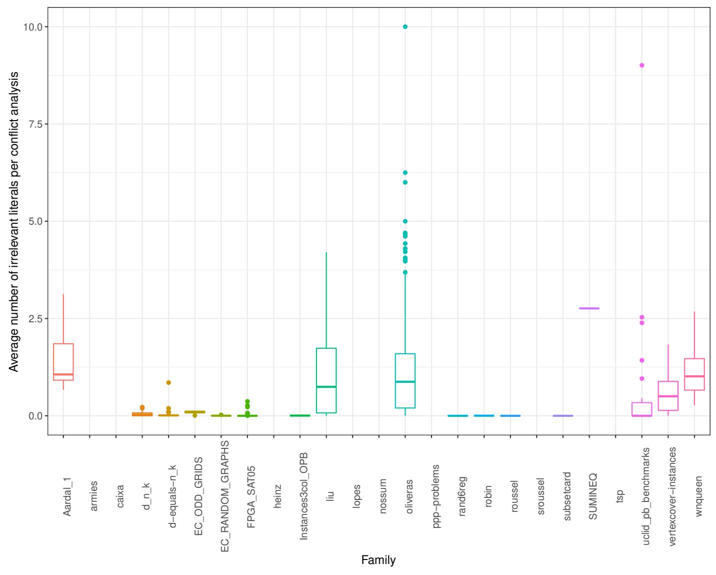


Figure 5.3: Boxplots of the average number of irrelevant literals detected per conflict analysis using the SAT-based algorithm in each family in *Sat4j-GeneralizedResolution* (logarithmic scale). Each family has its own box, for which the horizontal bars represent the quartiles and the vertical bars the estimated minimum and maximum. Points represent outlier, i.e., instances for which the number of detected irrelevant literals is either below or above the estimated minimum or maximum, respectively.

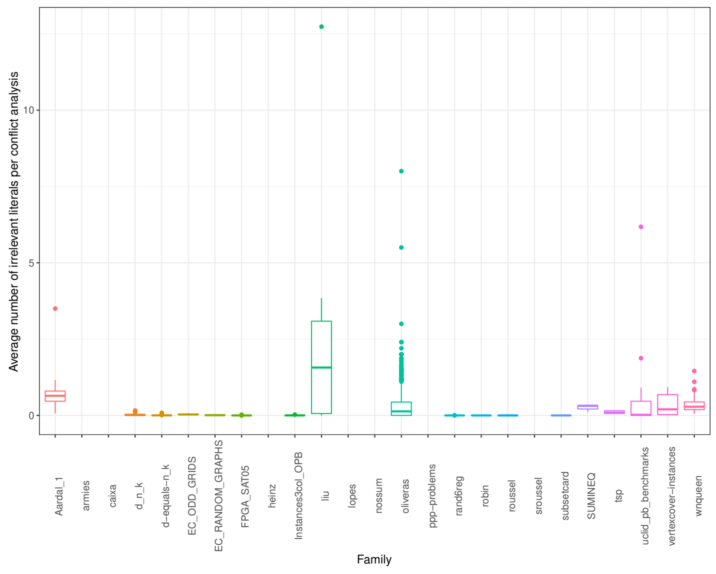


Figure 5.4: Boxplots of the average number of irrelevant literals detected per conflict analysis using the SAT-based algorithm in each family in *Sat4j-RoundingSat* (logarithmic scale). Each family has its own box, for which the horizontal bars represent the quartiles and the vertical bars the estimated minimum and maximum. Points represent outlier, i.e., instances for which the number of detected irrelevant literals is either below or above the estimated minimum or maximum, respectively.

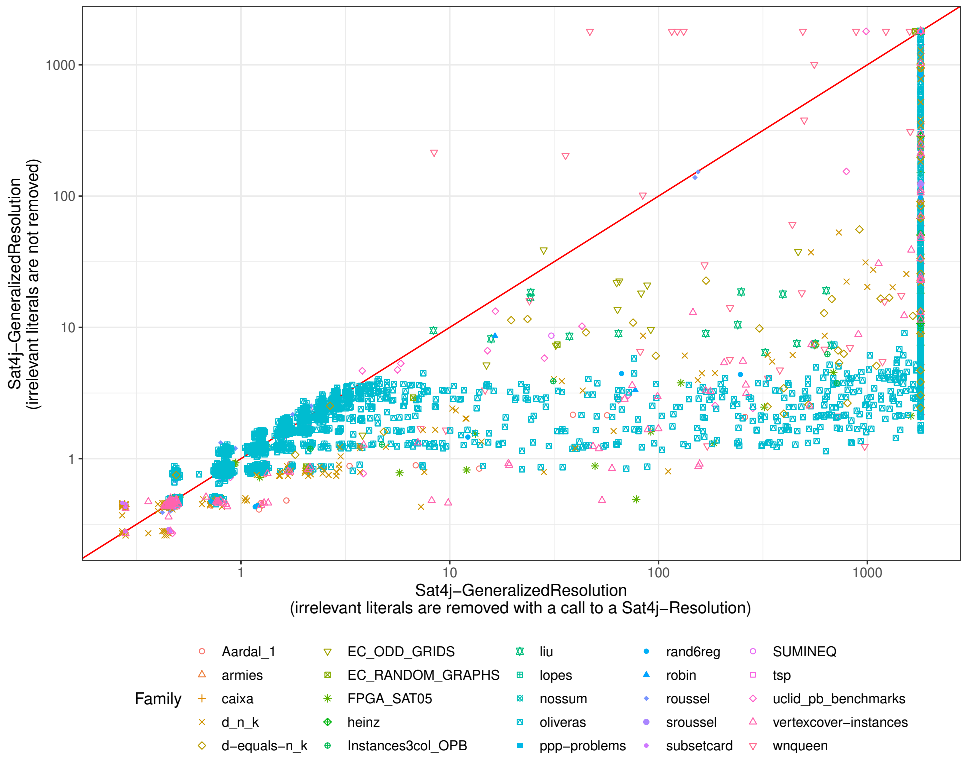


Figure 5.5: Scatter plot comparing the runtime (in seconds) of *Sat4j-GeneralizedResolution* with and without the SAT-based removal of irrelevant literals turned on (logarithmic scale).

These boxplots not only show that the detection algorithm allows in practice to detect irrelevant literals produced during conflict analysis by *Sat4j-GeneralizedResolution* and *Sat4j-RoundingSat*, but also that such literals are produced by both solvers in most of the families that have been considered. Moreover, the number of irrelevant literals being detected is quite similar in the two configurations. We can also observe, in the boxplots given in Figures 5.3 and 5.4, that there are not so many irrelevant literals identified per conflict analysis in average. This is because, most of the time, irrelevant literals are not produced during *all* conflict analyses, but only during *some* of them. We can however see that, for instance, in the `oliveras` family, the average number of irrelevant literals per conflict analysis remains high. This is most likely because, in this particular family, irrelevant literals are already present in the *original* constraints, and these literals are not removed once and for all for these constraints. They thus need to be removed each time an original constraint with irrelevant literals is involved in a conflict analysis. We chose not to remove irrelevant literals from the original constraints because their presence in these constraints is very rare: the `oliveras` family is the only family of our benchmarks that originally contains such literals.

Obviously, the detection algorithm implemented here is however too costly in practice to improve the runtime of the solver with the removal of irrelevant literals as shown in Figures 5.5 and 5.6, and as suggested by the number of solved instances shown in Table 5.1.

Except for some instances (especially those of the `wnqueen` family in the case of *Sat4j-GeneralizedResolution*), the runtime of the solver is highly impacted by the cost of the detection of irrelevant literals. In particular, most of the runtime is spent for this detection in many families, as illustrated in Figures 5.7 and 5.8.

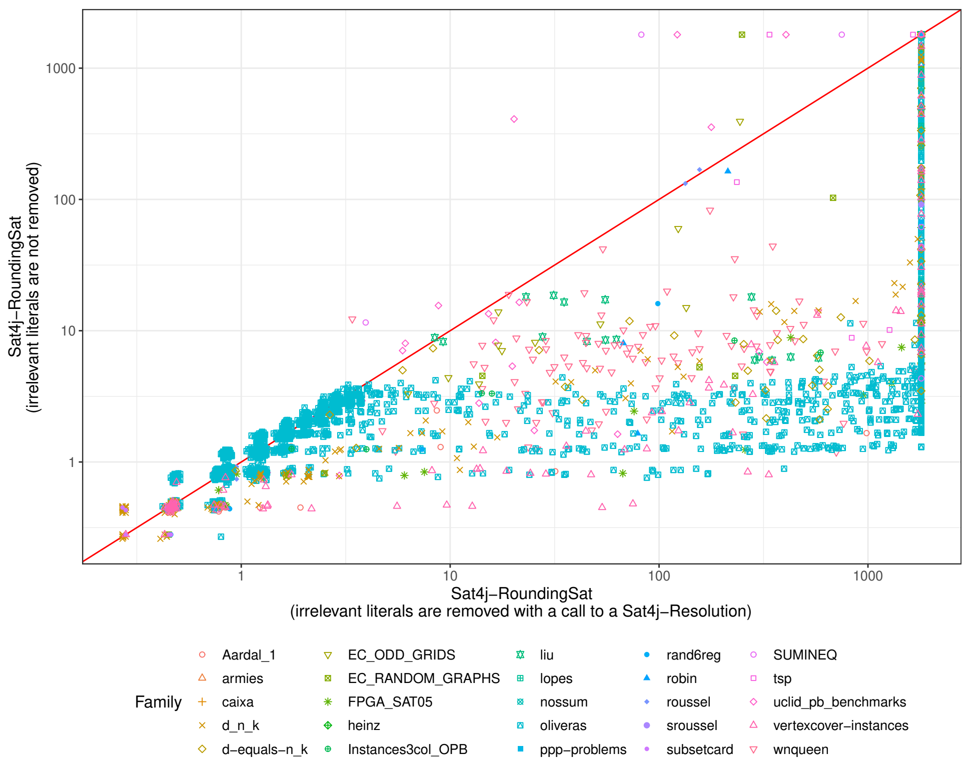


Figure 5.6: Scatter plot comparing the runtime (in seconds) of *Sat4j-RoundingSat* with and without the SAT-based removal of irrelevant literals turned on (logarithmic scale).

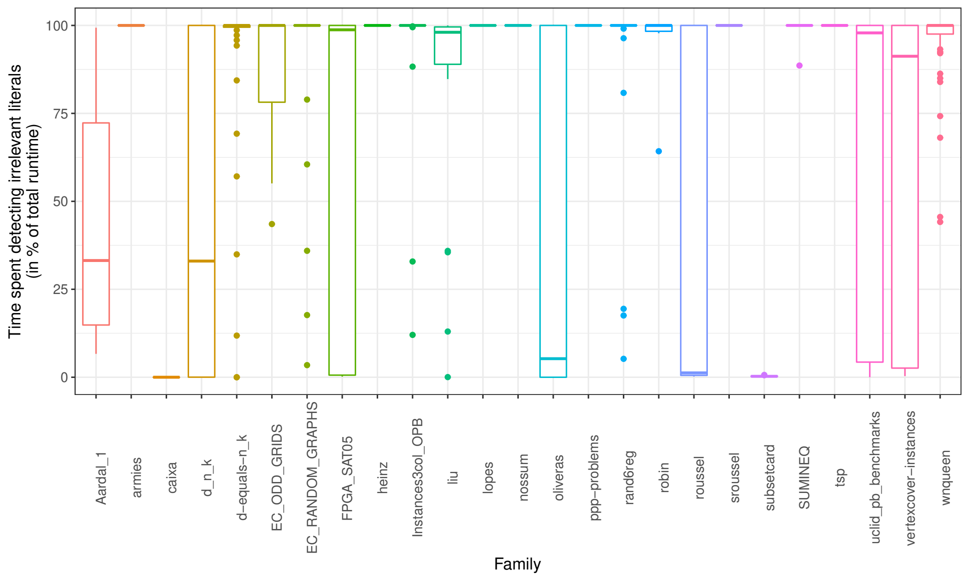


Figure 5.7: Boxplots of the percentage of the running time spent detecting irrelevant literals in *Sat4j-GeneralizedResolution* using the SAT-based algorithm, for each family. Each family has its own box, for which the horizontal bars represent the quartiles and the vertical bars the estimated minimum and maximum. Points represent outlier, i.e., instances for which the percentage of the running time is either below or above the estimated minimum or maximum, respectively.

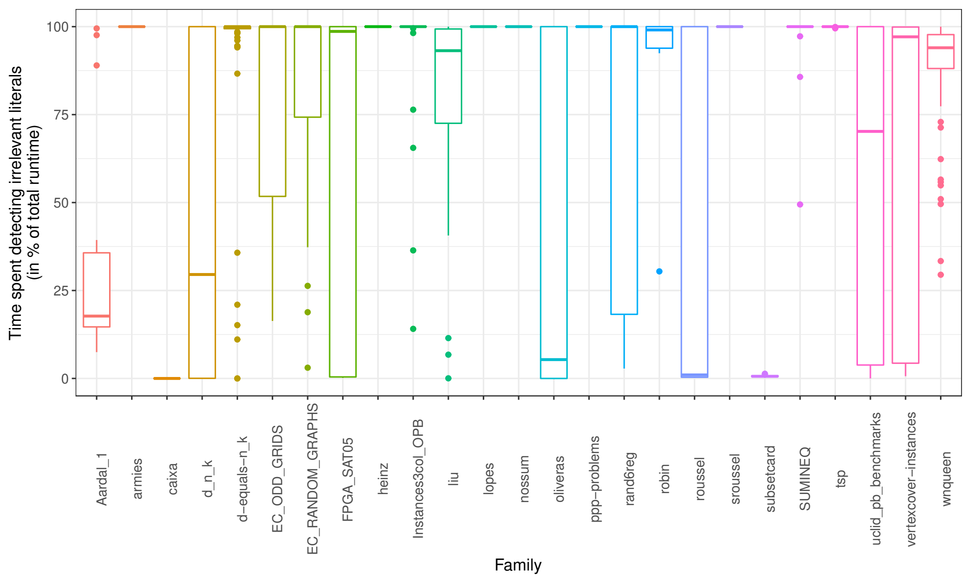


Figure 5.8: Boxplots of the percentage of the running time spent detecting irrelevant literals in *Sat4j-RoundingSat* using the SAT-based algorithm, for each family. Each family has its own box, for which the horizontal bars represent the quartiles and the vertical bars the estimated minimum and maximum. Points represent outlier, i.e., instances for which the percentage of the running time is either below or above the estimated minimum or maximum, respectively.

As shown by these experiments, irrelevant literals are indeed produced by pseudo-Boolean solvers during conflict analysis. However, the SAT-based detection algorithm is too costly in practice to detect them efficiently enough. This has motivated the development of an *ad hoc* incomplete algorithm offering runtime guarantees.

5.4.3 Relevance Check Based on Dynamic Programming

Recall that ℓ is irrelevant in χ if and only if

$$\sum_{i=1}^{n} \alpha_i \ell_i \geq \delta - \alpha \models \sum_{i=1}^{n} \alpha_i \ell_i \geq \delta$$

Observe that this statement holds if and only if there is no interpretation of $\sum_{i=1}^{n} \alpha_i \ell_i$ equal to any number between $\delta - \alpha$ and $\delta - 1$, as otherwise, satisfying ℓ in this interpretation will make it a model of χ. Thus, checking that ℓ is irrelevant is equivalent to checking that there is no subset of $\alpha_1, ..., \alpha_n$ whose sum equals any of these numbers, i.e., solving an instance of the subset-sum problem for each of these inputs.

It is folklore that this can be done in time $O(n\delta)$ using dynamic programming [CLRS09, Chapter 34.5], that is pseudopolynomial in the encoding size. However, in our context, both n and δ may be very large, and it would be very inefficient to solve subset-sum on such inputs. As a workaround, we present an approach for solving subset-sum *incompletely*. Our detection algorithm needs to ensure that there is *no* solution to the considered subset-sum instance in order to correctly detect irrelevant literals, even though some of them may be missed. To this end, we introduce a detection algorithm based on solving subset-sum *modulo* a given positive integer p (fixed for all applications of this algorithm). Since modular arithmetic is compatible with addition, one can ensure that, if there is a solution for the subset-sum problem with the original values, this solution is also a solution of the subset-sum problem considered modulo p.

Example 79 (Example 70 cont'd)

Take the constraint $10a + 5b + 5c + 2d + e + f \geq 15$. If we want to check the relevance of d in this constraint, the multiset of coefficients to be considered is $\{10, 5, 5, 1, 1\}$ (as d is ignored for the purpose of the check).
First, let us consider $p = 6$. The multiset of coefficients modulo p is $\{4, 5, 5, 1, 1\}$. The set of all possible subset sums modulo 6 is thus $\{0, 1, 2, 3, 4, 5\}$, and d is wrongly detected as relevant, since there exists a subset sum equal to $1 \equiv 13 \mod 6$.
If we now consider $p = 5$, the multiset of coefficients becomes $\{0, 0, 0, 1, 1\}$, and the possible subset sums modulo 5 are $\{0, 1, 2\}$. It is thus impossible to find any sum equal to either $3 \equiv 13 \mod 5$ or $4 \equiv 14 \mod 5$, so e is irrelevant. As a consequence, e can also be removed, as it has a smaller coefficient than d, and so does f, which has the same coefficient as e.
As c is relevant, it is detected as such by our algorithm, whatever the value of p, as it never gives the wrong answer for relevant literals. All remaining literals are thus relevant, so that there is no more literals to remove.

By choosing a "good" value for p, we can thus use the classical dynamic programming algorithm and perform quite efficiently the relevance check. However, as an incomplete approach, some irrelevant literals may be wrongly detected as relevant. To limit the number of wrong answers while remaining efficient enough, we can perform multiple relevance check with different small prime numbers, as in the Chinese remainder theorem.[5] In this case, the detection algorithm consists in applying the incomplete subset-sum algorithm described above on the literal to check with different numbers (in practice, we used the numbers 101, 199, 307 and 401). If, for one of these numbers, no solution to the corresponding subset-sum problem exists, then the literal is irrelevant.

Let us now experiment this detection algorithm in different configurations of *Sat4j*, using the same experimental setting as above. Figures 5.9 and 5.10 show a comparison of the number of irrelevant literals detected by the *ad hoc* algorithm compared to those detected by the SAT-based approach previously studied.

We can see that there is only little difference in the number of detected literals, except for the `wnqueen` family, for which the SAT-based approach allows to detect more literals (especially in the case of *Sat4j-GeneralizedResolution*). However, the main advantage of the *ad hoc* algorithm is that it is much faster in practice, as illustrated by Figures 5.11 and 5.12.

Quite interestingly, the scatter plots show that, even though the *ad hoc* algorithm is faster in general, the SAT-based approach performs better on the instances of the `wnqueen` family, which is precisely that on which we observe a significant difference in the number of detected irrelevant literals. On this particular family, the SAT-based approach seems to run faster, and thus allows to detect more irrelevant literals (recall that, if the SAT-based approach is too slow, irrelevant literals may be wrongly detected as relevant when the solver reaches the 5-second timeout).

[5]Many thanks to an anonymous reviewer of IJCAI 2020 who pointed this approach.

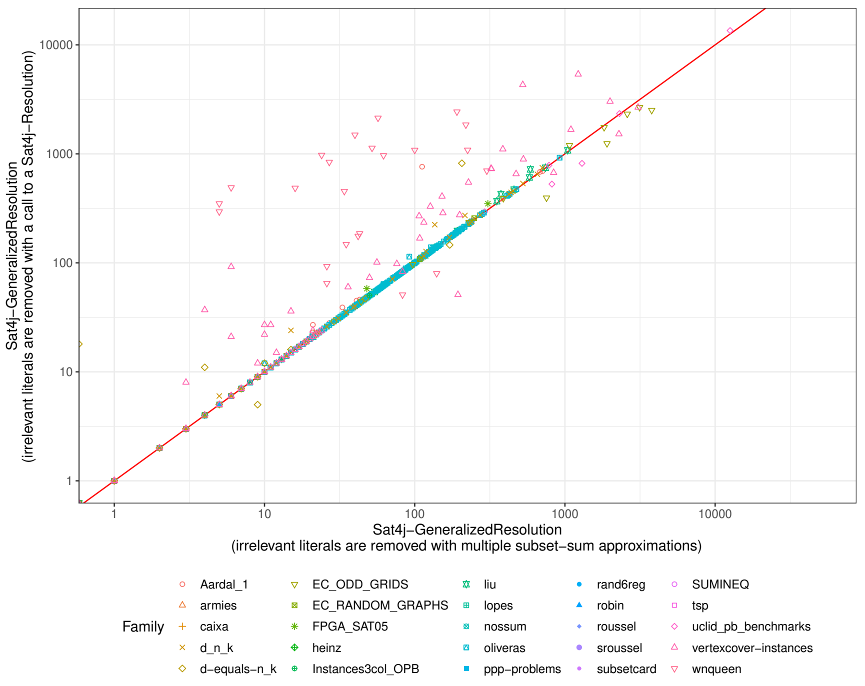


Figure 5.9: Scatter plot comparing the number of irrelevant literals detected using the *ad hoc* algorithm and the SAT-based approach in *Sat4j-GeneralizedResolution* (logarithmic scale).

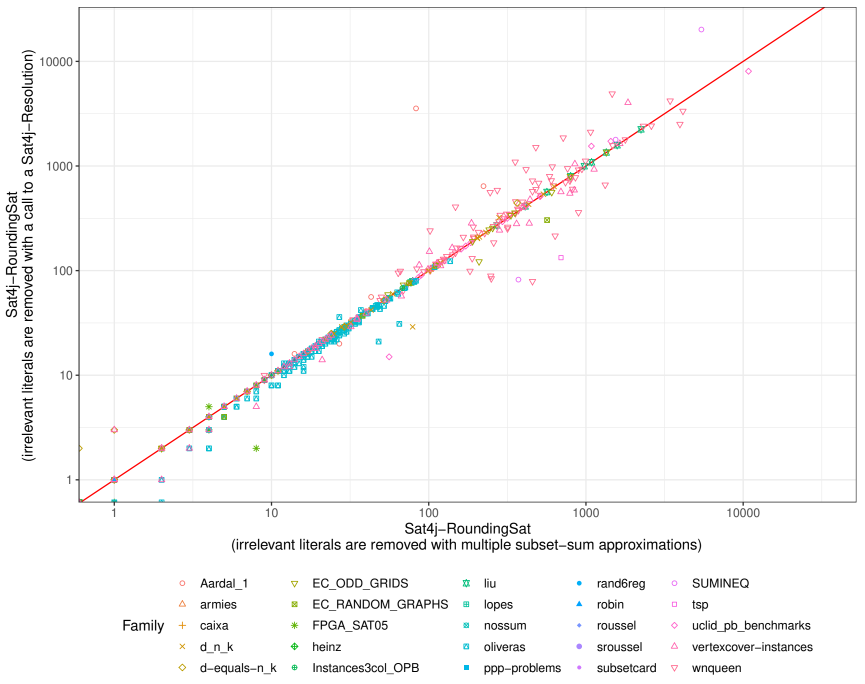


Figure 5.10: Scatter plot comparing the number of irrelevant literals detected using the *ad hoc* algorithm and the SAT-based approach in *Sat4j-RoundingSat* (logarithmic scale).

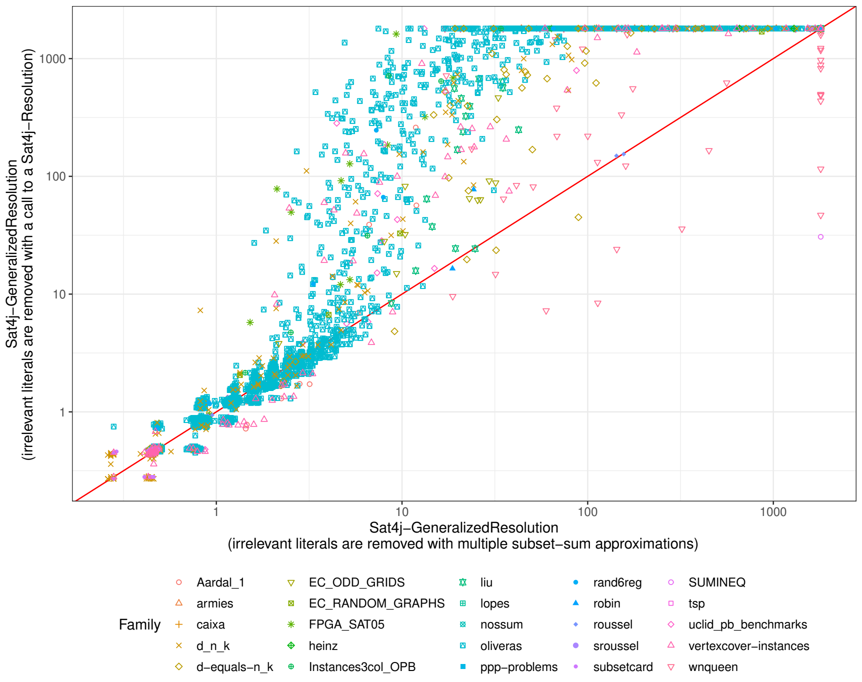


Figure 5.11: Scatter plot comparing the runtime (in seconds) of the *ad hoc* and SAT-based detection algorithms integrated in *Sat4j-GeneralizedResolution* (logarithmic scale).

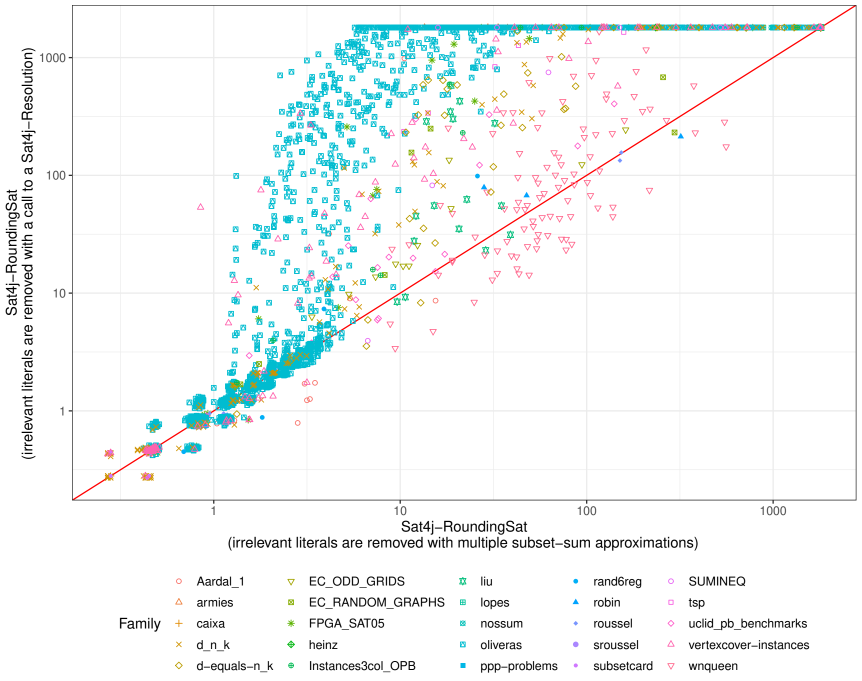


Figure 5.12: Scatter plot comparing the runtime (in seconds) of the *ad hoc* and SAT-based detection algorithms integrated in *Sat4j-RoundingSat* (logarithmic scale).

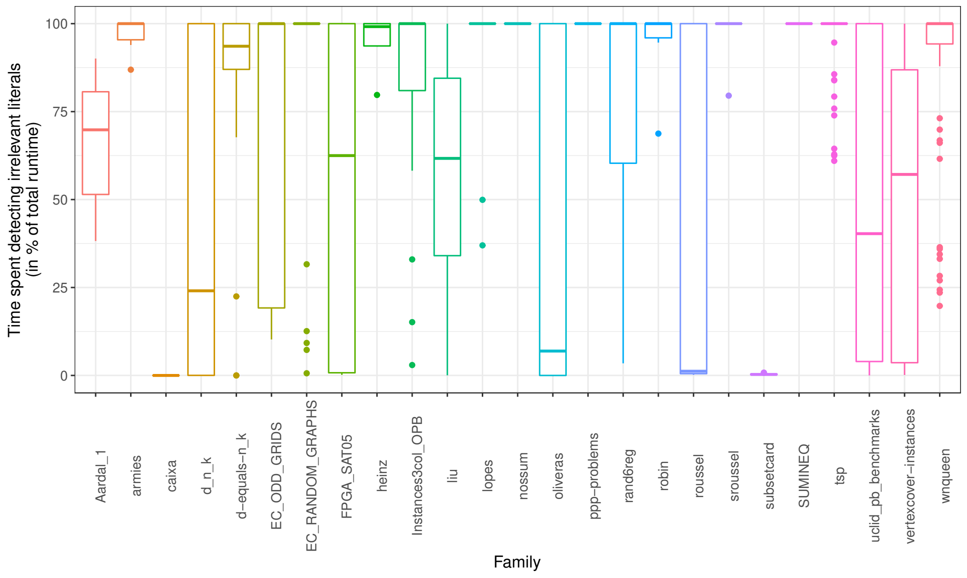


Figure 5.13: Boxplots of the percentage of the running time spent detecting irrelevant literals in *Sat4j-GeneralizedResolution* using the *ad hoc* algorithm, for each family. Each family has its own box, for which the horizontal bars represent the quartiles and the vertical bars the estimated minimum and maximum. Points represent outlier, i.e., instances for which the percentage of the running time is either below or above the estimated minimum or maximum, respectively.

However, the approach remains slow, and most of the runtime is still spent running the detection algorithm, as shown in Figures 5.13 and 5.14.

Because of the high cost of the detection, it is hard to evaluate the impact of the removal of irrelevant literals. This is why we first consider the *ideal* runtime of the solver to estimate this impact: Figures 5.15 and 5.16 consider the runtime of the solver as if the detection were made at no cost. More precisely, for each solved instances, the *ideal* runtime is obtained by removing the time spent detecting irrelevant literals from the total runtime. As the ideal runtime does not make sense when the solver reaches a timeout, the ideal runtime is not computed on instances for which the solver did not give an answer within the time limit (the timeout is left as runtime).

The scatter plots do not show a clear difference between the case in which irrelevant literals are removed and that in which they are not, even though we can still see that the solver remains in general faster when the removal of irrelevant literals is disabled. However, note that the ideal runtime may be biased, as it is based on the detection time measured by the solver itself: as such, the solver can only measure the *wall clock* time spent detecting such literals, while the execution environment, which gives the overall runtime, measures the CPU time more accurately. As a workaround, we now consider, in Figures 5.17 and 5.18 the number of cancellations applied during conflict analysis to evaluate the performance of the solver.

At first sight, it seems that there is not a great difference between the two solvers. However, remember that our approach is incomplete and that we may thus wrongly detect as relevant literals that are actually irrelevant. To really evaluate the impact of irrelevant literals on the solver performance, we would need to remove all of them. However, in practice, a complete approach is clearly unreasonable: our algorithm manages to deal with constraints having a degree up to 10^{410415} which is out of reach of any complete approach.

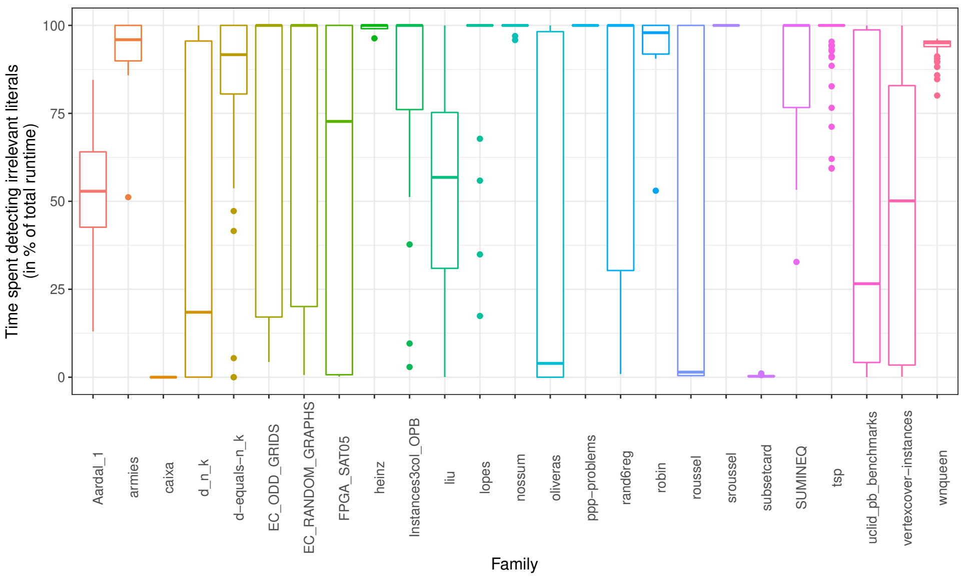


Figure 5.14: Boxplots of the percentage of the running time spent detecting irrelevant literals in *Sat4j-RoundingSat* using the *ad hoc* algorithm, for each family. Each family has its own box, for which the horizontal bars represent the quartiles and the vertical bars the estimated minimum and maximum. Points represent outlier, i.e., instances for which the percentage of the running time is either below or above the estimated minimum or maximum, respectively.

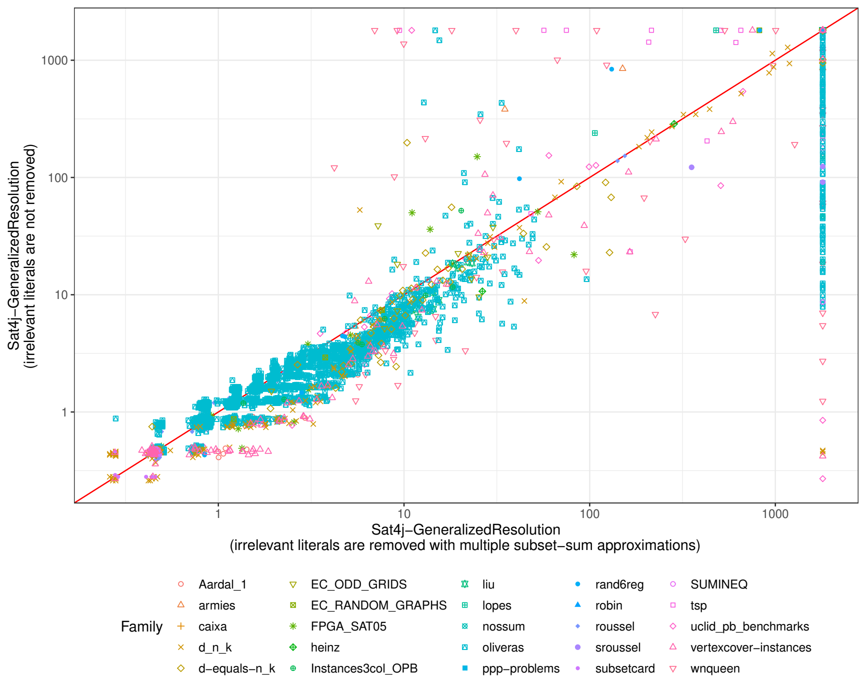


Figure 5.15: Scatter plot comparing the ideal runtime (in seconds) of the execution of *Sat4j-GeneralizedResolution* with the removal of irrelevant literals activated (with the *ad hoc* algorithm) and without it (logarithmic scale).

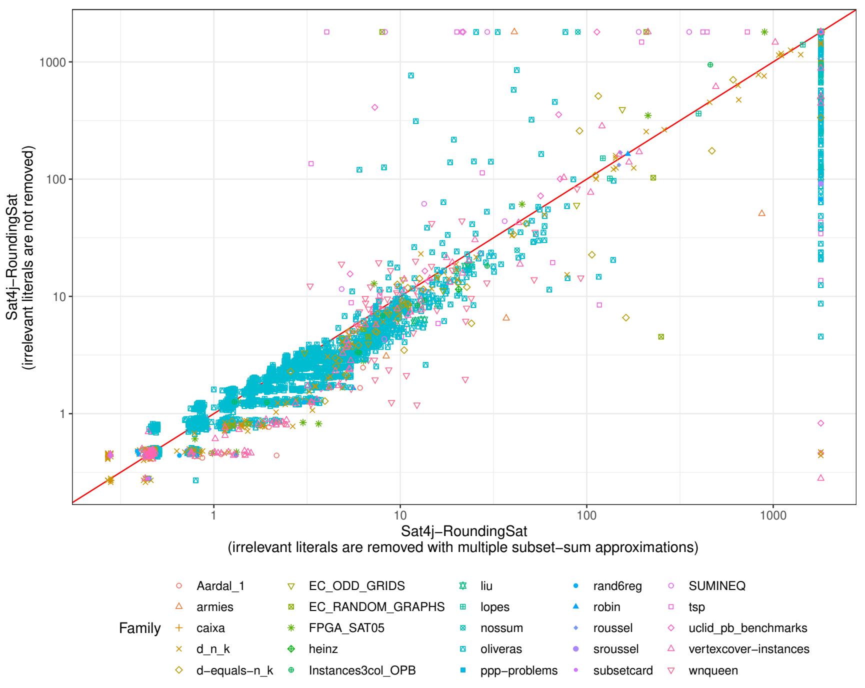


Figure 5.16: Scatter plot comparing the ideal runtime (in seconds) of the execution of *Sat4j-RoundingSat* with the removal of irrelevant literals activated (with the *ad hoc* algorithm) and without it (logarithmic scale).

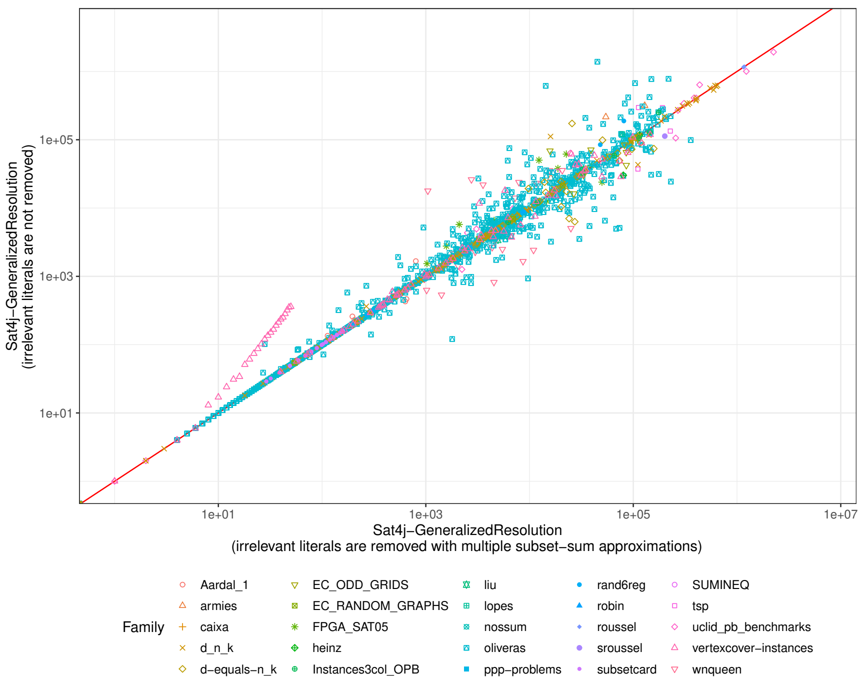


Figure 5.17: Scatter plot comparing the number of cancellations performed during the execution of *Sat4j-GeneralizedResolution* with the removal of irrelevant literals activated (with the *ad hoc* algorithm) and without it (logarithmic scale).

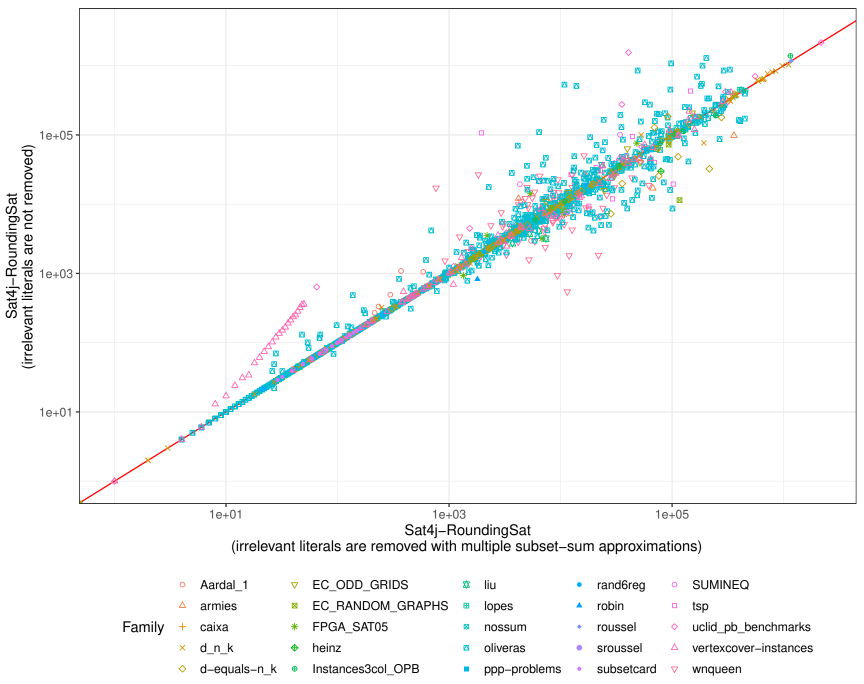


Figure 5.18: Scatter plot comparing the number of cancellations performed during the execution of *Sat4j-RoundingSat* with the removal of irrelevant literals activated (with the *ad hoc* algorithm) and without it (logarithmic scale).

Also, note that removing irrelevant literals impacts the heuristic used to select the variables to assign. When irrelevant literals are removed, the associated variables are not bumped anymore, which alters the behavior of the heuristic, and may have unexpected side effects as this heuristic is not fully understood [EGG+18]. In particular, it is hard to evaluate the impact of bumping irrelevant literals. On the one hand, one could argue that, because these literals are irrelevant, they do not play any role in the conflict. On the other hand, if they were relevant at some point, then their assignment may have triggered some propagations, and, in such a case, they may actually have contributed to the falsification of the constraint.

Yet, our algorithm allows to get a better understanding of irrelevant literals produced by pseudo-Boolean solvers. First, let us observe that, besides their different proof systems, *Sat4j-GeneralizedResolution* and *Sat4j-RoundingSat* both produce irrelevant literals. Even though *Sat4j-RoundingSat* is quite different compared to the original *RoundingSat* (see Appendix A), we argue that this solver also produces irrelevant literals. In Figures 5.19, 5.20 and 5.21, we show the percentage of irrelevant literals that were detected after the weakening operation applied on the reason side, after that on the conflict side and after the application of the cancellation rule, respectively. Observe that the percentage of irrelevant literals detected after the application of the weakening rule is high. Since this operation is applied exactly as specified in *RoundingSat*, this shows that this solver may introduce irrelevant literals as well.

Another interesting observation regarding our experimental results is that for the `vertexcover-instances` family (and more specifically, the `vertexcover-completegraph` family), the elimination of irrelevant literals has a significant impact on the size of the proof produced by the different configurations of *Sat4j*, as shown in Figures 5.17 and 5.18. The instances of this family encode that complete graphs do not have small vertex covers [EGNV18]. Figures 5.22 and 5.23 show more specifically the number of cancellations performed on this family.

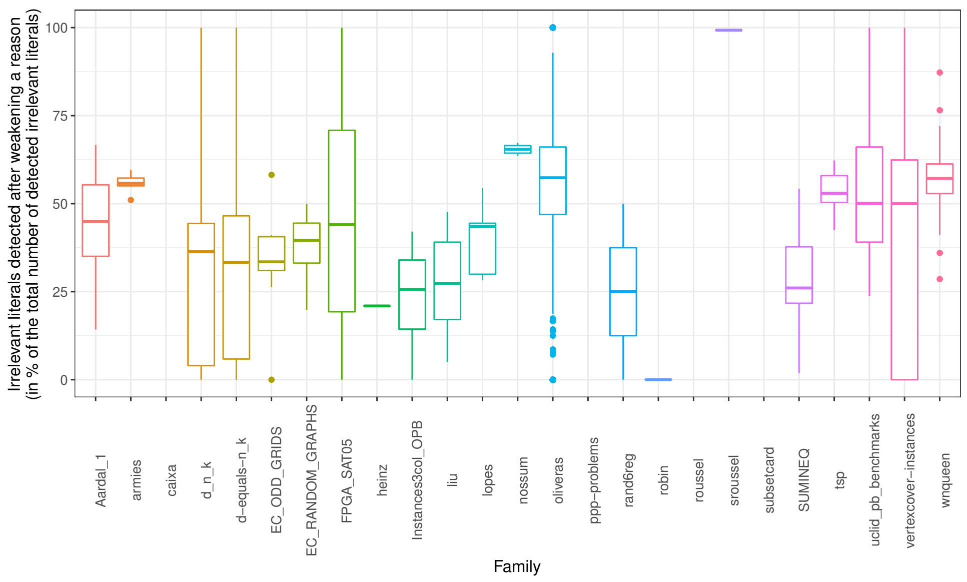


Figure 5.19: Boxplots of the percentage of irrelevant literals detected in *Sat4j-RoundingSat* after the application of the weakening rule on the reason side of the cancellation. Each family has its own box, for which the horizontal bars represent the quartiles and the vertical bars the estimated minimum and maximum. Points represent outliers, i.e., instances for which the percentage of the running time is either below or above the estimated minimum or maximum, respectively.

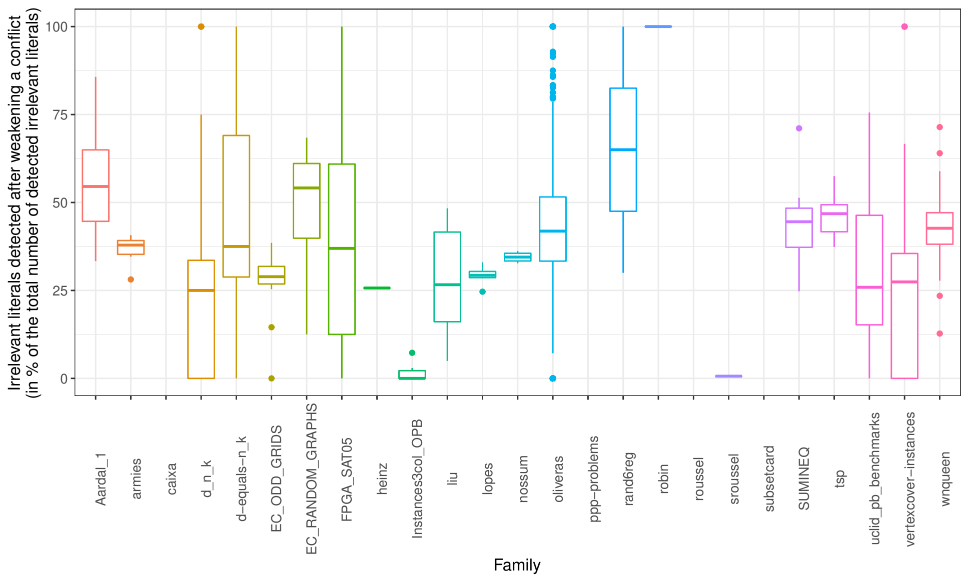


Figure 5.20: Boxplots of the percentage of irrelevant literals detected in *Sat4j-RoundingSat* after the application of the weakening rule on the conflict side of the cancellation. Each family has its own box, for which the horizontal bars represent the quartiles and the vertical bars the estimated minimum and maximum. Points represent outlier, i.e., instances for which the percentage of the running time is either below or above the estimated minimum or maximum, respectively.

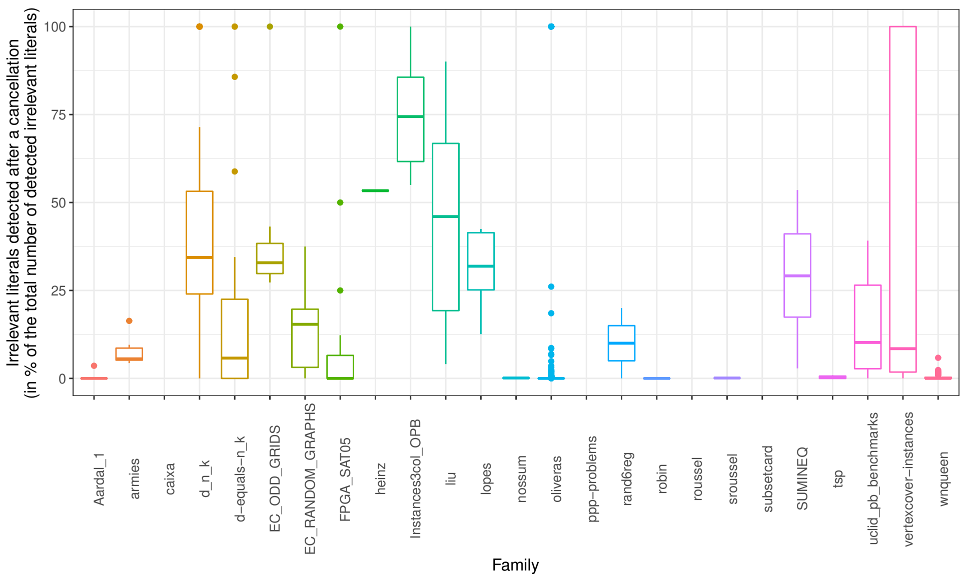


Figure 5.21: Boxplots of the percentage of irrelevant literals detected in *Sat4j-RoundingSat* after the application of the cancellation rule. Each family has its own box, for which the horizontal bars represent the quartiles and the vertical bars the estimated minimum and maximum. Points represent outlier, i.e., instances for which the percentage of the running time is either below or above the estimated minimum or maximum, respectively.

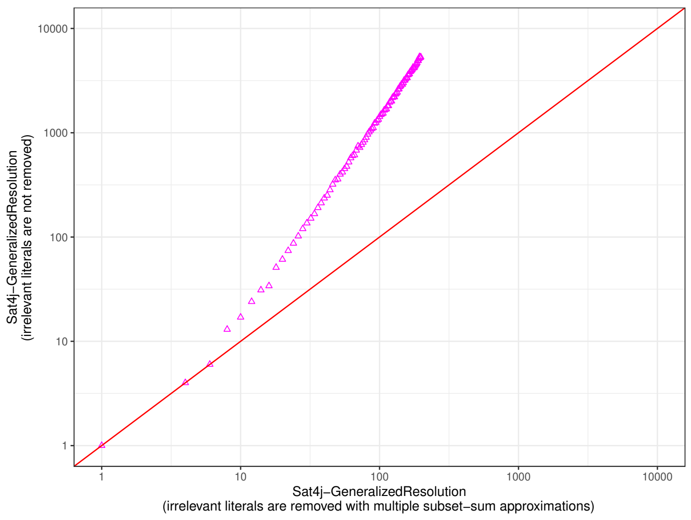


Figure 5.22: Scatter plot comparing the number of cancellations performed during the execution of *Sat4j-GeneralizedResolution* on the `vertexcover-completegraph` family with the removal of irrelevant literals activated (with the *ad hoc* algorithm) and without (logarithmic scale).

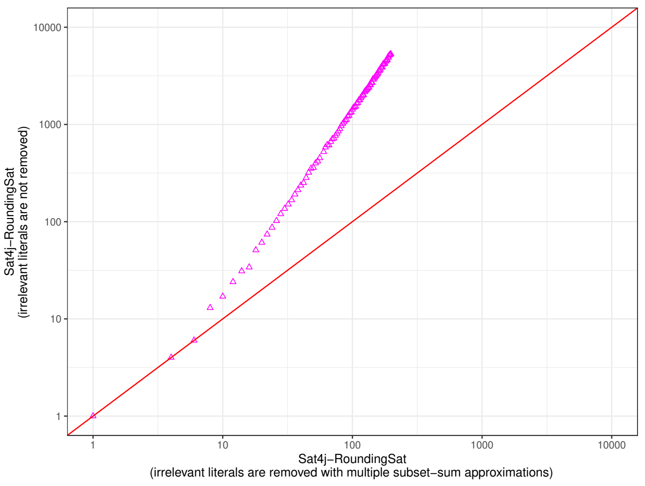


Figure 5.23: Scatter plot comparing the number of cancellations performed during the execution of *Sat4j-RoundingSat* on the `vertexcover-completegraph` family with the removal of irrelevant literals activated (with the *ad hoc* algorithm) and without (logarithmic scale).

The scatter plots show that the size of the proof built by the solver is exponentially smaller after removing irrelevant literals. A closer inspection of the behavior of the solver shows that only few irrelevant literals are actually removed during the search. In particular, all these literals are detected and removed after the first conflict analysis, which produces a constraint of the form $kx_1 + x_2 + ... + x_k \geq k$, where $k = \lceil \frac{n}{2} \rceil - 1$. One can observe that $x_2, ..., x_k$ are all irrelevant because their coefficients sum up to $k - 1$ only, and that the constraint is actually equivalent to the unit clause $x_1 \geq 1$. In all further conflict analyses, no irrelevant literals are produced: this illustrates how few irrelevant literals may have an impact on the whole proof built by the solver, and may degrade its performance.

In *RoundingSat*, a simplification procedure for decision level 0 allows, in this particular case, to infer exactly the unit clause $x_1 \geq 1$ that *Sat4j* infers only when eliminating irrelevant literals. This avoids the problematic behavior that we observe for *Sat4j*. However, it is possible to modify the `vertexcover-completegraph` to represent the problem of the 3-uniform complete hypergraphs, to create the `vertexcover-completehypergraph` family. These instances lead to learning constraints with irrelevant literals on a decision level higher than 0: more precisely the learned constraint has the form $kx_1 + kx_2 + ... + x_k + x_{k+1} \geq k$, and contain many irrelevant literals which the simplification procedure of *RoundingSat* does not eliminate since the constraint is assertive at decision level 1. Figures 5.24 and 5.25 show that a consistent improvement on the size of the proof is brought by the removal of irrelevant literals (even though the improvement is not exponential here).

The gain in the performance can also be observed on the ideal runtime of the solver for these instances, despite the bias we mentioned above regarding how the ideal runtime is computed. This is shown in Figures 5.26 and 5.27. Note that the ideal runtime is more relevant for the instances of this family rather than for the `vertexcover-completegraph` family, as the latter is much easier than the former (all its instances are solved in less than 4 seconds), and this is why we refrain from reporting the plot corresponding to this family.

These experiments show how the presence of irrelevant literals may have an impact on the performance of the solver. However, our approach for eliminating irrelevant literals is too costly in practice to be considered as a counter-measure to their production in current pseudo-Boolean solvers. We need to find other solutions to deal with irrelevant literals, for instance by taking advantage of the weakening rule.

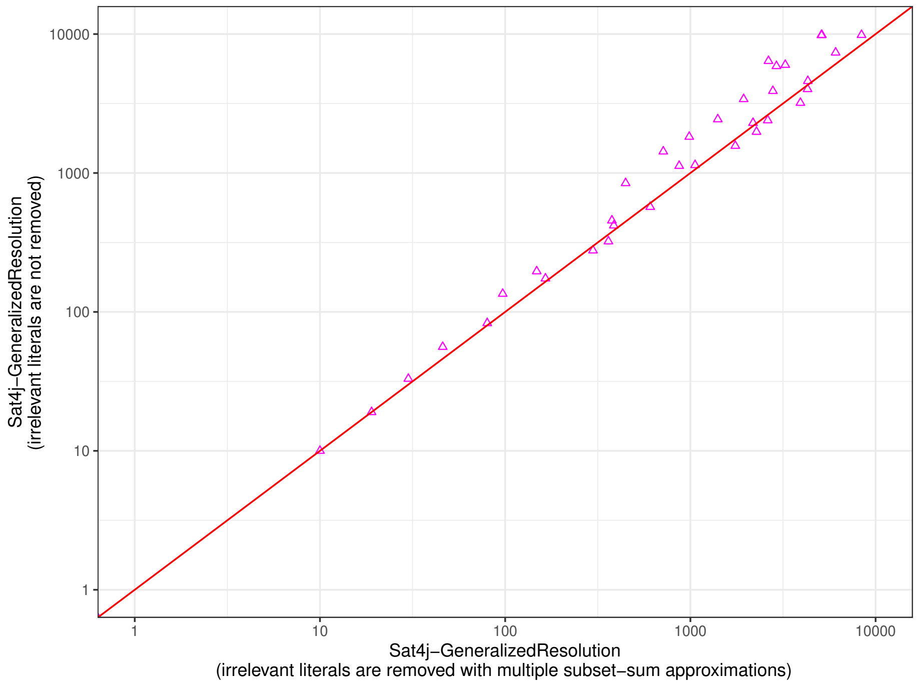


Figure 5.24: Scatter plot comparing the number of cancellations performed during the execution of *Sat4j-GeneralizedResolution* on the `vertexcover-completehypergraph` family with the removal of irrelevant literals activated (with the *ad hoc* algorithm) and without it (logarithmic scale).

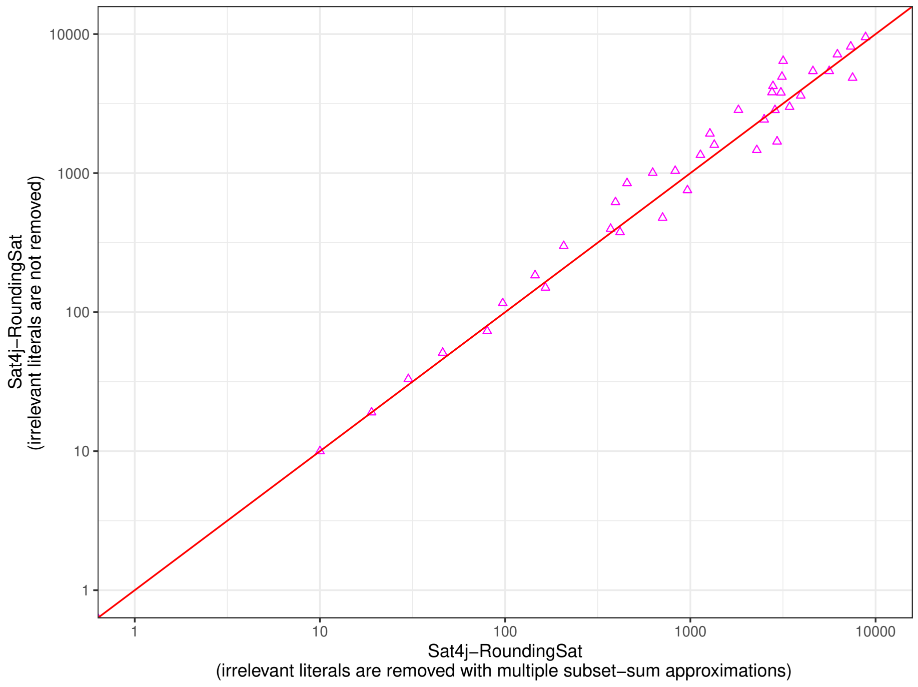


Figure 5.25: Scatter plot comparing the number of cancellations performed during the execution of *Sat4j-RoundingSat* on the `vertexcover-completehypergraph` family with the removal of irrelevant literals activated (with the *ad hoc* algorithm) and without it (logarithmic scale).

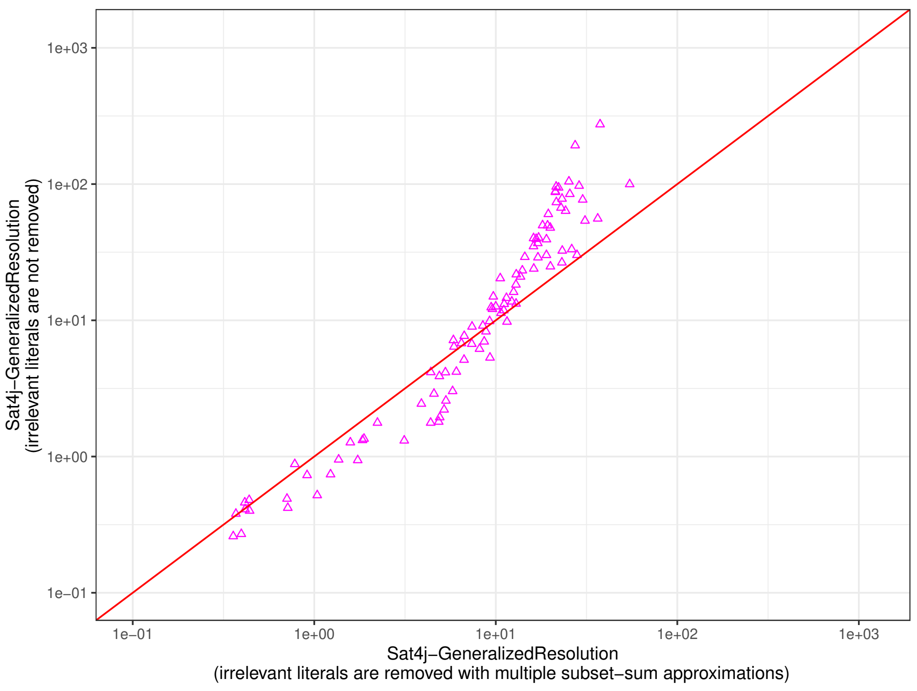


Figure 5.26: Scatter plot comparing the ideal runtime (in seconds) of *Sat4j-GeneralizedResolution* on the `vertexcover-completehypergraph` family with the removal of irrelevant literals activated (with the *ad hoc* algorithm) and without (logarithmic scale).

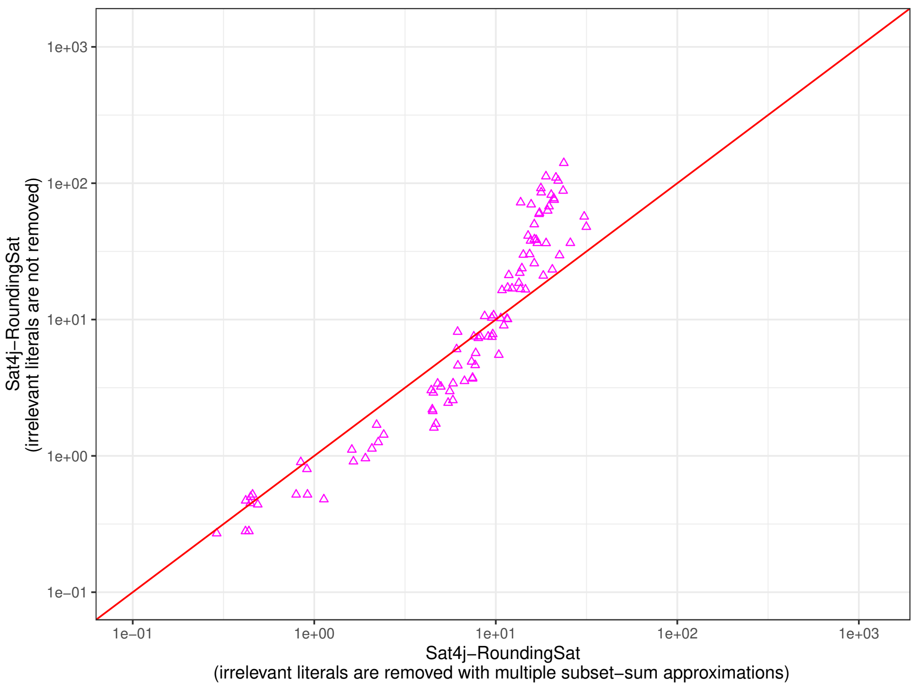


Figure 5.27: Scatter plot comparing the ideal runtime (in seconds) of *Sat4j-RoundingSat* on the `vertexcover-completehypergraph` family with the removal of irrelevant literals activated (with the *ad hoc* algorithm) and without (logarithmic scale).

Chapter 6

On Weakening Strategies for Pseudo-Boolean Solvers

As explained in Subsection 4.2.2, pseudo-Boolean solvers use the weakening rule to preserve conflicts during conflict analysis. Even though this forces the solver to weaken the constraints it derives, we show in this section that the performance of the solver can actually benefit from this rule, depending on how it is applied. In this context, we introduce different strategies for applying the weakening rule when Proposition 34 cannot be applied (and thus, when the conflict is potentially not preserved). Those strategies are designed towards reaching a tradeoff between the strength of the inferred constraints and their size [LMW20].

6.1 Weakening Ineffective Literals for Shorter Constraints

A first weakening strategy is to focus on *effective* literals in the constraints encountered during conflict analysis, and thus to weaken away all other literals from the constraint.

Definition 107 (Effective Literal)

Given a conflicting (resp. assertive) pseudo-Boolean constraint χ, a literal ℓ of χ is said to be *effective* in χ if it is falsified and satisfying it would not preserve the conflict (resp. propagation). We say that ℓ is *ineffective* when it is not effective.

Intuitively, ineffective literals are those that do not play a role in the propagation or in the conflict being considered. When all these ineffective literals are weakened away, the constraint is guaranteed to be a clause.

Proposition 39

Given a conflicting or assertive pseudo-Boolean constraint χ, the constraint derived after weakening away all ineffective literals in χ is equivalent to the disjunction of all the literals it contains, i.e., to a clause.

Proof. First, let χ be a conflicting pseudo-Boolean constraint $\sum_{i=1}^{n} \alpha_i \ell_i \geq \delta$, such that χ does not contain any ineffective literals (in particular, all literals in χ are falsified). Towards a contradiction, suppose that χ is not equivalent to a clause. There exists $i_0 \in \{1, \dots, n\}$ such that $\alpha_{i_0} < \delta$, i.e., $0 < \delta - \alpha_{i_0}$. Otherwise, observe that a saturation step can ensure that all coefficients would be equal to δ, and the constraint would be a clause. As a consequence, if χ is weakened on ℓ_{i_0} (i.e., if ℓ_{i_0} becomes now satisfied), we obtain the constraint $\sum_{i=1, i \neq i_0}^{n} \alpha_i \ell_i \geq \delta - \alpha_{i_0}$, which is clearly still conflicting as $\delta - \alpha_{i_0} > 0$, and all the literals ℓ_i are falsified. This contradicts the effectiveness of ℓ_{i_0}.

Let now χ be the assertive constraint $\alpha\ell + \sum_{i=1}^{n} \alpha_i \ell_i \geq \delta$, such that χ propagates ℓ, i.e., $\alpha > \mathsf{slack}(\chi)$. Suppose that all ℓ_i are effective and, towards a contradiction, suppose that χ is not equivalent to a clause. Observe that, as all ℓ_i are falsified (they are effective), $\mathsf{slack}(\chi) = \alpha - \delta$. As χ is not conflicting, we also have that $\mathsf{slack}(\chi) \geq 0$, so that $\alpha \geq \delta$. As χ is assumed to not be equivalent to a clause, there exists $i_0 \in \{1, \dots, n\}$ such that $\alpha_{i_0} < \delta$. Let us now satisfy ℓ_{i_0}. We now have $\mathsf{slack}(\chi) = \alpha + \alpha_{i_0} - \delta$. As $\alpha_{i_0} < \delta$, we have that $\mathsf{slack}(\chi) < \alpha + \delta - \delta$, i.e., $\mathsf{slack}(\chi) < \alpha$. The propagation of ℓ is thus preserved, which contradicts the effectiveness of ℓ_{i_0}.

$\square$

Proposition 39 illustrates that the weakening of ineffective literals is actually equivalent to the lazy clause inference implemented in solvers such as *SATIRE* [WS01] and *Sat4j-Resolution* [LP10] (see Section 4.3 for more details).

Example 80 (Example 65 cont'd)

Consider the constraint $5a(0@3) + 5b(?@?) + c(?@?) + d(?@?) + e(0@1) + f(1@2) \geq 6$. This constraint propagates b under the current partial assignment. This propagation still holds after weakening away c, d, e and f, giving after saturation $a(0@3) + b(?@?) \geq 1$.
After this propagation, the constraint $2a(0@3) + \bar{b}(0@3) + c(?@?) + e(0@1) \geq 2$ becomes conflicting. Observe that weakening the constraint on c and applying the saturation rule on this constraint produces $a(0@3) + \bar{b}(0@3) + e(0@1) \geq 1$, which is still conflicting.
The clause $a(0@3) + e(0@1) \geq 1$ is then obtained by applying the resolution rule between the two clauses.

Although this approach leads to the inference of weak constraints (as only clauses can be derived), it still has some advantages. In particular, ineffective literals can be seen as *locally* irrelevant, i.e., irrelevant under the current partial assignment, as shown in the example below.

Example 81 (Example 80 cont'd)

Consider again the constraint $5a(0@3) + 5b(?@?) + c(?@?) + d(?@?) + e(0@1) + f(1@2) \geq 6$. Under the current partial assignment, this constraint may be simplified into $5b(?@?) + c(?@?) + d(?@?) \geq 5$. In this constraint, c and d are irrelevant. However, since this constraint only exists under the current partial assignment, we cannot say that they are *globally* irrelevant, and this is why we consider them as *ineffective*.

Yet, ineffective literals may help to efficiently get rid of all irrelevant literals, as such literals are never effective.

Proposition 40

Given a conflicting or assertive pseudo-Boolean constraint χ, any literal ℓ that is irrelevant in χ is also ineffective.

Proof. We only have to prove the proposition in the case when ℓ is falsified (otherwise, it is obviously ineffective). If χ is conflicting, then by definition flipping the value of ℓ cannot satisfy the constraint, and thus preserves the conflict. If χ is assertive, let us suppose that it propagates a literal ℓ'. Equivalently, this means that χ becomes conflicting if ℓ' is falsified, and the same argument as before may be applied. □

This means that we can apply this weakening strategy to eliminate all irrelevant literals efficiently, at the price of also weakening away relevant literals. For instance, observe that, in Example 80, all ineffective literals that are weakened away are still relevant.

Remark 40

It is worth noting that, because we always apply the weakening rule when removing literals (irrelevant or not) here, the approach remains sound: the constraint that is eventually derived is indeed entailed by the original formula. As a downside, it is not possible to apply the *simple removal* of irrelevant literals when applying this strategy, as we do not know which of the ineffective literals are actually irrelevant.

Remark 41

Recall that, if a literal is irrelevant, then all literals with the same coefficient are also irrelevant by Proposition 35. We can thus conclude that, if a literal is ineffective but it is not possible to weaken away all literals with the same coefficient while preserving the conflict or the propagation, then this literal is not irrelevant. However, we do not exploit this property here, as we want the weakening strategy to ensure that a clause is derived to preserve the conflict after the application of the cancellation rule.

Another advantage of this strategy is that it allows to capture more precisely the reason for a conflict being encountered, which is the main purpose of conflict analysis in CDCL solvers. Indeed, as ineffective literals appearing in the constraints encountered during conflict analysis do not play a role in the conflict, they do not explain it either. They may thus be weakened away to get a tighter explanation of the conflict, as illustrated by the following example.

Example 82 (Example 81 cont'd)

Let us consider the constraint $5a(0@3)+5b(?@?)+c(?@?)+d(?@?)+e(0@1)+f(1@2) \geq 6$, which propagates b under the current partial assignment. This constraint is equivalent to the conjunction of the following clauses:

- $a(0@3) + b(?@?) \geq 1$
- $b(?@?) + c(?@?) + d(?@?) + e(0@1) + f(1@2) \geq 1$
- $a(0@3) + c(?@?) + d(?@?) + e(0@1) + f(1@2) \geq 1$

Clearly, the propagation of b is triggered by the clause $a(0@3) + b(?@?) \geq 1$, which is precisely the clause we obtain after weakening away ineffective literals.
Similarly, the conflicting constraint $2a(0@3) + \bar{b}(0@3) + c(?@?) + e(0@1) \geq 2$ is equivalent to the conjunction of the clauses:

- $a(0@3) + \bar{b}(0@3) + c(?@?) \geq 1$
- $a(0@3) + \bar{b}(0@3) + e(0@1) \geq 1$
- $a(0@3) + c(?@?) + e(0@1) \geq 1$

Once again, the clause that is falsified here is exactly $a(0@3) + \bar{b}(0@3) + e(0@1) \geq 1$, which is also the clause we obtain after weakening away ineffective literals.

However, as explained before, weakening away ineffective literals always yields a clause. On the one hand, this means that the coefficient of the pivot on which the cancellation rule is applied during conflict analysis will always be 1, which ensures that the conflict will be preserved (see Proposition 34). On the other hand, this also means that, if we apply this strategy on both sides of the cancellation rule, the proof system boils down to the weaker resolution proof system. In particular, multiple clauses represented by the pseudo-Boolean constraint being considered may be conflicting at the same time, while this approach can only identify one of them.

To preserve the strength of the proof system used by the solver, a possible solution is to weaken away ineffective literals from only *one* side of the cancellation rule (which still guarantees by Proposition 34 that the conflict will be preserved). This is illustrated by the following example.

Example 83 (Example 82 cont'd)

Let us consider again the constraint $5a(0@3) + 5b(?@?) + c(?@?) + d(?@?) + e(0@1) + f(1@2) \geq 6$, which propagates b under the current partial assignment. The constraint $2a(0@3) + \bar{b}(0@3) + c(?@?) + e(0@1) \geq 2$ is now conflicting. As shown in Example 82, the weakening of ineffective literals from these two constraints yields the clauses $a(0@3) + b(1@3) \geq 1$ and $a(0@3) + \bar{b}(0@3) + e(0@1) \geq 1$ respectively.
Let us suppose that ineffective literals are only weakened away from the reason side. The cancellation between $a(0@3) + b(?@?) \geq 1$ and $2a(0@3) + \bar{b}(0@3) + c(?@?) + e(0@1) \geq 2$ produces, after saturation, the constraint $2a(0@3) + c(?@?) + e(0@1) \geq 2$.
Note that this latter constraint is stronger than the clause $a(0@3) + e(0@1) \geq 1$ derived by applying the resolution between the two clauses derived from the original pseudo-Boolean constraints.

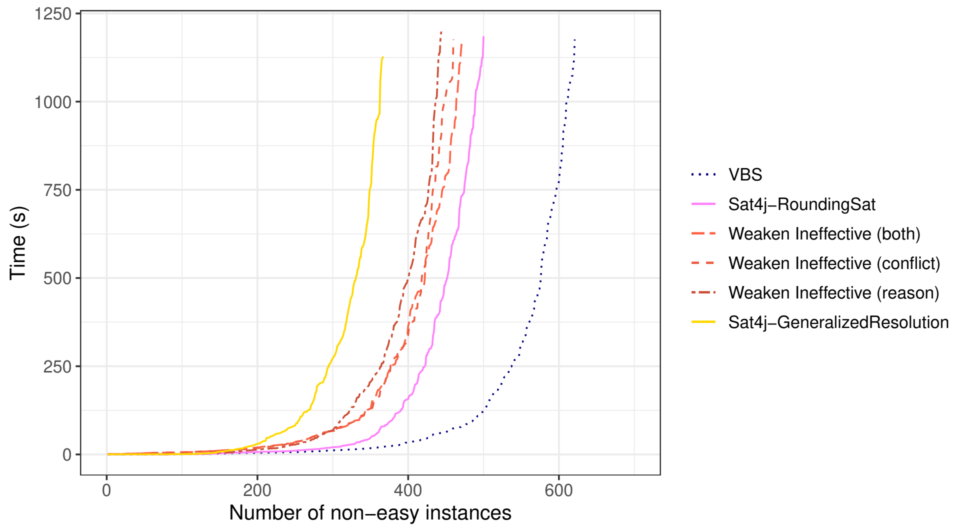


Figure 6.1: Cactus plot of the different *Weaken Ineffective* strategies implemented in *Sat4j*, compared to *Sat4j*'s state-of-the-art pseudo-Boolean solvers.

Let us now empirically evaluate the performance of the different variants of the weakening of ineffective literals, i.e., when it is applied on either the conflict side (*Weaken Ineffective (conflict)*), the reason side (*Weaken Ineffective (reason)*) or both sides (*Weaken Ineffective (both)*) of the cancellation. These variants have been implemented in the pseudo-Boolean solver *Sat4j* (see Appendix A), and executed in the usual experimental setting (see Appendix B). The timeout was set to 1200 seconds and the memory limit to 32 GB. The results are given in Figure 6.1.

An interesting observation from the cactus plot is that performing the weakening of ineffective literals on the conflict side has better performance than applying it on the reason side. This is quite surprising, because, as presented in Subsection 4.2.2, weakening operations in solvers based on cutting planes are mainly performed on the *reason* side (except for *RoundingSat* [EN18], which applies it on *both* sides). Our experiments show that it may be preferable to apply it only on the *conflict* side: literals introduced there when cancelling may still be weakened away during a later weakening operation. Another key observation is that the VBS is far better than each individual strategy, which suggests that none of them is better than the others on all benchmarks.

In order to explain the performance of this VBS, let us make a pairwise comparison of the three variants through the scatter plots given in Figure 6.3. A first observation is that there is no clear difference between *Weaken Ineffective (both)* and *Weaken Ineffective (conflict)*. On the contrary, there are some interesting differences between these two variants and *Weaken Ineffective (reason)*. In particular, this latter strategy seems very efficient at solving instances from the families `FPGA_SAT05`, `rand6reg` and `sroussel` while it exhibits poor performance on the family `wnqueen`.

Yet, the weakening of ineffective literals on both sides of the cancellation provides the best performance overall. Despite being quite similar, by construction, to *Sat4j-Resolution*, its performance remains however very different from the one of this solver, as shown in Figure 6.2. From this latter scatter plot, we can make two main observations. First, many instances are solved faster by *Sat4j-Resolution* compared to the *Weaken Ineffective (both)* strategy. Indeed, when using the former, the conflict analysis procedure is optimized for resolution-based reasoning, while the latter is integrated in the conflict analysis based on cutting planes, and is not “aware” that clauses are inferred in many circumstances. This is however inevitable in this case, as the weakening operation is applied here *only when Proposition 34 cannot be applied.*

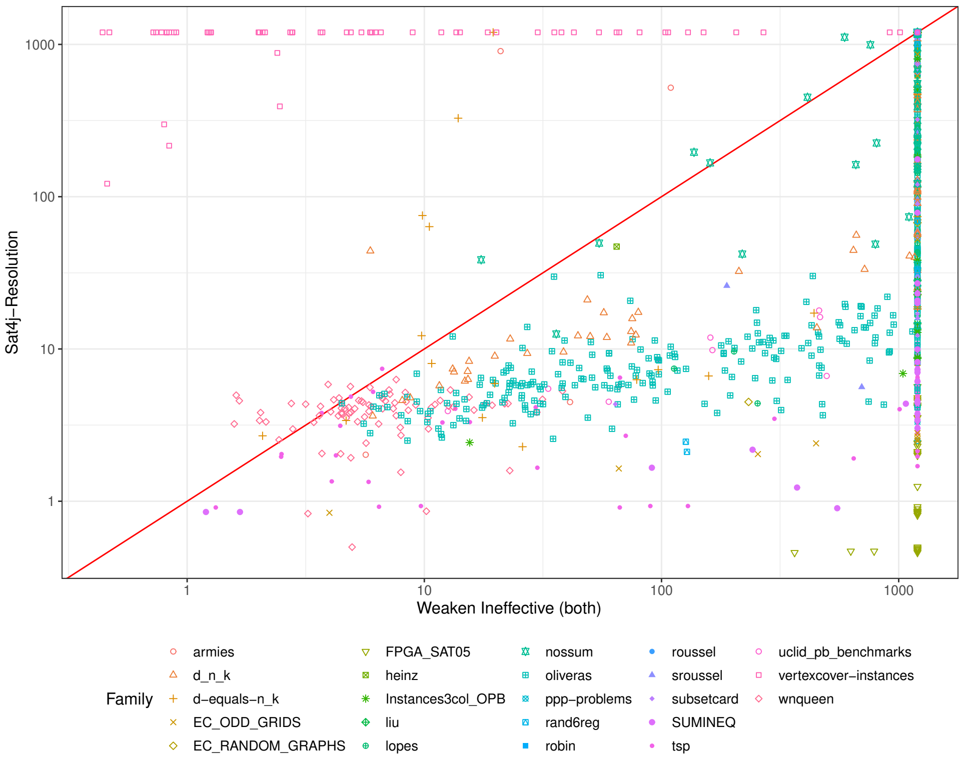


Figure 6.2: Scatter plot comparing the runtime (in seconds) of *Weaken Ineffective (both)* and *Sat4j-Resolution* (logarithmic scale).

This means that, in some cases, the solver is still able to derive pseudo-Boolean constraints, which leads us to the second main observation: on some families, especially `vertexcover-instances`, the *Weaken Ineffective (both)* strategy is far better than *Sat4j-Resolution*. For these instances, the solver can apply Proposition 34: the weakening of ineffective literals *is not always used*, which allows to infer pseudo-Boolean constraints, which seem to be crucial for solving efficiently these problems in *Sat4j*.

Let us now consider the number of instances solved by the different strategies based on the results in Table 6.1. We can see that both *Weaken Ineffective (reason)* and *Weaken Ineffective (both)* are far better than the other solvers for different families, such as `d_n_k`, `oliveras` or `tsp`. It may also be worse than all the other solvers, for instance for the family `FPGA_SAT05`, `roussel` or `subsetcard`. This may be explained by the fact that, when the solvers use these two strategies, the constraints to be learned will most likely be clauses, so that the power of the cutting planes proof system is not exploited on families that require this strength. On the contrary, when this strength is not required, learning clauses allows to be more efficient when detecting propagations and analyzing conflicts.

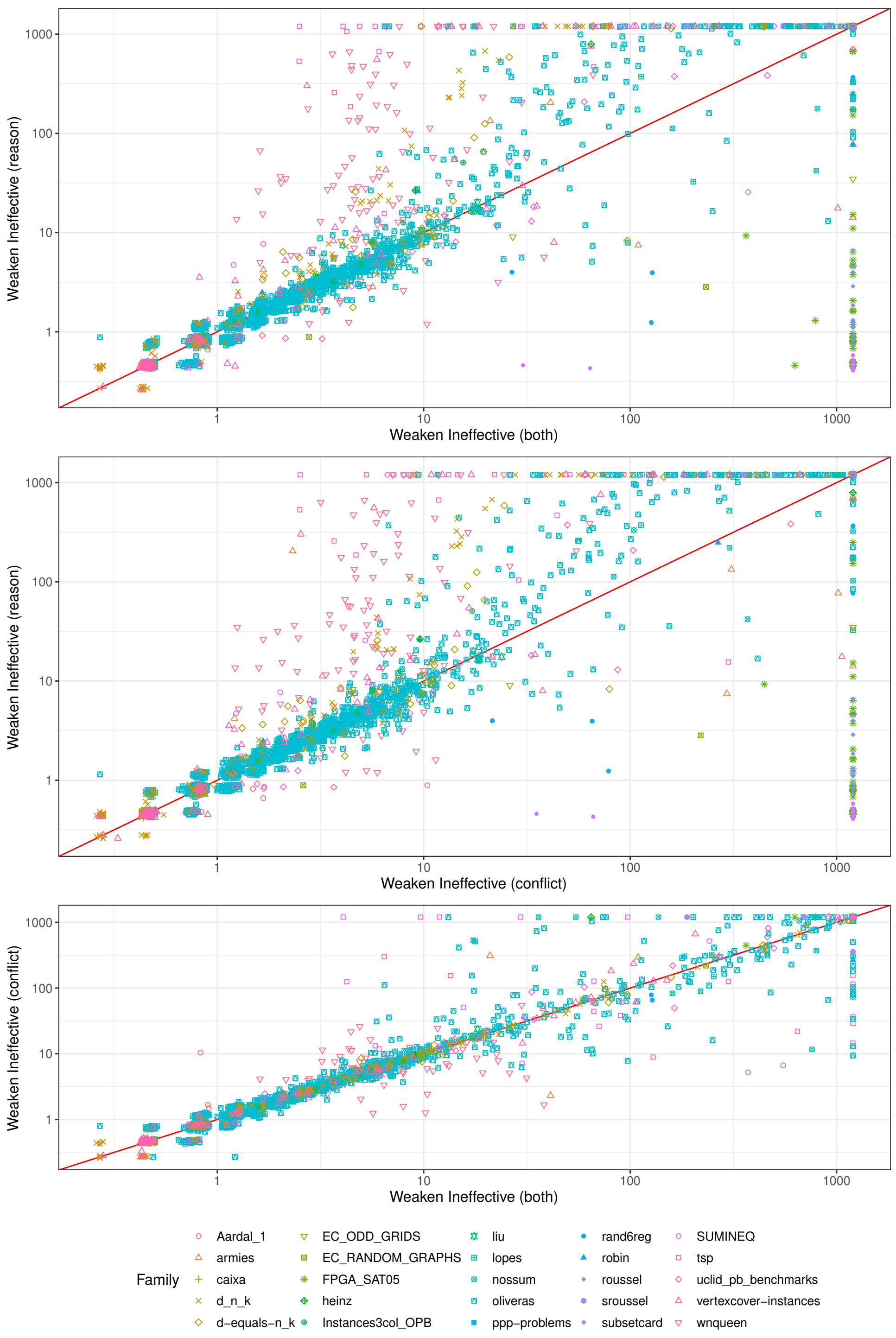


Figure 6.3: Scatter plots comparing the runtime (in seconds) of the different *Weaken Ineffective* variants (logarithmic scale).

Family	Number of instances in the family	Number of solved easy instances	*Sat4j GeneralizedResolution*	*Weaken-Ineffective* (reason)	*Weaken-Ineffective* (conflict)	*Weaken-Ineffective* (both)	*Sat4j RoundingSat*
Aardal_1	14	14	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)
armies	12	1	3 (765.84)	6 (440.06)	5 (1627.47)	4 (177.16)	4 (74.6)
caixa	1	1	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)
d_n_k	234	155	14 (4318.16)	13 (3204.97)	**32 (5861.61)**	**32 (5843.09)**	18 (6167.55)
d-equals-n_k	70	28	12 (489.46)	12 (2104.55)	15 (877.21)	15 (917.48)	14 (3221.75)
EC_ODD_GRIDS	25	2	2 (65.3)	2 (42.81)	5 (1956.68)	4 (773.38)	5 (656.94)
EC_RANDOM_GRAPHS	22	3	2 (237.45)	1 (2.83)	1 (219.49)	1 (232.85)	5 (906.13)
FPGA_SAT05	57	0	**43 (1835.72)**	**43 (2001.91)**	1 (446.22)	3 (1779.76)	**41 (2279.38)**
heinz	4	1	0 (0)	1 (790.59)	0 (0)	1 (64.69)	0 (0)
Instances3col_OPB	26	7	1 (51.69)	1 (50.77)	2 (1029.34)	2 (1057.77)	1 (20.23)
liu	20	16	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)
lopes	193	0	1 (256.01)	2 (405.98)	2 (262.88)	3 (569.3)	2 (1210.13)
nossum	180	0	2 (1567.69)	**8 (2428.21)**	**11 (3862.85)**	**13 (5756.47)**	**10 (3378.61)**
oliveras	4080	3017	121 (27936.74)	135 (32019.35)	**198 (38731.39)**	**204 (40946.32)**	151 (35341.14)
ppp-problems	6	0	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)
rand6reg	33	4	3 (208.84)	4 (448.23)	2 (144.05)	2 (254.64)	7 (20.47)
robin	6	2	0 (0)	1 (248.51)	1 (266.42)	0 (0)	0 (0)
roussel	40	0	**20 (31.28)**	**20 (32.06)**	0 (0)	0 (0)	**20 (30.46)**
sroussel	122	0	4 (853.18)	0 (0)	1 (355.29)	2 (887.12)	2 (326.76)
subsetcard	56	1	**55 (23.81)**	**55 (25.4)**	1 (66.21)	1 (63.89)	**55 (24.42)**
SUMINEQ	24	0	1 (5.72)	3 (38.06)	6 (593.72)	7 (2331.13)	3 (847.47)
tsp	100	0	14 (5756.76)	9 (3301.4)	**28 (1613.72)**	**27 (2636.63)**	14 (2151.31)
uclid_pb_benchmarks	50	29	5 (824.86)	6 (1699.25)	8 (2291.5)	8 (1855.23)	10 (2252.06)
vertexcover-instances	107	46	42 (1806.68)	47 (3132.34)	59 (3267.71)	60 (3485.92)	56 (3292.67)
wnqueen	100	18	22 (4923.28)	75 (8579.31)	**82 (674.24)**	**82 (724.09)**	**82 (780.94)**

Table 6.1: Table summarizing, for each variant of *Weaken-Ineffective* and *Sat4j*'s state-of-the-art solvers, the number of non-easy instances solved by this approach for each of the considered families. The numbers in parentheses correspond to the total runtime (in seconds) needed to solve the instances. Bold numbers highlight solvers that perform well on a given family compared to the others.

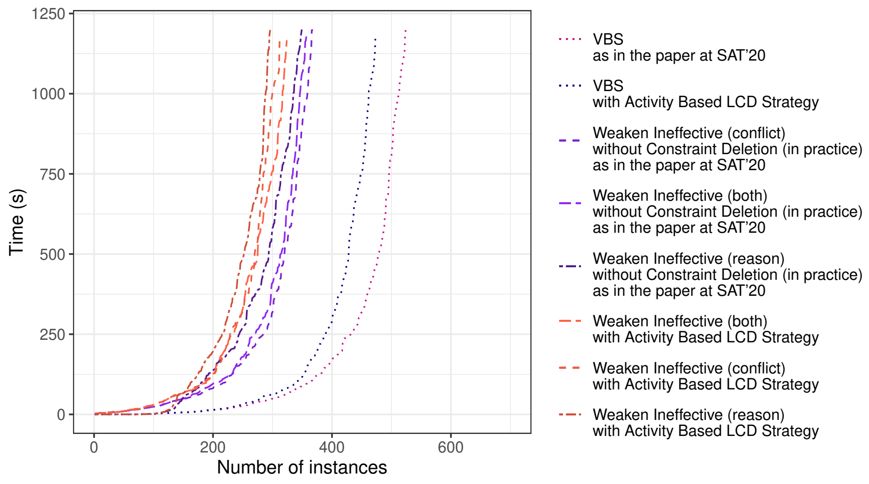


Figure 6.4: Cactus plot comparing the current implementations of the *Weaken Ineffective* with those described in [LMW20].

Note that the experimental results presented here are slightly different from those presented in [LMW20], especially regarding the results of the *Weaken Ineffective (both)* strategy: as mentioned above, this strategy now has better performance than the others, whereas, as illustrated by Figure 6.4, the best strategy was *Weaken Ineffective (conflict)* in [LMW20].

The discrepancy between the results is due to a change in the default configuration of *Sat4j* that occurred between the publication of [LMW20] and the writing of this thesis. In particular, an incorrect behavior of the learned constraint deletion strategy used by default in *Sat4j* preventing it from actually deleting constraints has been fixed, and the default strategy is now based on the activity of the learned constraints to decide which constraints should be deleted. In practice, however, this strategy does not have good performance, as further studied in Subsection 7.2.2. For the consistency of this thesis, we have chosen to use the same configuration of *Sat4j* in all the experiments and thus have rerun these experiments with the new default strategy, which has an impact on the performance of the solver. For the same reason, the experimental results presented in the following may also be slightly different from those from [LMW20].

Yet, similar conclusions may be drawn from both experiments. In particular, the *Weaken Ineffective (conflict)* strategy remains better than the *Weaken Ineffective (reason)* strategy, and more importantly, all three strategies are better than the classical generalized resolution-based approach. In both cases, we can also see that the VBS is much faster than each individual strategy, which does not seem to have robust performance. In particular, the *RoundingSat*-based approach is still faster than these strategies, and this is why we now consider some variants of the proof system used by this solver.

6.2 Stronger Constraints in *RoundingSat*-Based Solvers

Despite being more efficient in practice, *RoundingSat* has a major downside: the constraints it infers are in general weaker than that derived by generalized resolution-based solvers (see Examples 60 and 62). Even though this allows to keep coefficients small, this also means that more constraints may be required to eliminate a subpart of the search space. In order to improve the strength of the constraints inferred by solvers based on *RoundingSat*'s algorithm, let us consider different variants of the proof system implemented by this solver.

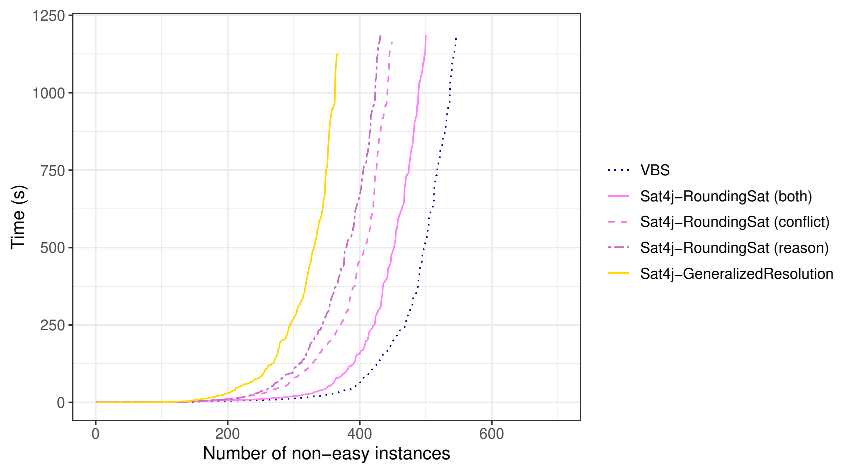


Figure 6.5: Cactus plot of the different *RoundingSat* variants implemented in *Sat4j*, compared to *Sat4j*'s state-of-the-art pseudo-Boolean solvers.

A first variant to consider in this direction is applying the weakening rule as proposed in the original implementation of *RoundingSat*, but only on *one side* of the cancellation rule when Proposition 34 cannot be applied. Doing so allows to restore the conflict when it may be lost, while it may enable the inference of stronger constraints, as illustrated by the following example.

Example 84

Let us consider the constraint $4c(0@2)+2\bar{b}(?@?)+2\bar{d}(?@?)+a(1@1) \geq 4$, which propagates both $\bar{b}$ and $\bar{d}$ under the current partial assignment. After these propagations, the constraint $8a(1@1) + 7b(0@2) + 7c(0@2) + 2d(0@2) + 2e(0@1) + f(?@?) \geq 11$ becomes conflicting. If the weakening and division operations are only performed on the reason side, giving the constraint $2c(0@2) + \bar{b}(0@2) + \bar{d}(0@2) \geq 2$, the cancellation rule applied on b between the weakened reason and the original conflict produces the constraint $21c(0@2) + 8a(1@1) + 5\bar{d}(0@2)+2e(0@1)+f(?@?) \geq 16$, which is stronger than the clause derived by the classical implementation of *RoundingSat*, i.e., $c(0@2) + e(0@1) \geq 1$.

Let us now empirically evaluate variants of *RoundingSat* in *Sat4j*, called *RoundingSat (conflict)* and *RoundingSat (reason)*. These strategies have been implemented in the solver *Sat4j* (see Appendix A), and executed in the usual experimental setting (see Appendix B). The timeout was set to 1200 seconds and the memory limit to 32 GB. The results are given in Figure 6.5.

The cactus plot shows that the two new variants of *RoundingSat* are not as good as the default *RoundingSat*. However, as illustrated by the VBS of the different *RoundingSat* variants (including the original approach), there is still room for improvements, and finding a tradeoff between the three approaches would help improve the performance of solvers based on *RoundingSat*.

Another interesting observation is that, as for the *Weaken Ineffective* strategies, the application of the weakening operation on the conflict side leads to better performance than its application on the reason side, which tends to confirm the observation we made before.

Family	Number of instances in the family	Number of solved easy instances	*Sat4j GeneralizedResolution*	*Sat4j-RoundingSat* (reason)	*Sat4j-RoundingSat* (conflict)	*Sat4j-RoundingSat* (both)
`Aardal_1`	14	14	0 (0)	0 (0)	0 (0)	0 (0)
`armies`	12	1	3 (765.84)	4 (1034.93)	**6 (333.74)**	4 (74.6)
`caixa`	1	1	0 (0)	0 (0)	0 (0)	0 (0)
`d_n_k`	234	155	14 (4318.16)	15 (5899.68)	14 (4632.95)	18 (6167.55)
`d-equals-n_k`	70	28	12 (489.46)	10 (787.94)	13 (1752.47)	14 (3221.75)
`EC_ODD_GRIDS`	25	2	2 (65.3)	4 (772.12)	3 (758.66)	5 (656.94)
`EC_RANDOM_GRAPHS`	22	3	2 (237.45)	6 (350.86)	3 (217.36)	5 (906.13)
`FPGA_SAT05`	57	0	43 (1835.72)	42 (1018.41)	45 (1423.39)	41 (2279.38)
`heinz`	4	1	0 (0)	0 (0)	1 (954.06)	0 (0)
`Instances3col_OPB`	26	7	1 (51.69)	1 (91.71)	1 (47.69)	1 (20.23)
`liu`	20	16	0 (0)	0 (0)	0 (0)	0 (0)
`lopes`	193	0	1 (256.01)	2 (671.58)	1 (462.66)	2 (1210.13)
`nossum`	180	0	2 (1567.69)	4 (1036.19)	5 (2370.59)	**10 (3378.61)**
`oliveras`	4080	3017	121 (27936.74)	115 (34274.58)	127 (29039.25)	**151 (35341.14)**
`ppp-problems`	6	0	0 (0)	0 (0)	0 (0)	0 (0)
`rand6reg`	33	4	3 (208.84)	**9 (1170.23)**	**8 (44.64)**	**7 (20.47)**
`robin`	6	2	0 (0)	1 (29.35)	1 (230.79)	0 (0)
`roussel`	40	0	20 (31.28)	20 (30.69)	20 (31.14)	20 (30.46)
`sroussel`	122	0	4 (853.18)	0 (0)	3 (2058.76)	2 (326.76)
`subsetcard`	56	1	55 (23.81)	55 (25.51)	55 (24.78)	55 (24.42)
`SUMINEQ`	24	0	1 (5.72)	2 (1049.72)	4 (301.37)	3 (847.47)
`tsp`	100	0	14 (5756.76)	17 (4138.05)	12 (2254.83)	14 (2151.31)
`uclid_pb_benchmarks`	50	29	5 (824.86)	5 (1787.53)	5 (1576.51)	10 (2252.06)
`vertexcover-instances`	107	46	42 (1806.68)	46 (2205.3)	46 (4169.74)	56 (3292.67)
`wnqueen`	100	18	22 (4923.28)	75 (11569.51)	76 (9049.53)	**82 (780.94)**

Table 6.2: Table summarizing, for each variant of *Sat4j-RoundingSat* and *Sat4j*'s state-of-the-art solvers, the number of non-easy instances solved by this approach for each of the considered families. The numbers in parentheses correspond to the total runtime (in seconds) needed to solve the instances. Bold numbers highlight solvers that perform well on a given family compared to the others.

Let us now consider the number of instances solved by the different *RoundingSat*-based strategies as given in Table 6.2. This table coheres with the observations made on the cactus plot above, in the sense that, in general, all *RoundingSat*-based approaches solve more instances than *Sat4j-GeneralizedResolution*, and this is particularly the case for the *Sat4j-RoundingSat (both)* strategy.

To improve *RoundingSat* performance, another variant of its proof system that may be considered is the application of the *partial weakening* rule, instead of the weakening rule. Let us described this approach, which was already mentioned in [EN18, Remark 3.4]. Before cancelling a literal out during conflict analysis, all literals that are not currently falsified and have coefficients not divisible by the weight of the pivot are *partially weakened* (instead of simply *weakened*). This operation is applied so that the resulting coefficient becomes a multiple of the weight of the pivot. Algorithm 13 below describes this procedure.

Algorithm 13: partialRoundingSatReduce

```
Input  : A pseudo-Boolean constraint $\chi$ and a literal $\ell$ of $\chi$
Output: The constraint $\chi$ after reduction
1  $\alpha \leftarrow \mathsf{coefficient}(\ell, \chi)$
2  foreach literal $\ell'$ in $\chi$ do
3      $\alpha' \leftarrow \mathsf{coefficient}(\ell', \chi)$
4      if $\ell'$ is not currently falsified then
5          if $\alpha'$ is not divisible by $\alpha$ then
6              // The literal $\ell'$ is partially weakened
7              $\mu \leftarrow \alpha' \mod \alpha$
8              $\mathsf{degree}(\chi) \leftarrow \mathsf{degree}(\chi) - \mu$
9              $\alpha' \leftarrow \alpha' - \mu$
10         end
11     end
12     // Dividing the coefficient: $\alpha'$ is always divisible by $\alpha$ here
13     $\mathsf{coefficient}(\ell', \chi) \leftarrow \alpha'/\alpha$
14 end
```

This approach has several advantages. First, it preserves the nice properties of *RoundingSat*, and in particular the fact that the constraint after the cancellation step will be conflictual (the coefficient of the pivot will be 1), while having a cost comparable to that of *RoundingSat*: checking whether a coefficient is divisible by the weight of the pivot is computed with the remainder of the division of the former by the latter, which is the amount by which the literal must be partially weakened. Second, the constraints it infers may be stronger than that of *RoundingSat*, as illustrated by the following example.

Example 85 (Example 84 cont'd)

Consider again the (conflicting) constraint $8a(1@1) + 7b(0@2) + 7c(0@2) + 2d(0@2) + 2e(0@1) + f(?@?) \geq 11$ where b is the literal to be cancelled out. The above rule yields $7a(1@1)+7b(0@2)+7c(0@2)+2d(0@2)+2e(0@1) \geq 9$ which, divided by 7, gives $a+b+c+d+e \geq 2$. This constraint is stronger than the clause $b(0@2)+c(0@2)+d(0@2)+e(0@1) \geq 1$ that *RoundingSat* would infer, as the literal a is completely weakened away in this case.

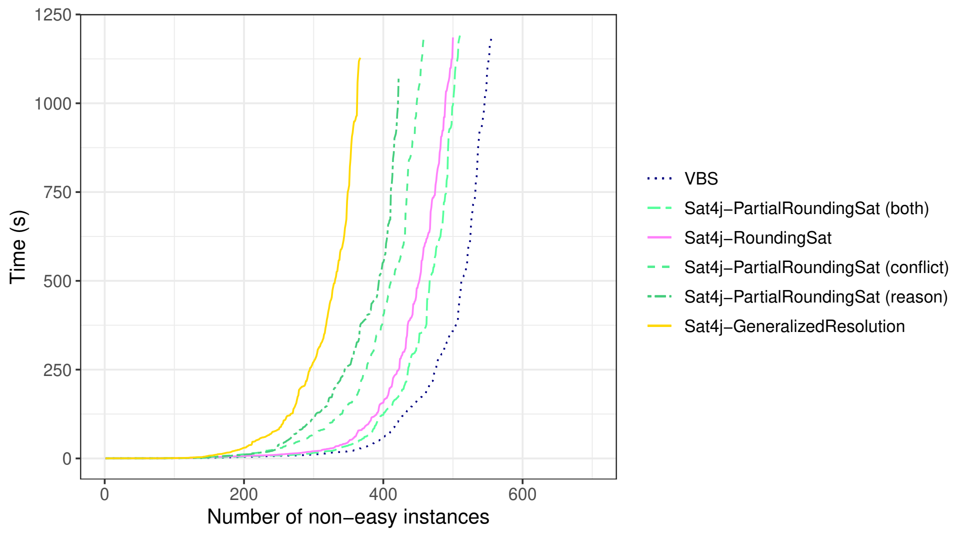


Figure 6.6: Cactus plot of the different *Partial RoundingSat* variants implemented in *Sat4j*, compared to *Sat4j*'s state-of-the-art pseudo-Boolean solvers.

Let us now evaluate the performance of this approach, named *PartialRoundingSat*, and of its variants obtained by applying the weakening rule on only one side of the cancellation rule. To do so, we compare the implementation of these strategies with *Sat4j*'s implementation of *RoundingSat*. Figure 6.6 shows the results of their implementation in *Sat4j* and executed in the usual experimental setting (see Appendix B). The timeout was set to 1200 seconds and the memory limit to 32 GB.

The relative order of the different *Partial RoundingSat* variants is quite similar to that of *RoundingSat* variants. The main observation here is that applying the weakening and division rules on both sides of the cancellation rule improves the performance of the solver compared to that of *RoundingSat*.

Remark 42

In [EN18, Remark 3.4], it is observed that the performance of *RoundingSat* gets worse when the partial weakening rule is applied instead of the weakening rule, which is not what we observe in our implementation in *Sat4j*.
However, there are actually many implementation details that differ between *RoundingSat* and its implementation in *Sat4j*, especially because the former applies the weakening rule unconditionally, while the latter applies it only when Proposition 34 cannot be applied (see also Appendix A for a complete overview of the difference of implementations between the two solvers). All these differences may also have an impact on the performance of both solvers, and thus may explain why our observations do not coincide.

Moreover, despite being more robust than the *Weaken Ineffective* strategies, we still observe that the VBS of the different *RoundingSat*-based approaches is better than each individual strategy.

Let us now consider more precise results by looking at the scatter plots given in Figures 6.7 and 6.8. From these figures, we can see that the applications of the weakening and division rules on either the conflict or the reason side are mostly incomparable. Interestingly, applying the same rules on both sides is clearly better than these approaches, and this is particularly clear for the instances of the `wnqueen` family.

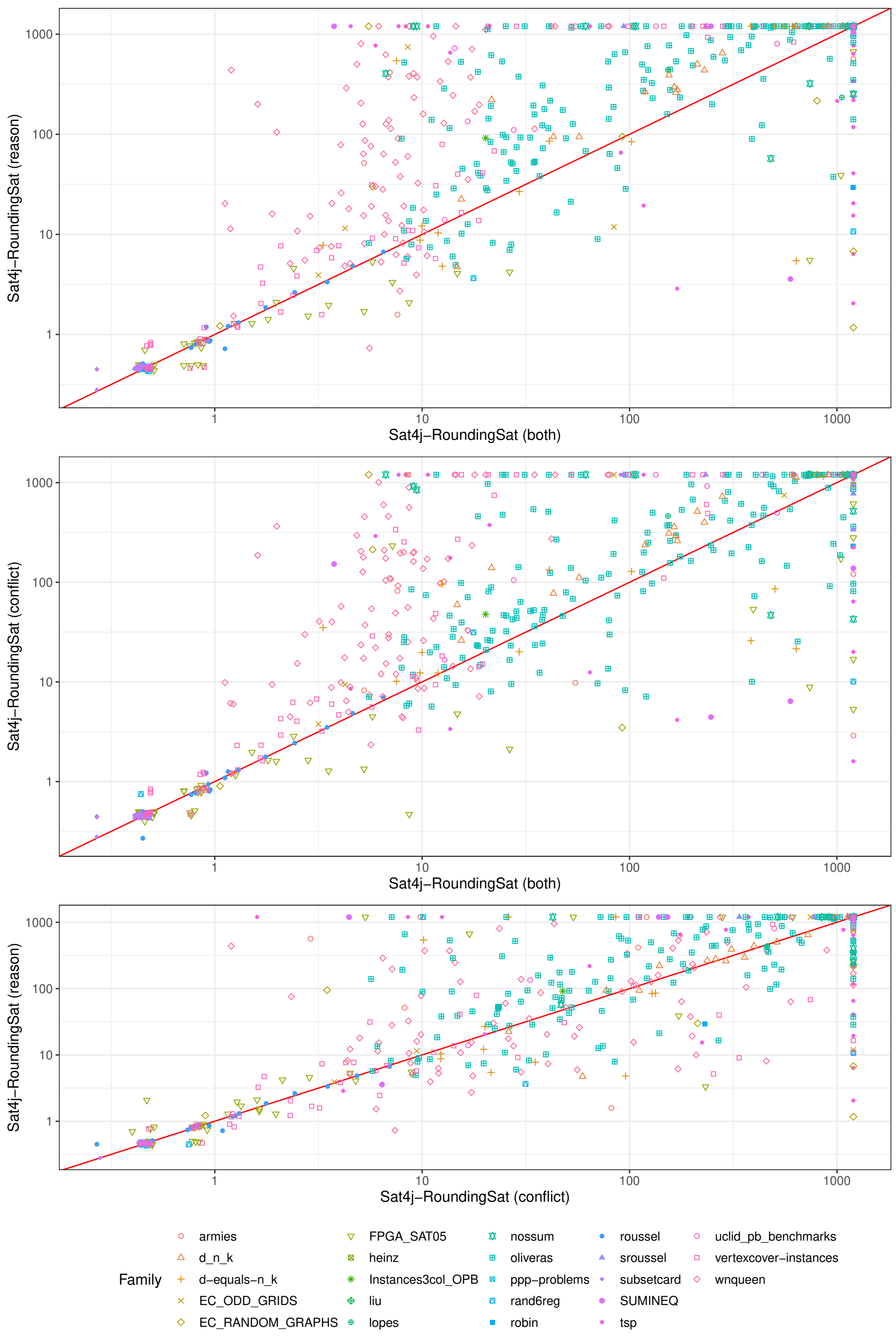


Figure 6.7: Scatter plots comparing the runtime (in seconds) of the different *RoundingSat* variants in *Sat4j* (logarithmic scale).

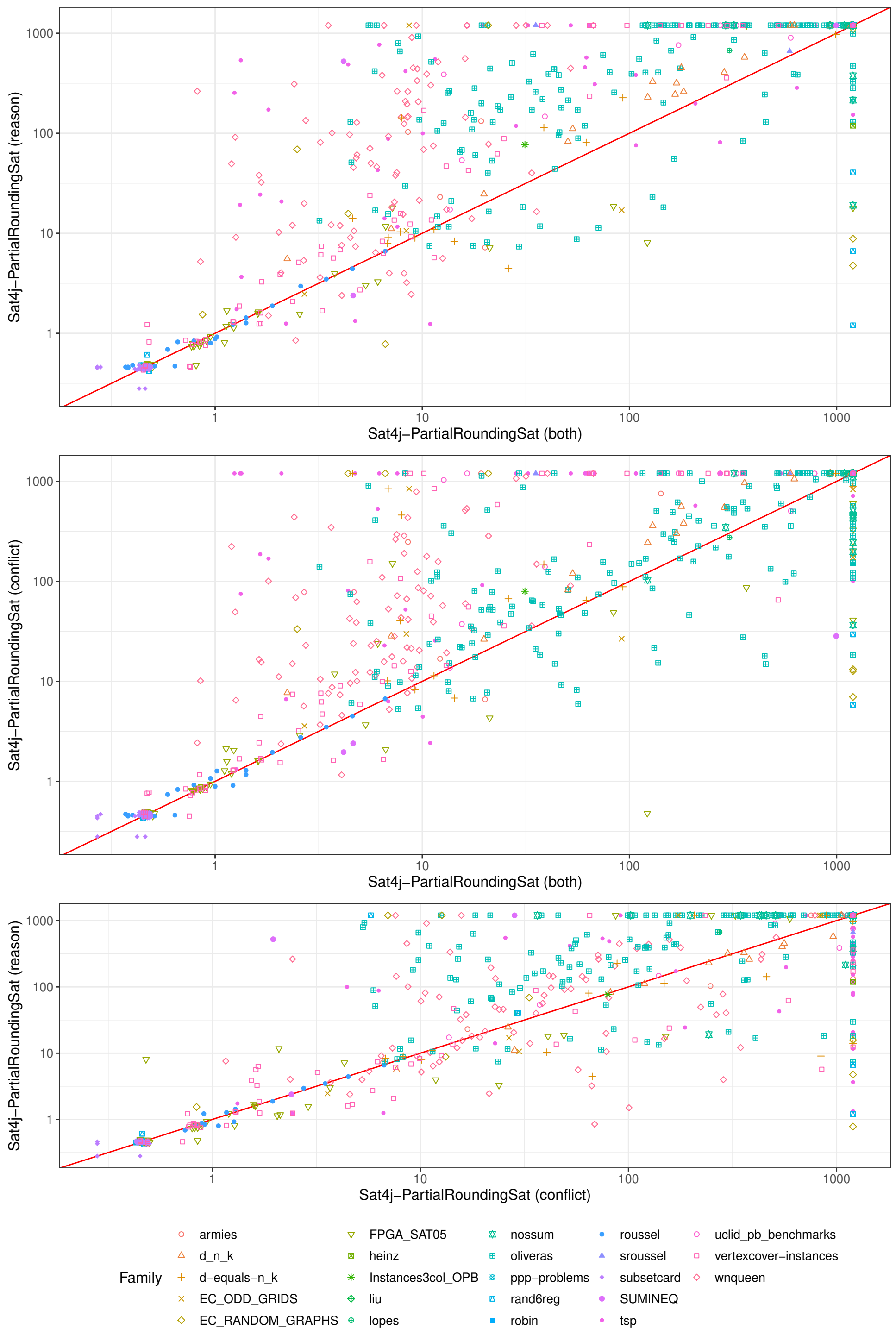


Figure 6.8: Scatter plots comparing the runtime (in seconds) of the different *Partial RoundingSat* variants in *Sat4j* (logarithmic scale).

As for the different *Sat4j-RoundingSat* variants, the number of instances solved by the different *Sat4j-PartialRoundingSat* variants is given in Table 6.3. This table confirms the observations made on the cactus plot, and in particular that applying the weakening and division rules on *both* sides of the cancellation has better performance than the other approaches. Quite interestingly, while the approach applying these operations on the conflict side is in general better than that applying them on the reason side, we can observe that this latter strategy has better performance on the `tsp` family (as well as that applying weakening and division on both sides).

Yet, the performance of *PartialRoundingSat (both)* (named *Sat4j-PartialRoundingSat* from now on) is not completely satisfactory. Indeed, *Sat4j-RoundingSat* is still faster on some benchmarks, as illustrated in Figure 6.10. To a lesser extent, we also observe that even *Sat4j-GeneralizedResolution* remains better on some instances, as shown by Figure 6.9.

Considering these latter observations, and the conclusions made regarding the different *Weaken Ineffective* strategies, it is clear that none of the strategies, and thus none of the underlying proof systems, has the best performance on all benchmarks.

Family	Number of instances in the family	Number of solved easy instances	*Sat4 GeneralizedResolution*	*Sat4j-PartialRoundingSat* (reason)	*Sat4j-PartialRoundingSat* (conflict)	*Sat4j-PartialRoundingSat* (both)	*Sat4j RoundingSat*
`Aardal_1`	14	14	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)
`armies`	12	1	3 (765.84)	4 (265.78)	4 (1025.44)	5 (201.84)	4 (74.6)
`caixa`	1	1	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)
`d_n_k`	234	155	14 (4318.16)	13 (3047.66)	13 (4672.28)	**17 (5052.98)**	**18 (6167.55)**
`d-equals-n_k`	70	28	12 (489.46)	13 (1607.26)	13 (2848.51)	13 (1278.3)	14 (3221.75)
`EC_ODD_GRIDS`	25	2	2 (65.3)	3 (30.12)	6 (1906.38)	4 (111.49)	5 (656.94)
`EC_RANDOM_GRAPHS`	22	3	2 (237.45)	6 (100.64)	5 (67.05)	5 (35.21)	5 (906.13)
`FPGA_SAT05`	57	0	43 (1835.72)	42 (1190.76)	45 (1577.34)	41 (649.44)	41 (2279.38)
`heinz`	4	1	0 (0)	1 (119.1)	0 (0)	0 (0)	0 (0)
`Instances3col_OPB`	26	7	1 (51.69)	1 (77.05)	1 (79.58)	1 (31.3)	1 (20.23)
`liu`	20	16	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)
`lopes`	193	0	1 (256.01)	1 (672.21)	1 (273.98)	2 (1422.57)	2 (1210.13)
`nossum`	180	0	2 (1567.69)	3 (609.05)	**10 (3862.79)**	6 (3932.72)	**10 (3378.61)**
`oliveras`	4080	3017	121 (27936.74)	92 (23343.48)	129 (31078.38)	**145 (33110.93)**	**151 (35341.14)**
`ppp-problems`	6	0	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)
`rand6reg`	33	4	3 (208.84)	9 (51.27)	8 (38.01)	6 (2.76)	7 (20.47)
`robin`	6	2	0 (0)	0 (0)	0 (0)	1 (1190.61)	0 (0)
`roussel`	40	0	20 (31.28)	20 (31.01)	20 (31.22)	20 (30.87)	20 (30.46)
`sroussel`	122	0	4 (853.18)	1 (661.72)	0 (0)	2 (628.59)	2 (326.76)
`subsetcard`	56	1	55 (23.81)	55 (24.59)	55 (24.82)	55 (24.53)	55 (24.42)
`SUMINEQ`	24	0	1 (5.72)	2 (525.86)	3 (32.72)	3 (1003.93)	3 (847.47)
`tsp`	100	0	14 (5756.76)	**30 (6148.71)**	17 (2643.42)	**35 (2717.07)**	14 (2151.31)
`uclid_pb_benchmarks`	50	29	5 (824.86)	6 (2266.85)	5 (1728.88)	**10 (2256.44)**	**10 (2252.06)**
`vertexcover-instances`	107	46	42 (1806.68)	46 (1174.96)	44 (2260.64)	**57 (2741.84)**	**56 (3292.67)**
`wnqueen`	100	18	22 (4923.28)	74 (8941.34)	79 (9336.16)	**82 (797.08)**	**82 (780.94)**

Table 6.3: Table summarizing, for each variants of *Sat4j-PartialRoundingSat* and *Sat4j*'s state-of-the-art solvers, the number of non-easy instances solved by this approach for each of the considered families. The numbers in parentheses correspond to the total runtime (in seconds) needed to solve the instances. Bold numbers highlight solvers that perform well on a given family compared to the others.

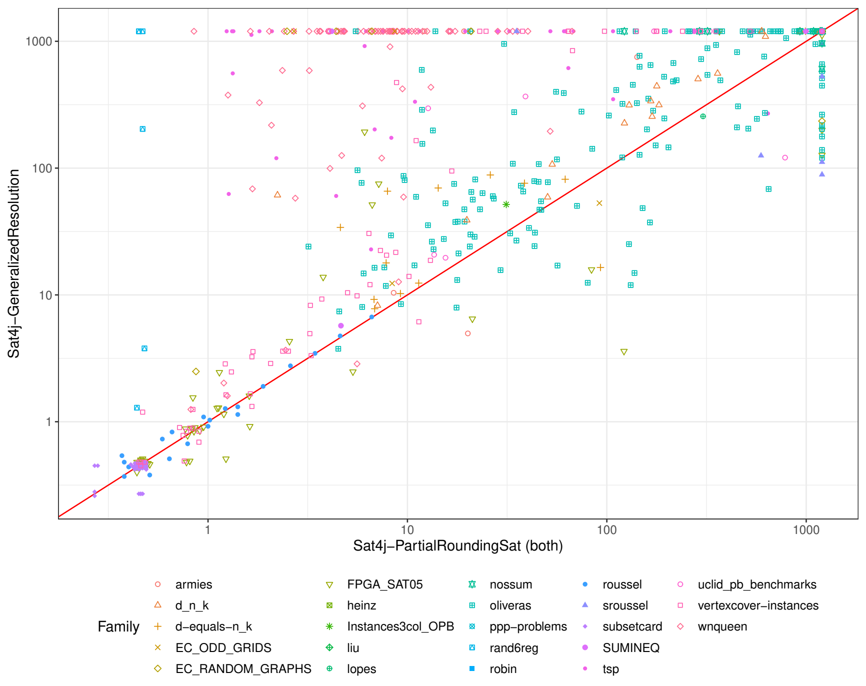


Figure 6.9: Scatter plot comparing the runtime (in seconds) of *Sat4j-GeneralizedResolution* and *Sat4j-PartialRoundingSat* (logarithmic scale).

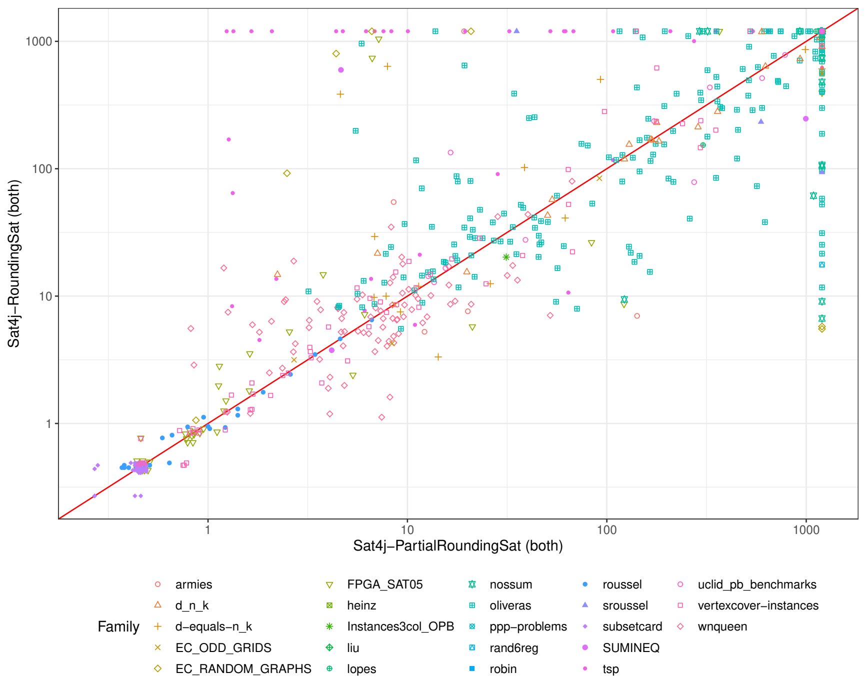


Figure 6.10: Scatter plot comparing the runtime (in seconds) of *Sat4j-RoundingSat* and *Sat4j-PartialRoundingSat* (logarithmic scale).

6.3 The Need for Tradeoffs

When using the *Weaken Ineffective* and *RoundingSat*-based strategies, we have observed that the weakening rule may help finding short explanations for conflicts, but may also infer weaker constraints. Our experiments have also revealed that none of these strategies is better than the others, especially when looking at their VBS or at their pairwise comparisons (see, for instance, Figures 6.2, 6.10 and 6.9).

In order to find better tradeoffs, we need to investigate new ways of applying the weakening rule. A possible approach for doing so is the following, that we call *Multiply and Weaken*. Let r be the coefficient of the pivot used in the cancellation appearing in the reason and c that in the conflict. Find two values μ and ν such that $(\nu - 1) \cdot r < \mu \cdot c \leq \nu \cdot r$ (which can be done using Euclidean division). Then, multiply the reason by ν, and apply successively weakening operations on this constraint so as to reduce the coefficient of the pivot to $\mu \cdot c$. Note that, to preserve the propagation, this coefficient cannot be weakened directly. Instead, ineffective literals (as described above) are successively weakened away so that the saturation rule produces the expected reduction on the coefficient (partial weakening may also be applied to make sure that the degree has exactly the appropriate value). Since this operation does not necessarily preserve the conflict, an additional weakening operation has to be performed, as for the generalized resolution. Note that this approach may also derive clauses, even though this is not always the case, as shown by the following example.

Example 86

Let us consider the constraint $5a(0@1) + 5b(?@?) + 3c(?@?) + 2d(0@2) + e(1@1) \geq 6$, which propagates b under the current partial assignment. Let us now consider the conflicting constraint $3\bar{b}(0@2) + 2a(0@1) + 2d(0@2) + \bar{e}(0@1) \geq 5$. Instead of using the lcm of 3 and 5 (i.e., 15), the reason of b is weakened on e and partially on c to get, after saturation, $3a(0@1) + 3b(0@1) + 2d(0@2) + c(?@?) \geq 3$. The cancellation produces then $5a(0@1) + 4d(0@2) + c(?@?) + \bar{e}(0@1) \geq 5$.

Using the same experimental configuration as in the previous sections, let us compare the results of this new strategy to the others studied before. The results are given in Figure 6.11.

One can observe that, unfortunately, the *Multiply and Weaken* strategy does not work well, even if it still exhibits better performance than *Sat4j-GeneralizedResolution*. Its main interest is to illustrate how to design new weakening schemes so as to improve the performance of pseudo-Boolean solvers.

A key observation to make from the cactus plot is that the VBS, computed here over all the strategies appearing in the plot, clearly beats each individual strategy, confirming that identifying good tradeoffs would allow significant improvements in pseudo-Boolean solving. This is also confirmed by the contributions to the VBS, given in Table 6.4.

The table confirms that the best results are given by the *PartialRoundingSat (both)* variant, which solves 144 more instances than *Sat4j-GeneralizedResolution* and 12 more than *RoundingSat (both)* (i.e., *Sat4j-RoundingSat*), even though the solved instances are not the same ones. For each strategy, none of its variants has a strong contribution to the VBS, since these variants are very similar given a main weakening strategy. However, if we consider the main strategies, and in particular *RoundingSat*, *Partial-RoundingSat* and *Weaken Ineffective*, their state-of-the art contributions (i.e., the instances only solved by these variants) become clearer: *Generalized Resolution* contributes 3 instances, *Multiply and Weaken* 4 instances *Sat4j-RoundingSat* 12 instances, *Sat4j-PartialRoundingSat* 17 instances, and *Weaken Ineffective* 76 instances. This confirms that choosing the *right* variant, even with the same main weakening strategy, plays a key role in the performance of the solver.

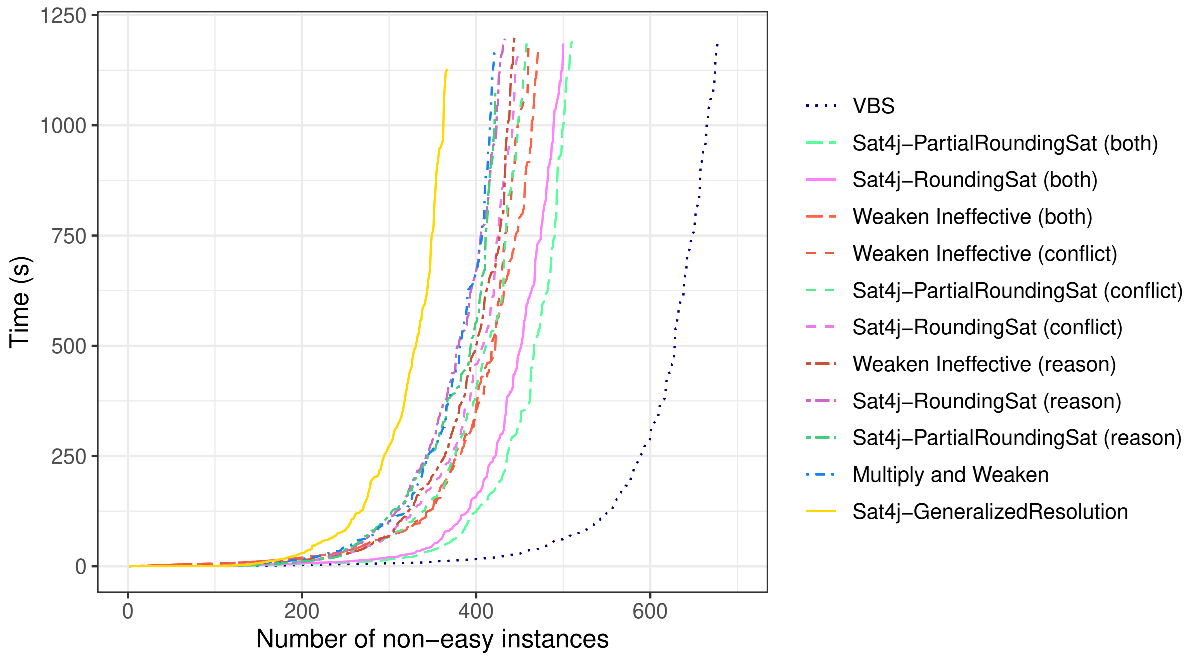


Figure 6.11: Cactus plot comparing the results of all the weakening strategies implemented in *Sat4j*.

The gain we observe between the different strategies has several plausible explanations. First, the solver does not explore the same search space from one strategy to another, and thus the constraints it learns may be completely different. In particular, they may be stronger or weaker, which has an impact on the size of the proof built by the solver, and thus on its runtime. Second, these constraints may contain different literals, which may have side effects on the VSIDS heuristic: different literals will be bumped during conflict analysis. Such side effects are hard to assess, due to the tight link between the heuristic used and the other components of the solver.

Weakening Strategy	Variant	Solved Instances	SOTA Contribution of Variant	SOTA Contribution of Strategy
Generalized Resolution		3711	3	3
Multiply and Weaken		3767	4	4
Partial RoundingSat	both	3855	6	17
Partial RoundingSat	conflict	3803	8	
Partial RoundingSat	reason	3766	2	
RoundingSat	both	3843	3	12
RoundingSat	conflict	3793	4	
RoundingSat	reason	3778	4	
Weaken Ineffective	both	3815	10	76
Weaken Ineffective	conflict	3804	6	
Weaken Ineffective	reason	3789	3	

Table 6.4: Table summarizing the results of the different weakening strategies and their variants. Columns display, from left to right, the main strategy, the variant of this strategy (if any), the number of instances solved by this variant, the state-of-the-art contribution of this variant and the state-of-the-art contribution of the main strategy.

Chapter 7

Evaluating the Impact of CDCL Strategies in Pseudo-Boolean Solvers

As described in Section 4.2, the implementation of the CDCL architecture in pseudo-Boolean solvers is widely inspired by the development of this architecture in SAT solvers. In particular, pseudo-Boolean solvers not only generalize the conflict analysis of classical SAT solvers, but also implement some complementary strategies that are tightly linked to the CDCL algorithm, namely branching heuristics, learned constraint deletion and restarts. However, these strategies inherited from SAT solving are most often reused "as they are" by current pseudo-Boolean solvers, without taking into account the particular form of the pseudo-Boolean constraints they deal with.

In this chapter, we show how these strategies may be adapted to pseudo-Boolean solvers, so as to retrieve the properties of their implementations in SAT solvers while applied to pseudo-Boolean constraints. In practice, we show that doing so allows to improve, sometimes significantly, the performance of the solvers.

7.1 Adapting (E)VSIDS for Pseudo-Boolean Constraints

Current implementations of the VSIDS heuristic in SAT solvers, and in particular the EVSIDS heuristic (see Subsection 4.1.3), are designed to favor the selection of variables that are involved in recent conflicts. When only considering clauses, identifying such literals is straightforward: the literals involved in a conflict are those appearing in the clauses encountered during conflict analysis. However, this is no longer the case when pseudo-Boolean constraints are considered. Indeed, given a pseudo-Boolean constraint, the literals it contains may not play the same role in the constraint, and thus may not have the same influence in the conflicts in which this constraint is involved. In order to take into account this asymmetry between the literals when computing VSIDS scores, we introduce different ways of bumping the variables appearing in the constraints encountered during conflict analysis.

The main reason for the asymmetry of the literals in a pseudo-Boolean constraint is the presence of coefficients in the constraint. To take these literals into account, pseudo-Boolean solvers such as *pbChaff* [DG02, Dix04, Section 4.5] and *Pueblo* [SS06] have introduced different bumping strategies (see Subsection 4.2.3 for more details). To generalize the heuristics implemented in these solvers, we define the following bumping strategies:

- The *bump-degree* strategy multiplies the increment by the *degree* of the constraint, as a naive generalization of *pbChaff*'s approach, which only considers the degree of the original cardinality constraints.

- The *bump-coefficient* strategy multiplies the increment by the *coefficient* of the literal being bumped, as a tentative measure of the importance of the corresponding variable.
- The *bump-ratio-coefficient-degree* strategy multiplies the increment by the *ratio* of the coefficient of the literal by the degree of the constraint, as proposed in *Pueblo*.
- The *bump-ratio-degree-coefficient* strategy multiplies the increment by the *ratio* of the degree of the constraint by the coefficient of the literal, as a generalization of *pbChaff*'s strategy taking into account the relative importance of the variable in the constraint.

Let us illustrate these different strategies by the following example.

Example 87

When bumping the variable a from the constraint $5a + 5b + c + d + e + f \geq 6$, the increment is multiplied by:

- 6 in the case of *bump-degree*,
- 5 in the case of *bump-coefficient*,
- $5/6$ in the case of *bump-ratio-coefficient-degree*, and
- $6/5$ in the case of *bump-ratio-degree-coefficient*

before being added to the variable's score.

To compare the performance of these different bumping strategies, we implemented them in the pseudo-Boolean solver *Sat4j*, and executed different configurations of this solver in the usual experimental setting (see Appendix B). The timeout was set to 1200 seconds and the memory limit to 32 GB.

Let us first study the impact of these heuristics in *Sat4j-GeneralizedResolution*, for which the results are given in Figure 7.1.

The cactus plot clearly shows that the best strategy is *bump-ratio-coefficient-degree* (i.e., the one implemented in *Pueblo*), which is also the only of the four strategies that beats the default one. We observe a similar behavior with the implementations of these heuristics in *Sat4j-RoundingSat* and *Sat4j-PartialRoundingSat*, for which the results are given in Figures 7.2 and 7.3, respectively.

The main difference between *Sat4j-GeneralizedResolution* and the *RoundingSat* implementations in *Sat4j* is the performance of the *bump-coefficient* strategy. A plausible explanation for this behavior is that, because of the use of the weakening and division rules, the coefficients and degrees of the learned constraints get more "balanced" in these configurations, so that the coefficient of a literal in itself is a sufficient measure of the relative importance of the corresponding variable in the constraint.

Another observation that we can make from these experiments is that, for all the solvers studied, the strategies *bump-degree* and *bump-ratio-degree-coefficient* have poor performance. Actually, this is not so surprising: as described in [Dix04, Section 4.5], these strategies are designed to estimate the number of clauses that are represented by the pseudo-Boolean constraint whose literals are being bumped. However, when a conflict occurs, not all these clauses are actually involved in the conflict, and thus some variables get "more bumped" than they should be.

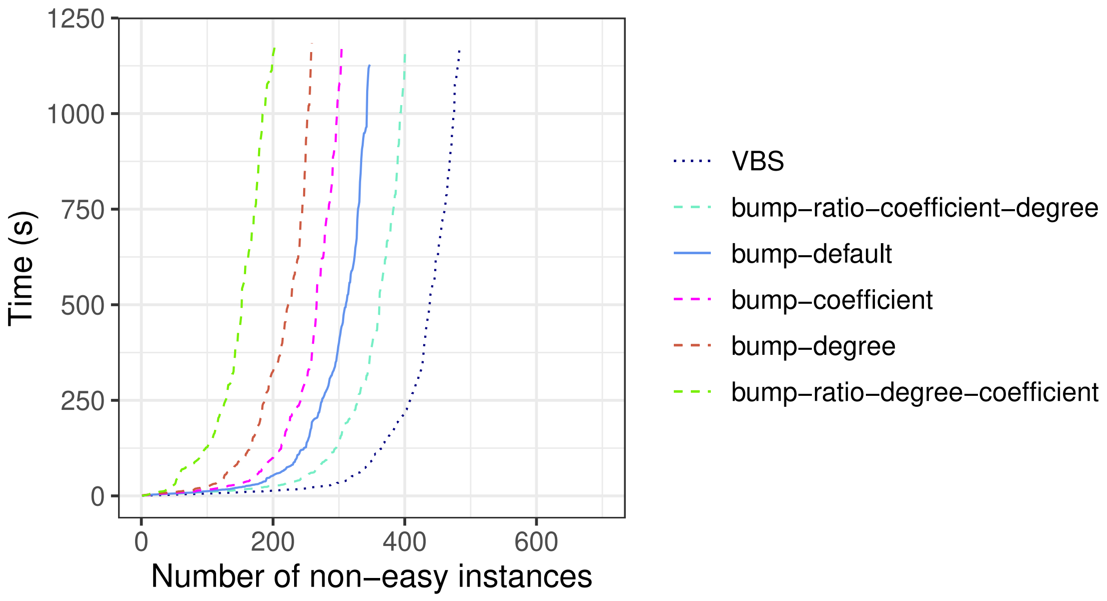


Figure 7.1: Cactus plot of the various coefficient and degree-based bumping strategies implemented in *Sat4j-GeneralizedResolution*.

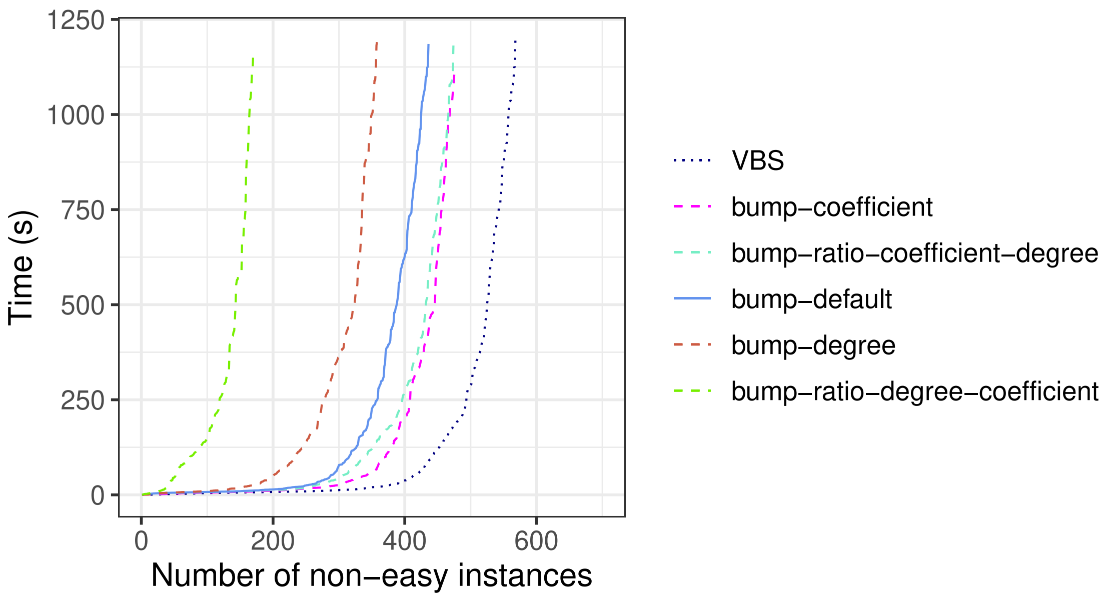


Figure 7.2: Cactus plot of the various coefficient and degree-based bumping strategies implemented in *Sat4j-RoundingSat*.

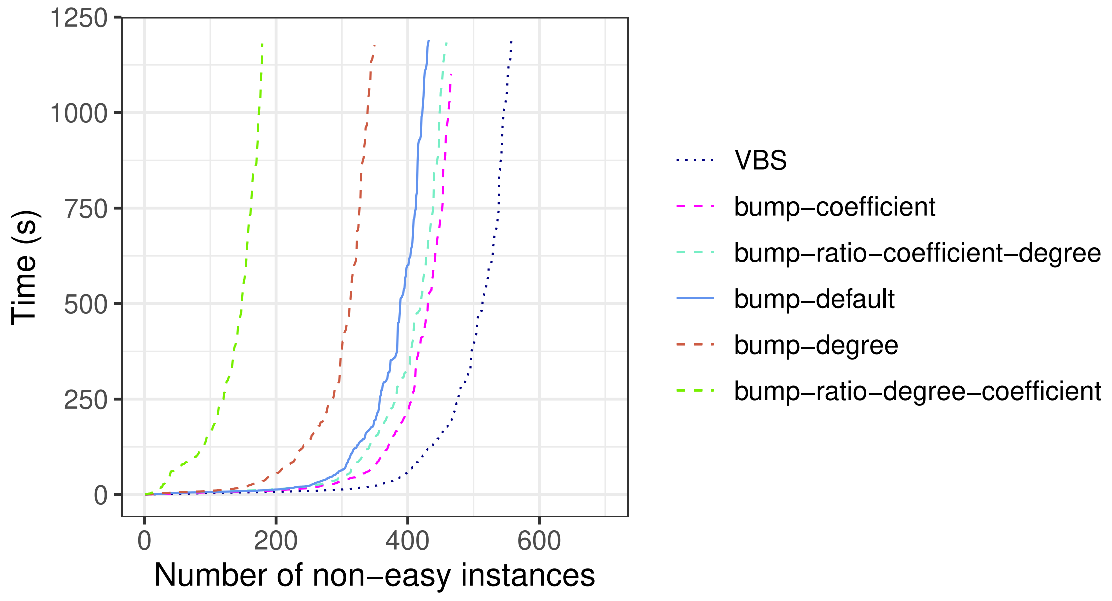


Figure 7.3: Cactus plot of the various coefficient and degree-based bumping strategies implemented in *Sat4j-PartialRoundingSat*.

Recall that modern implementations of VSIDS favor variables *involved* in the most recent conflicts. In this context, let us now consider another main difference between the clauses and pseudo-Boolean constraints encountered during conflict analysis: the existence of ineffective literals in the constraints (see Definition 107). Following this observation, we introduce five other bumping strategies, which take into account the current variable assignment to decide which variables should be bumped.

- The *bump-assigned* strategy bumps only assigned variables appearing in the constraints encountered during conflict analysis.
- The *bump-falsified* strategy bumps only variables whose literals appear as falsified in the constraints encountered during conflict analysis.
- The *bump-falsified-propagated* strategy bumps only variables whose literals appear as falsified in the constraints encountered during conflict analysis, and those that were propagated at the latest decision level.
- The *bump-effective* strategy bumps only variables whose literals are effective in the constraints encountered during conflict analysis.
- The *bump-effective-propagated* strategy bumps only variables whose literals are effective in the constraints encountered during conflict analysis, and those that were propagated at the latest decision level.

Example 88

When bumping the variables of the constraint $5a(0@3) + 5b(1@3) + c(?@?) + d(?@?) + e(0@1) + f(1@2) \geq 6$ at decision level 3,

- the strategy *bump-assigned* bumps the variables a, b, e and f,
- the strategy *bump-falsified* bumps the variables a and e,
- the strategy *bump-falsified-propagated* bumps the variables a, b and e,
- the strategy *bump-effective* bumps only the variable a, and
- the strategy *bump-effective-propagated* bumps the variables a and b.

As for the previous strategies, let us study the practical impact of these strategies in *Sat4j-GeneralizedResolution*, using the same experimental setting as above. The results of these experiments are given in Figure 7.4.

In the cactus plot, it is clear that considering the current assignment when bumping the literals has a significant impact on the performance of the solver. In this case, the *bump-effective* strategy, which allows to focus on the literals that are actually involved in the conflict, appears as the best one. However, in *RoundingSat*-based solvers, the *bump-assigned* and *bump-effective-propagated* strategies have better performance than the others, as shown in Figures 7.5 and 7.6.

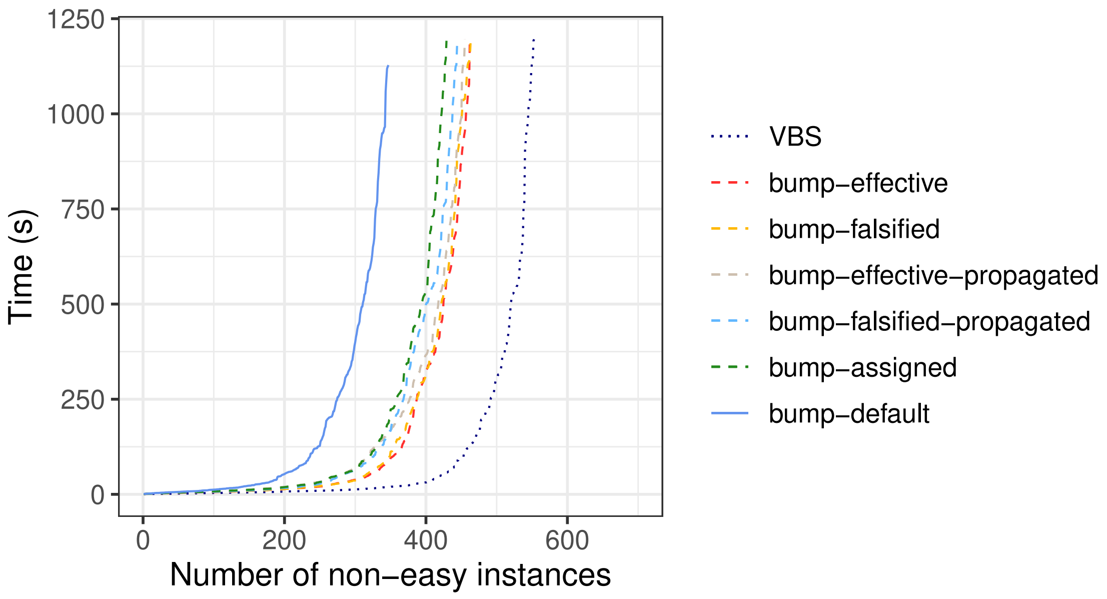


Figure 7.4: Cactus plot of the various assignment-based bumping strategies implemented in *Sat4j-GeneralizedResolution*.

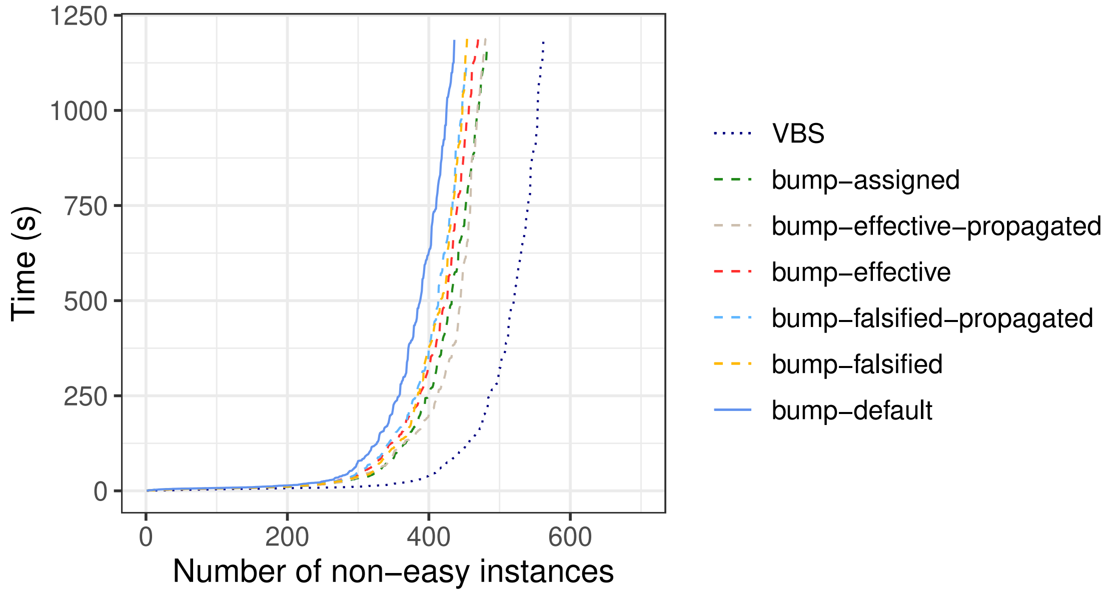


Figure 7.5: Cactus plot of the various assignment-based bumping strategies implemented in *Sat4j-RoundingSat*.

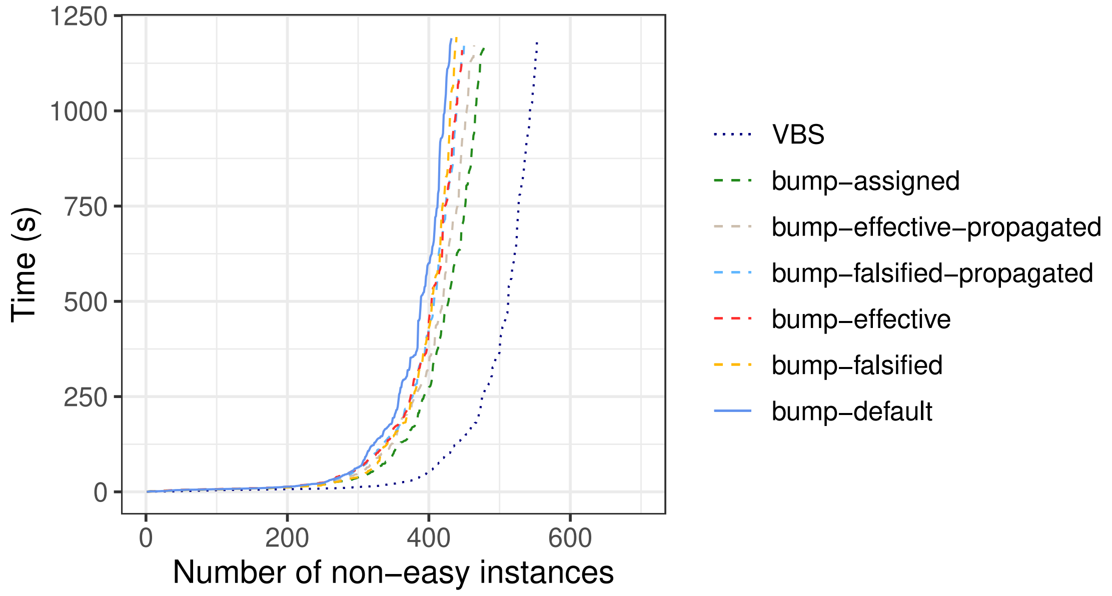


Figure 7.6: Cactus plot of the various assignment-based bumping strategies implemented in *Sat4j-PartialRoundingSat*.

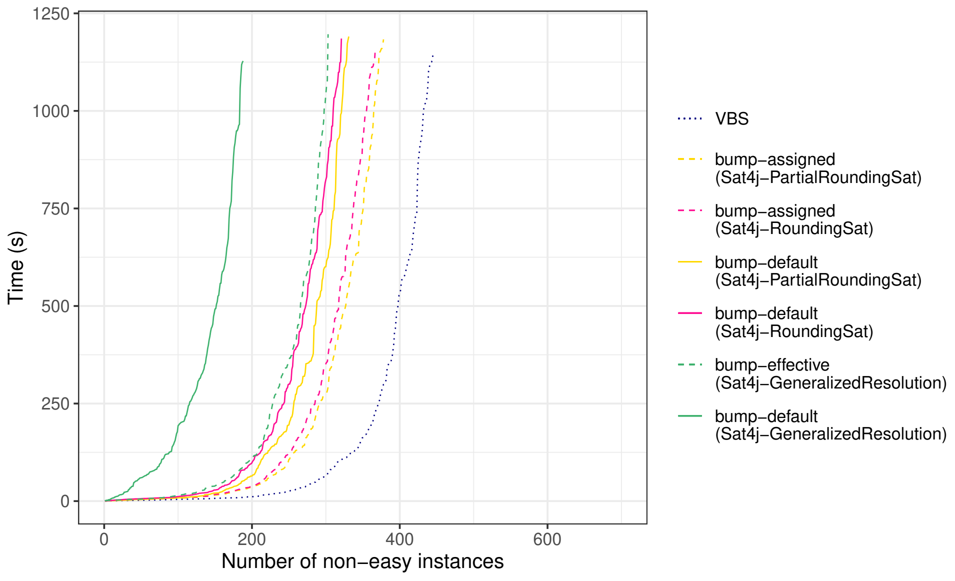


Figure 7.7: Cactus plot of the best bumping strategies implemented in different configurations of *Sat4j*.

The performance shift between the bumping strategies we study, and especially *bump-assigned* and *bump-effective*, in *Sat4j-GeneralizedResolution* and the two *RoundingSat* variants implemented in *Sat4j* may be explained by the aggressive weakening applied in the latter configurations. In a sense, this weakening strategy allows to keep the literals that are important in the constraint (and that will be bumped afterwards), by weakening away the other literals at each cancellation step during conflict analysis. This may also explain why the difference between the default bumping strategy and these two strategies is less marked in *RoundingSat*-based solvers than in *Sat4j-GeneralizedResolution*.

Let us now recapitulate the experiments we have conducted. Figure 7.7 shows, for each of the three considered configurations of *Sat4j*, the performance of the best bumping strategies that we observed in the previous experiments. From this cactus plot, we can see that the three solvers we have considered can be improved by choosing the appropriate bumping strategy. Quite interestingly, when the *bump-effective* strategy is used in *Sat4j-GeneralizedResolution*, this allows a significant improvement on the performance of the solver, so that it now solves 115 more instances than with the default heuristic, meaning that it solves only 17 less instances than *Sat4j-RoundingSat*. More detailed results are given in Table 7.1.

Remark 43

Note that it is possible to combine the coefficient and degree-based bumping strategies with assignment-based strategies. We performed some experiments in this direction, but in practice, doing so gives results worse than when the strategies are used independently. For more readability, these results are omitted in the cactus plots presented in this section.

Let us now consider the number of instances solved by the bumping strategies for the considered families, given in Tables 7.2, 7.3 and 7.4. Quite interestingly, one can observe that the strategy *bump-degree* which overall does not have good performance, is the best strategy for solving instances of the `FPGA_SAT05` family, for any of the three solvers we considered.

Bumping Strategy	Solved Instances	SOTA Contribution	1-second Contribution	10-second Contribution
bump-assigned (Sat4j-PartialRoundingSat)	3899	13	81	44
bump-coefficient (Sat4j-PartialRoundingSat)	3889	8	61	31
bump-assigned (Sat4j-RoundingSat)	3890	11	73	36
bump-coefficient (Sat4j-RoundingSat)	3883	8	57	24
bump-all (default) (Sat4j-PartialRoundingSat)	3855	6	58	22
bump-all (default) (Sat4j-RoundingSat)	3843	6	37	18
bump-effective (Sat4j-GeneralizedResolution)	3826	0	33	15
bump-ratio-coefficient-degree (Sat4j-GeneralizedResolution)	3763	6	32	21
bump-all (default) (Sat4j-GeneralizedResolution)	3711	2	18	11

Table 7.1: Table summarizing the results of the best bumping strategies for several configurations of *Sat4j*. Columns display, from left to right the bumping strategy, the number of instances solved by this strategy, its state-of-the-art contribution, the number of instances for which the configuration was at least 1 second faster than all the others and the number of instances for which the configuration was at least 10 seconds faster than all the others

Family	Number of instances in the family	Number of solved easy instances	*bump-effective*	*bump-falsified*	*bump-effective-propagated*	*bump-falsified-propagated*	*bump-assigned*	*bump-ratio-coefficient-degree*	*bump-default*	*bump-coefficient*	*bump-degree*	*bump-ratio-degree-coefficient*
`Aardal_1`	14	13	1 (6.29)	1 (5.23)	1 (4.52)	1 (4.73)	1 (1.19)	1 (3.6)	1 (2.13)	1 (2.51)	1 (2.49)	0 (0)
`armies`	12	1	6 (841.77)	6 (770.5)	4 (562.68)	5 (1344.75)	4 (154.29)	4 (314.69)	3 (774.01)	3 (135.41)	4 (96.72)	4 (386.64)
`caixa`	1	1	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)
`d_n_k`	234	143	**25 (4708.17)**	**26 (5076.87)**	**25 (4309.88)**	**25 (4664.6)**	**25 (3334.16)**	**25 (4118.02)**	**26 (4531.7)**	16 (4076.74)	11 (900)	5 (1373.65)
`d-equals-n_k`	70	18	18 (2156.29)	21 (2910.36)	20 (1972.58)	18 (3065.9)	23 (2876.88)	21 (1904.82)	22 (620.26)	12 (2282.32)	21 (2354.35)	13 (2612.69)
`EC_ODD_GRIDS`	25	1	6 (267.4)	5 (54.7)	5 (93.16)	3 (17.72)	4 (68.82)	3 (34.68)	3 (70.46)	3 (19.81)	2 (11.53)	1 (41.98)
`EC_RANDOM_GRAPHS`	22	3	2 (271.12)	1 (4.63)	2 (29.17)	3 (85.69)	1 (2.02)	2 (110.32)	2 (241.69)	3 (750.39)	2 (68.74)	2 (254.69)
`FPGA_SAT05`	57	29	6 (1445.99)	5 (1225.09)	6 (895.69)	5 (68.63)	**13 (1809.74)**	**13 (1999.92)**	**14 (1813.52)**	9 (351.29)	**12 (2464.33)**	**15 (968.06)**
`heinz`	4	1	0 (0)	0 (0)	1 (395.27)	0 (0)	0 (0)	1 (1169.8)	0 (0)	0 (0)	0 (0)	0 (0)
`Instances3col_OPB`	26	7	1 (118.22)	1 (295.02)	1 (45.04)	1 (30.94)	1 (49.96)	1 (196.12)	1 (51.69)	1 (59.65)	1 (63.45)	1 (293.82)
`liu`	20	16	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)
`lopes`	193	0	2 (1016.8)	3 (2122.97)	3 (1851.95)	2 (959.35)	2 (679.3)	2 (280.16)	1 (256.01)	1 (312.07)	1 (315.5)	0 (0)
`nossum`	180	0	7 (1119.73)	6 (1622.41)	6 (1291.3)	5 (1751.14)	6 (1297.85)	8 (4405.32)	2 (1567.69)	3 (208.27)	2 (891.18)	5 (1163.28)
`oliveras`	4080	2953	197 (30667.67)	197 (32570.78)	**210 (32433.45)**	198 (32242.84)	197 (30456.01)	**211 (33597.55)**	185 (28872.83)	147 (29145.39)	127 (24916.4)	103 (34496.74)
`ppp-problems`	6	0	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)
`rand6reg`	33	6	3 (1172.55)	2 (343.09)	3 (1515.03)	2 (437.66)	3 (122.94)	1 (13.46)	1 (203.76)	2 (168.96)	1 (35.7)	0 (0)
`robin`	6	1	2 (90.62)	2 (166.96)	1 (8.35)	1 (25.71)	2 (499.19)	1 (23.48)	1 (8.2)	1 (24.84)	1 (15.17)	1 (139.23)
`roussel`	40	20	0 (0)	2 (448.08)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)
`sroussel`	122	0	0 (0)	2 (1045.92)	3 (794.58)	0 (0)	1 (517.05)	0 (0)	4 (853.18)	0 (0)	1 (42.11)	0 (0)
`subsetcard`	56	56	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)
`SUMINEQ`	24	0	3 (14.58)	1 (8.33)	3 (101.67)	1 (2.78)	1 (173.94)	2 (74.94)	1 (5.72)	1 (2.71)	0 (0)	0 (0)
`tsp`	100	0	**40 (6170.71)**	**39 (3343.67)**	**40 (6601.86)**	**38 (4299.49)**	**38 (5170.55)**	34 (2450.36)	14 (5756.76)	32 (6393.93)	24 (8208.48)	27 (13855.19)
`uclid_pb_benchmarks`	50	28	11 (392.82)	12 (760.25)	11 (322.49)	12 (2022.1)	10 (1343)	**15 (1547.87)**	6 (815.2)	10 (1059.8)	5 (2278.72)	4 (1141.26)
`vertexcover-instances`	107	66	37 (2071.71)	**38 (710.83)**	37 (3604.58)	34 (3601.99)	32 (3576.36)	26 (2286.06)	22 (1780.05)	18 (868.98)	13 (1361.93)	1 (694.88)
`wnqueen`	100	2	**95 (1565.63)**	**93 (6220.92)**	73 (4354.94)	**90 (4469.18)**	65 (2349.3)	30 (2576.3)	38 (5059.07)	42 (6363.49)	30 (5454.26)	21 (4999.84)

Table 7.2: Table summarizing, for each of the considered heuristics, the number of non-easy instances solved by *Sat4j-GeneralizedResolution* when using this heuristic for each of the considered families. The numbers in parentheses correspond to the total runtime (in seconds) needed to solve the instances. Bold numbers highlight strategies that perform well on a given family compared to the others.

Family	Number of instances in the family	Number of solved easy instances	*bump-assigned*	*bump-effective-propagated*	*bump-coefficient*	*bump-ratio-coefficient-degree*	*bump-effective*	*bump-falsified-propagated*	*bump-falsified*	*bump-default*	*bump-degree*	*bump-ratio-degree-coefficient*
`Aardal_1`	14	11	3 (6)	3 (5.53)	3 (2.97)	3 (2.43)	3 (4.07)	3 (14)	3 (21.11)	3 (3.23)	3 (3.82)	0 (0)
`armies`	12	2	5 (1586.59)	6 (1320.42)	3 (1253.99)	4 (1647.15)	4 (1365.66)	2 (1238.28)	4 (501.58)	3 (67)	3 (269.67)	5 (1478.02)
`caixa`	1	1	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)
`d_n_k`	234	152	20 (4779.79)	20 (5137.54)	18 (5254.8)	19 (4184.29)	19 (4514.05)	19 (4690.34)	19 (4114.75)	21 (6213.29)	18 (4010.79)	13 (3869.43)
`d-equals-n_k`	70	23	17 (1792.14)	16 (1843)	17 (754.83)	16 (2979.99)	17 (5010.58)	15 (2469.14)	16 (2012.58)	19 (3258.66)	18 (2274.92)	16 (3923.42)
`EC_ODD_GRIDS`	25	3	5 (185.45)	5 (220.96)	5 (389.96)	4 (346.56)	5 (563.64)	4 (77.25)	6 (449.48)	4 (653.78)	1 (0.76)	2 (1051.5)
`EC_RANDOM_GRAPHS`	22	3	5 (59.14)	4 (335.92)	16 (1216.14)	3 (352.8)	4 (284.16)	7 (161.48)	4 (1039.41)	5 (909.52)	14 (1856.55)	6 (264.98)
`FPGA_SAT05`	57	32	11 (1513.18)	8 (1064.12)	10 (1060.66)	8 (413.33)	6 (1005.77)	4 (46)	6 (2153.16)	9 (2233.1)	15 (1710.83)	**12 (449.74)**
`heinz`	4	1	0 (0)	1 (46.87)	1 (827.58)	0 (0)	1 (1163.44)	1 (87.97)	0 (0)	0 (0)	0 (0)	1 (73.57)
`Instances3col_OPB`	26	7	1 (33.3)	1 (39.74)	1 (202.19)	1 (24.24)	1 (33.72)	1 (20.54)	1 (117.23)	1 (20.23)	1 (45.37)	1 (45.27)
`liu`	20	16	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)
`lopes`	193	0	3 (1875.42)	3 (763.39)	1 (412.35)	4 (2317.81)	3 (813.85)	3 (1128.52)	2 (1153.69)	2 (1210.13)	2 (433.16)	0 (0)
`nossum`	180	0	12 (6048.48)	11 (3785.5)	5 (2120.49)	13 (5259.79)	11 (6103.59)	10 (4221.42)	8 (2318.46)	10 (3378.61)	3 (2034.69)	1 (272.35)
`oliveras`	4080	2915	255 (34156.28)	264 (34085.54)	233 (27906.34)	263 (34862.09)	259 (37111.88)	250 (31181.56)	252 (35266.18)	253 (36221.7)	187 (30975.65)	61 (16030.95)
`ppp-problems`	6	0	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)
`rand6reg`	33	9	2 (205.2)	2 (42.69)	15 (1809.86)	1 (991.95)	1 (6.42)	1 (7.43)	1 (0.47)	2 (18.16)	11 (175.6)	4 (379.93)
`robin`	6	2	2 (1474.1)	1 (183.02)	0 (0)	1 (280.89)	0 (0)	1 (773.74)	1 (136.07)	0 (0)	1 (459.61)	0 (0)
`roussel`	40	20	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	2 (399.94)	0 (0)	0 (0)	0 (0)
`sroussel`	122	0	2 (586.93)	1 (237.1)	2 (1829.78)	0 (0)	4 (1639.68)	1 (216.65)	2 (626.66)	2 (326.76)	2 (188.84)	0 (0)
`subsetcard`	56	56	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)
`SUMINEQ`	24	0	3 (1204.77)	6 (2529.12)	11 (1083.4)	5 (1349.07)	2 (7.8)	2 (200.91)	1 (14.58)	3 (847.47)	1 (58.04)	1 (998.9)
`tsp`	100	0	47 (6618.32)	43 (4386.11)	48 (2812.3)	40 (3967.67)	41 (3415.01)	42 (6630.65)	39 (3409.6)	14 (2151.31)	9 (1907.96)	3 (2274.59)
`uclid_pb_benchmarks`	50	26	13 (335.16)	13 (366.56)	15 (853)	14 (592.26)	15 (1629.64)	14 (1505.88)	13 (585.48)	13 (2270.43)	4 (520.69)	5 (1174.16)
`vertexcover-instances`	107	68	38 (4474.99)	35 (2267.89)	34 (1538.22)	37 (2947.41)	36 (2461.49)	36 (3378.95)	38 (1851.58)	34 (3268.84)	27 (2732.62)	3 (696.19)
`wnqueen`	100	62	38 (359.66)	38 (359.17)	38 (1695.72)	38 (441.86)	38 (258.51)	38 (268.07)	38 (303.11)	38 (533.07)	38 (8820.38)	36 (6206.3)

Table 7.3: Table summarizing, for each of the considered heuristics, the number of non-easy instances solved by *Sat4j-RoundingSat* when using this heuristic for each of the considered families. The numbers in parentheses correspond to the total runtime (in seconds) needed to solve the instances. Bold numbers highlight strategies that perform well on a given family compared to the others.

Family	Number of instances in the family	Number of solved easy instances	*bump-assigned*	*bump-coefficient*	*bump-effective-propagated*	*bump-ratio-coefficient-degree*	*bump-falsified-propagated*	*bump-effective*	*bump-falsified*	*bump-default*	*bump-degree*	*bump-ratio-degree-coefficient*
`Aardal_1`	14	9	5 (5.7)	5 (4.28)	5 (4.69)	5 (4.19)	5 (5.11)	5 (6.39)	5 (6.67)	5 (3.91)	5 (6.42)	2 (413.09)
`armies`	12	2	5 (512.13)	1 (599.26)	3 (148.35)	3 (1197.75)	3 (186.61)	2 (900.65)	3 (233.44)	4 (181.75)	3 (753.09)	4 (896.01)
`caixa`	1	1	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)
`d_n_k`	234	152	20 (6773.09)	18 (4601.71)	20 (5656.64)	21 (6690.5)	19 (5577.56)	20 (5156.68)	21 (6229.61)	20 (5099.91)	20 (7082.36)	12 (4998.27)
`d-equals-n_k`	70	23	18 (1422.62)	18 (1597.12)	17 (3621.62)	19 (2858.79)	14 (1883.65)	14 (760.97)	16 (2096.05)	18 (1337.56)	19 (2112.49)	14 (2583.39)
`EC_ODD_GRIDS`	25	3	6 (2058.67)	6 (844.32)	5 (335.22)	6 (1704.36)	4 (53.3)	6 (319.84)	6 (639.6)	3 (108.79)	2 (9.5)	1 (34.55)
`EC_RANDOM_GRAPHS`	22	4	6 (1041.86)	**14 (584.34)**	3 (42.47)	3 (18.59)	7 (1322.05)	2 (91.75)	4 (564.49)	4 (34.34)	**14 (551.35)**	7 (1271.36)
`FPGA_SAT05`	57	31	11 (1879.16)	12 (1856.08)	9 (1822.46)	10 (437.06)	7 (2628.76)	5 (834.55)	3 (25.48)	10 (618.61)	**15 (833.92)**	15 (1779.39)
`heinz`	4	1	1 (80.89)	0 (0)	1 (80.61)	1 (888.33)	1 (169.34)	1 (23.9)	0 (0)	0 (0)	0 (0)	1 (45.22)
`Instances3col_OPB`	26	7	1 (38.71)	1 (59.68)	1 (30.59)	1 (17.32)	1 (21.69)	1 (25.37)	1 (132.91)	1 (31.3)	1 (48.28)	1 (46.18)
`liu`	20	16	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)
`lopes`	193	0	3 (1675.75)	2 (683.02)	3 (1599.55)	2 (951.33)	3 (1362.88)	3 (1007.61)	3 (1100.08)	2 (1422.57)	1 (400.68)	2 (1572.28)
`nossum`	180	0	**16 (6661.7)**	4 (1549.83)	**13 (7014.46)**	**10 (2760.78)**	**13 (5047.33)**	**13 (4877.84)**	8 (2046.31)	6 (3932.72)	2 (292.61)	2 (637.54)
`oliveras`	4080	2929	245 (34088.45)	224 (30036.23)	248 (31989.84)	244 (33870.67)	240 (35331.53)	237 (34003.67)	233 (29737.9)	233 (33933.1)	172 (27939.81)	74 (23166.23)
`ppp-problems`	6	0	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)
`rand6reg`	33	9	4 (118.89)	**14 (624.26)**	0 (0)	2 (538.09)	3 (62.45)	1 (14.16)	1 (0.46)	1 (0.47)	8 (897.55)	4 (283.42)
`robin`	6	2	1 (690.56)	0 (0)	1 (249.46)	0 (0)	0 (0)	1 (119.08)	0 (0)	1 (1190.61)	0 (0)	0 (0)
`roussel`	40	20	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	2 (405.08)	0 (0)	0 (0)	0 (0)
`sroussel`	122	0	2 (863.11)	1 (390.25)	3 (1677.26)	3 (773.86)	2 (956.6)	4 (2424.16)	1 (182.44)	2 (628.59)	1 (424.58)	0 (0)
`subsetcard`	56	56	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)
`SUMINEQ`	24	0	3 (170.23)	**11 (722.66)**	5 (1252.21)	4 (1199.46)	4 (1165)	4 (54.16)	3 (1414.04)	3 (1003.93)	1 (179.94)	0 (0)
`tsp`	100	0	48 (4612.54)	49 (2373.03)	41 (5414.25)	39 (2851.31)	42 (3426.18)	41 (3521.45)	42 (3331.81)	35 (2717.07)	20 (6882.95)	0 (0)
`uclid_pb_benchmarks`	50	26	12 (736.12)	16 (1653.32)	14 (1736.14)	13 (341.58)	12 (456.38)	14 (880.42)	14 (773.57)	13 (2270.02)	3 (117.26)	5 (1515.7)
`vertexcover-instances`	107	66	38 (4636.63)	36 (1384.64)	38 (2280.35)	39 (3707.02)	36 (1914.29)	39 (3758.16)	39 (1247.57)	37 (2720.3)	29 (3311.48)	2 (251.28)
`wnqueen`	100	66	34 (361.99)	34 (999.02)	34 (312.01)	34 (515.65)	34 (261.36)	34 (268.48)	34 (278.15)	34 (539.76)	34 (4830.24)	33 (6461.29)

Table 7.4: Table summarizing, for each of the considered heuristics, the number of non-easy instances solved by *Sat4j-PartialRoundingSat* when using this heuristic for each of the considered families. The numbers in parentheses correspond to the total runtime (in seconds) needed to solve the instances. Bold numbers highlight strategies that perform well on a given family compared to the others.

The bumping strategies studied in this section showed that carefully taking into account the particular form of pseudo-Boolean constraints (either through their coefficients and degrees or through their properties under the current partial assignment) in the branching heuristic of the solver may drastically improve its performance. Let us now study how similar improvements may be brought by considering these criteria when measuring the quality of learned constraints.

7.2 Measuring the Quality of Learned Pseudo-Boolean Constraints

In order to evaluate the quality of the reasoning of the solver during its execution, a common approach is evaluating the quality of the constraints that are learned during conflict analysis. To do so, current pseudo-Boolean solvers rely on measures used in classical SAT solvers, and thus on measures designed to evaluate the quality of learned *clauses* (see Subsection 4.1.3). As pseudo-Boolean solvers may infer general pseudo-Boolean constraints, we study new quality measures for assessing the quality of these constraints, and show how they can be used to improve the performance of the solver.

7.2.1 Introducing New Quality Measures

Many quality measures have been designed to evaluate the quality of learned clauses in SAT solvers. Some of them can be reused as they are, as they do not take into account the representation nor the semantics of the constraints they evaluate. This is the case of the age-based and activity-based measures (see Subsection 4.1.3). However, for other evaluation schemes, paying attention to the particular form of pseudo-Boolean constraints may be more relevant to properly evaluate the quality of the constraints.

This is the case, for example, of the size-based measure, which deletes large clauses, containing many literals. The intuition behind this evaluation scheme is that large clauses are weak, especially from a propagation viewpoint: a propagation can only be triggered after many literals have become falsified. When considering pseudo-Boolean constraints, this is not the case anymore. Indeed, recall that pseudo-Boolean constraints may propagate literals while some other literals remain unassigned, and that the number of literals in a pseudo-Boolean constraint does not necessarily reflect its strength. Yet, evaluating *precisely* the strength of a pseudo-Boolean constraint is hard in practice: for instance, counting the number of models of such a constraint is already NP-hard (see Proposition 13). Instead, we can use the (absolute) *slack* of the pseudo-Boolean constraint as a heuristic to estimate the strength of the constraint. Intuitively, strong constraints have a slack that is close to 0: recall that, in particular, if a constraint has a slack equal to 0, then it propagates all its literals.

Another reason that motivated the use of size-based measures in SAT solving is that large clauses are expensive to handle, which is also true for pseudo-Boolean constraints. In particular, in such constraints, the size also takes into account the size of the coefficients, which is not negligible: as coefficients may become very large during conflict analysis, arbitrary precision encoding is required to represent these coefficients. As we already discussed, this representation slows down arithmetic operations, and thus the conflict analysis performed by the solver. Different approaches have been studied to limit the growth of the coefficient, such as those based on the division [EN18] or the weakening [LMW20] rules. However, these approaches lead to the inference of weaker constraints. By using a quality measure that takes into account the size of the coefficients, we can favor the learning of constraints with "small" coefficients. Towards this direction, we introduce quality measures based on the degree of the learned constraints, as described below:

- The *degree* quality measure evaluates the quality of a learned constraint by the *value* of its degree.
- The *degree-size* quality measure evaluates the quality of a learned constraint by the *size* of its degree, measured in the minimum number of bits required to represent it.

In both cases, the smaller the degree, the better the constraint. Indeed, thanks to the saturation rule, all coefficients are guaranteed to be upper bounded by the degree of the constraint, so that considering only the degree is enough for the purpose of this measure.

Example 89

The coefficient and degree-based quality measures for the constraint $5a+5b+c+d+e+f \geq 6$ are:

- 8 in the case of *slack*,
- 6 in the case of *degree*,
- 3 in the case of *degree-size* (as the binary representation of 6, i.e., 110, needs 3 bits).

More recently, the LBD measure [AS09] has been introduced to evaluate the quality of learned clauses in SAT solvers. Recall that the LBD of a clause is first computed when this clause is learned, and is then updated each time the clause is used as a reason (see Subsection 4.1.3 for more details). In this context, the notion of LBD relies on the fact that all literals in a conflicting clause are falsified, and when the clause is used as a reason, only one literal is not falsified (the propagated literal), but its decision level is also that of another (falsified) literal, which has triggered the propagation. When pseudo-Boolean constraints are considered, this is not the case anymore. As such, LBD is not well-defined for such constraints. To consider it as a quality measure for learned pseudo-Boolean constraints, we thus need to take into account the literals that are unassigned in these constraints. To do so, we introduce five different definitions of this measure. First, we consider a sort of default definition of LBD for pseudo-Boolean constraints, which only takes into account assigned literals. This definition of LBD was used for instance in the first version *RoundingSat* [EN18].

Definition 108 (LBD_a)

Consider a pseudo-Boolean constraint χ and the current assignment of its *assigned* literals. Let π be a partition of these literals, such that literals are partitioned w.r.t. their decision levels. The LBD_a of χ is the number of elements in π ("a" stands for "assigned").

Second, unassigned literals may be considered as if they were assigned to a "dummy" decision level. This decision level may be the same for all literals, or not.

Definition 109 (LBD_s)

Consider a pseudo-Boolean constraint χ and the current assignment of its *assigned* literals. Let π be a partition of these literals, such that literals are partitioned w.r.t. their decision levels. Let n be the number of elements in π. The LBD_s of χ is n if all literals in χ are assigned, and $n+1$ otherwise ("s" stands for "same").

Definition 110 (LBD_d)

Consider a pseudo-Boolean constraint χ and the current assignment of its *assigned* literals. Let π be a partition of these literals, such that literals are partitioned w.r.t. their decision levels. Let n be the number of elements in π. The LBD_d of χ is $n + u$, where u is the number of unassigned literals in χ ("d" stands for "different").

Another possible extension of standard LBD is to only consider falsified literals, as in the latest version of *RoundingSat*:

Definition 111 (LBD_f)

Consider a pseudo-Boolean constraint χ and the current assignment of its *falsified* literals. Let π be a partition of these literals, such that literals are partitioned w.r.t. their decision levels. The LBD_f of χ is the number of elements in π ("f" stands for "falsified").

The definition above is based on the observation that, when a clause is learned, all literals in this clause are falsified. Recall that they are also *effective* (see Definition 107), so that we can also restrict the computation of the LBD on such literals:

Definition 112 (LBD_e)

Consider a pseudo-Boolean constraint χ and the current assignment of its *effective* literals. Let π be a partition of these literals, such that literals are partitioned w.r.t. their decision levels. The LBD_e of χ is the number of elements in π ("e" stands for "effective").

Example 90

The LBD-based quality measures for the constraint χ given by $5a(0@3) + 5b(1@3) + c(?@?) + d(?@?) + e(0@1) + f(1@2) \geq 6$ are:

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

Remark 44

All the definitions of LBD introduced in this section are *extensions* of the original definition of LBD (as given by Definition 102), in the sense that they all coincide when learning clauses.

Thanks to the different quality measures we have introduced in this section, we can refine the learned constraint deletion and restart schemes implemented in pseudo-Boolean solvers, so as to take into account the particular form of the constraints they deal with.

7.2.2 Application to Learned Constraint Deletion

The main purpose of the quality measures is to choose, when constraint deletion is performed (see Subsection 4.1.3), which constraints to delete and which ones to keep. Taking advantage of the measures described above, we define the following deletion strategies, which are considered each time the learned clause database is reduced:

- *delete-slack*, which deletes the constraints with the highest *slack*,
- *delete-degree*, which deletes the constraints with the highest *degree*,
- *delete-degree-size*, which deletes the constraints with the highest *degree-size*,
- *delete-lbd-a*, which deletes the constraints with the highest LBD_a,
- *delete-lbd-s*, which deletes the constraints with the highest LBD_s,
- *delete-lbd-d*, which deletes the constraints with the highest LBD_d,
- *delete-lbd-f*, which deletes the constraints with the highest LBD_f, and
- *delete-lbd-e*, which deletes the constraints with the highest LBD_e.

Let us study the practical impact of these strategies for different configurations of the pseudo-Boolean solver *Sat4j*, with the same experimental setting as above. The results of these experiments are given in Figures 7.8, 7.9 and 7.10.

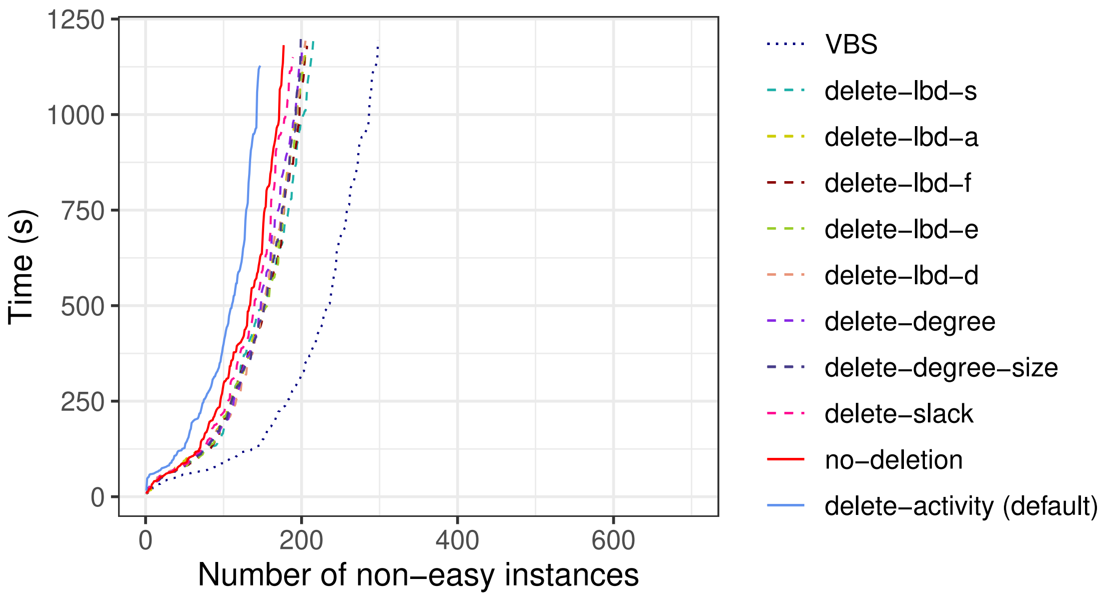


Figure 7.8: Cactus plot of the various learned constraint deletion strategies implemented in *Sat4j-GeneralizedResolution*.

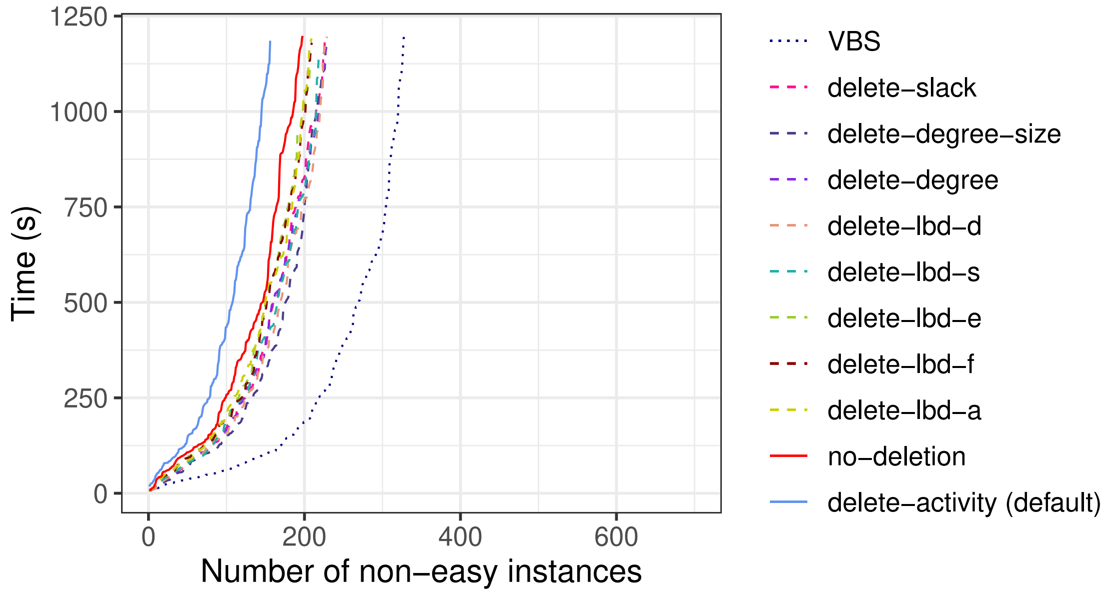


Figure 7.9: Cactus plot of the various learned constraint deletion strategies implemented in *Sat4j-RoundingSat*.

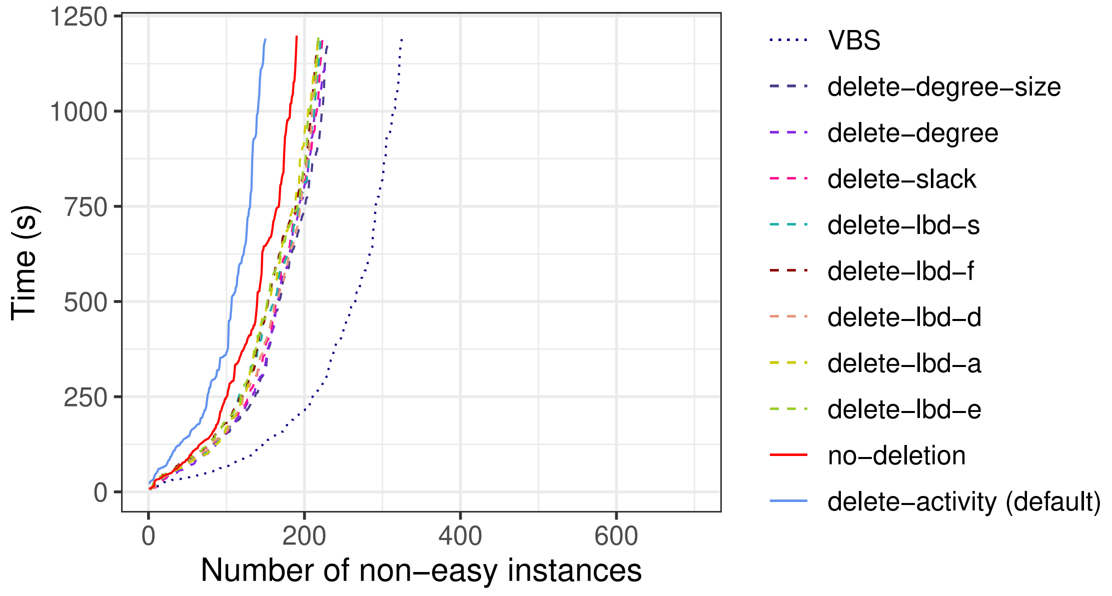


Figure 7.10: Cactus plot of the various learned constraint deletion strategies implemented in *Sat4j-PartialRoundingSat*.

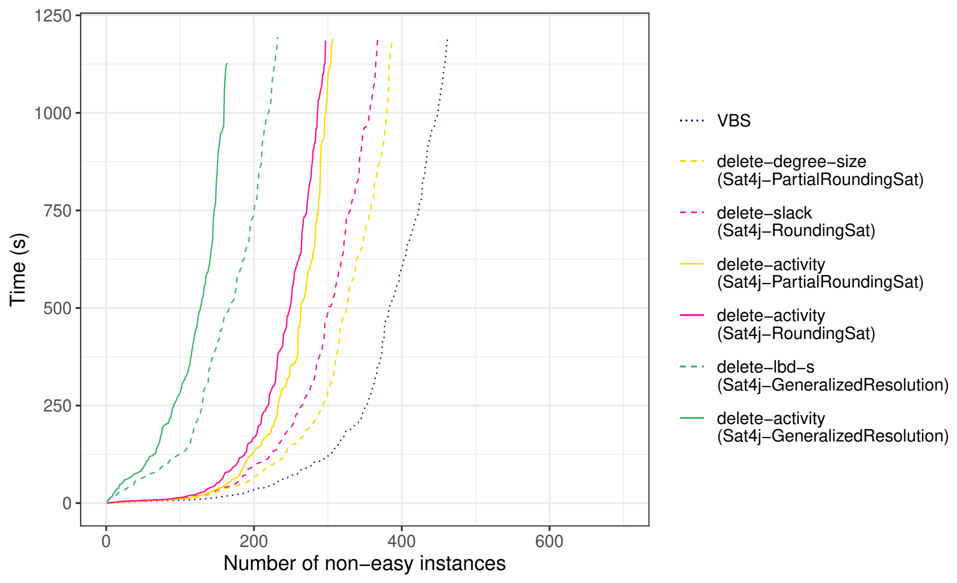


Figure 7.11: Cactus plot of the best learned constraint deletion strategies implemented for several configurations of *Sat4j*.

For the three configurations studied here, the performance shift between the strategies presented in this section is not really significant. However, a quite clear observation is that all of them perform better than the default *delete-activity* strategy. In particular, even the *no-deletion* strategy, which does not delete any constraint, performs better than this strategy, suggesting that the activity-based quality measure does not work well on the considered problems. Quite interestingly, for *RoundingSat*-based solvers, the size-based measures have better performance than the LBD-based measures, as also shown in Table 7.5. In practice, these quality measures do not actually bring any significant improvement in the speed of arithmetic operations, suggesting that this measure is good at evaluating the quality of the learned constraints.

To summarize the experiments of this section, Figure 7.11 shows, for each of the three considered configurations of *Sat4j*, the performance of the best learned constraint deletion (LCD) strategies that we identified in the previous experiments.

The cactus plot shows that each configuration can be improved using one of the new quality measures when performing constraint deletion, which is also confirmed by the results given in Table 7.5. However, the improvement brought by these strategies has less impact on the performance of the solver than that of the bumping strategies as studied before.

As observed in the cactus plots above, the number of instances solved with the learned constraint deletion strategies is quite similar for all solvers. This is true for almost all families, as shown in Tables 7.6, 7.7 and 7.8. In the case of *Sat4j-RoundingSat* and *Sat4j-PartialRoundingSat*, we can see that the *delete-slack* strategy particularly improves the performance of the solvers on the `nossum` and `SUMINEQ` families, while the *delete-degree* and *delete-degree-size* allow such an improvement for the `tsp` family.

LCD Strategy	Solved Instances	SOTA Contribution	1-second Contribution	10-second Contribution
delete-degree-size (Sat4j-PartialRoundingSat)	3935	26	124	98
delete-slack (Sat4j-RoundingSat)	3918	25	104	73
delete-activity (default) (Sat4j-PartialRoundingSat)	3855	8	49	23
delete-activity (default) (Sat4j-RoundingSat)	3843	4	37	16
delete-lbd-s (Sat4j-GeneralizedResolution)	3779	8	27	20
delete-activity (default) (Sat4j-GeneralizedResolution)	3711	3	17	7

Table 7.5: Table summarizing the results of the best quality measures used during learned constraint deletion in several configurations of *Sat4j*. Columns display, from left to right, the quality measure, the number of instances solved by the corresponding strategy, its state-of-the-art contribution, the number of instances for which the configuration was at least 1 second faster than all the others and the number of instances for which the configuration was at least 10 seconds faster than all the others

Family	Number of instances in the family	Number of solved easy instances	*delete-lbd-s*	*delete-lbd-a*	*delete-lbd-f*	*delete-lbd-e*	*delete-lbd-d*	*delete-degree*	*delete-degree-size*	*delete-slack*	*no-deletion*	*delete-activity*
`Aardal_1`	14	14	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)
`armies`	12	3	1 (121.71)	3 (778.23)	1 (771.18)	1 (747.47)	1 (256.6)	1 (98.84)	1 (92.03)	2 (758.78)	3 (2077.13)	1 (750.48)
`caixa`	1	1	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)
`d_n_k`	234	157	18 (6295.76)	18 (6918.91)	18 (7396.45)	18 (7461.39)	17 (6823.66)	15 (6184.2)	15 (6190.79)	12 (3783.15)	14 (5384.13)	12 (4271.03)
`d-equals-n_k`	70	35	7 (1646.1)	6 (1435.79)	6 (1396.56)	6 (1381.36)	6 (1320.47)	7 (1542.14)	6 (1440.55)	5 (423.51)	7 (2562.76)	5 (381.4)
`EC_ODD_GRIDS`	25	4	7 (117.13)	7 (108.43)	7 (111.87)	7 (111.17)	7 (134.21)	7 (148.69)	8 (930.87)	7 (169.21)	7 (136.9)	0 (0)
`EC_RANDOM_GRAPHS`	22	4	0 (0)	0 (0)	1 (519.77)	1 (488.65)	0 (0)	2 (190.65)	1 (561.43)	1 (668.45)	0 (0)	1 (234.96)
`FPGA_SAT05`	57	36	7 (2956.91)	4 (953.85)	7 (2131.55)	7 (2338.81)	7 (3103.74)	6 (1675.58)	6 (1300.17)	4 (438.31)	5 (308.79)	7 (1780.69)
`heinz`	4	1	1 (59.19)	1 (99.84)	1 (82.76)	1 (74.4)	1 (625.7)	1 (119.32)	0 (0)	1 (87.24)	1 (301.94)	0 (0)
`Instances3col_OPB`	26	8	0 (0)	0 (0)	1 (813.69)	1 (871.88)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)
`liu`	20	16	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)
`lopes`	193	0	1 (205.37)	1 (165.39)	2 (89.9)	2 (95.31)	2 (250.04)	1 (151.7)	2 (1357.17)	1 (199.86)	1 (227.32)	1 (256.01)
`nossum`	180	0	4 (951.63)	5 (993.17)	3 (1415.04)	3 (1409.3)	5 (2306.98)	8 (3695.9)	10 (4354.97)	4 (1258.32)	4 (1145.5)	2 (1567.69)
`oliveras`	4080	3060	106 (43321.51)	96 (38061.37)	100 (36890.9)	97 (33818.74)	99 (37621.8)	94 (37467.3)	92 (31912.5)	92 (35382.93)	85 (33310.27)	78 (26794.01)
`ppp-problems`	6	0	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)
`rand6reg`	33	6	2 (208.12)	2 (208.71)	2 (485.7)	2 (473.54)	2 (392.51)	3 (594.21)	1 (754.4)	2 (1020.14)	2 (915.44)	1 (203.76)
`robin`	6	2	1 (941.14)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)
`roussel`	40	20	2 (89.71)	2 (88.47)	2 (105.96)	2 (110.93)	2 (82.37)	2 (293.61)	2 (318.4)	2 (85.37)	2 (294.45)	0 (0)
`sroussel`	122	0	3 (316.05)	3 (331.09)	4 (670.94)	4 (623.07)	4 (545.5)	4 (579.73)	3 (316.89)	3 (315.42)	3 (313.66)	4 (853.18)
`subsetcard`	56	56	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)
`SUMINEQ`	24	1	0 (0)	0 (0)	0 (0)	0 (0)	1 (1047.92)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)
`tsp`	100	1	22 (11315.19)	22 (9443.91)	19 (7658.67)	19 (7656.51)	18 (6063.84)	19 (6796.92)	20 (5521.98)	19 (9690.89)	12 (3559.82)	13 (5733.91)
`uclid_pb_benchmarks`	50	31	6 (1605.38)	7 (2459.67)	7 (2604.86)	7 (2645.52)	7 (2736.3)	5 (2387.15)	5 (1388.44)	7 (3065.7)	5 (978.02)	3 (784.49)
`vertexcover-instances`	107	84	12 (2440.91)	13 (3878.36)	11 (2120.78)	11 (2216.41)	11 (1900.99)	11 (1636.54)	12 (1674.21)	12 (2060.39)	11 (2159.79)	4 (1577.05)
`wnqueen`	100	25	15 (4920.5)	15 (4879.66)	15 (4817.11)	15 (4958.76)	15 (4976.81)	15 (4883.61)	15 (5017.65)	15 (4841.71)	15 (4776.06)	15 (4841.28)

Table 7.6: Table summarizing, for each of the considered learned constraint deletion strategies, the number of non-easy instances solved by *Sat4j-GeneralizedResolution* when using this strategy for each of the considered families. The numbers in parentheses correspond to the total runtime (in seconds) needed to solve the instances.

Family	Number of instances in the family	Number of solved easy instances	*delete-slack*	*delete-degree-size*	*delete-degree*	*delete-lbd-d*	*delete-lbd-s*	*delete-lbd-e*	*delete-lbd-f*	*delete-lbd-a*	*no-deletion*	*delete-activity*
`Aardal_1`	14	14	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)
`armies`	12	4	1 (56.91)	1 (61.27)	2 (256.34)	2 (259.15)	2 (127.86)	1 (57.02)	1 (53.79)	2 (600.89)	2 (164.84)	1 (54.76)
`caixa`	1	1	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)
`d_n_k`	234	159	18 (6623.05)	21 (8687.29)	21 (8132.56)	21 (7328.47)	21 (7519.46)	21 (8156.64)	21 (8125.78)	21 (7400.61)	18 (8974.26)	14 (6037.25)
`d-equals-n_k`	70	36	6 (2034.19)	5 (1313.42)	6 (1270.2)	6 (2047.09)	4 (851.04)	5 (1891.73)	5 (1917.61)	5 (2092.56)	6 (2187.61)	6 (3096.3)
`EC_ODD_GRIDS`	25	5	7 (282.87)	7 (321.29)	7 (645.59)	7 (191.54)	7 (399.29)	7 (488.51)	7 (488.74)	7 (313.86)	7 (559.74)	2 (641)
`EC_RANDOM_GRAPHS`	22	6	6 (1267.2)	7 (2470.81)	5 (825.24)	4 (467.5)	3 (417.28)	3 (993.72)	3 (999.49)	3 (830.88)	2 (1039.23)	2 (893.78)
`FPGA_SAT05`	57	37	6 (3271.15)	6 (772.35)	4 (1201.99)	5 (1156.53)	6 (2556.92)	5 (2103.31)	5 (1965.49)	5 (842.52)	5 (1571.04)	4 (2206.69)
`heinz`	4	1	1 (100.45)	1 (106.95)	1 (631.72)	1 (748.16)	0 (0)	0 (0)	0 (0)	1 (52.05)	0 (0)	0 (0)
`Instances3col_OPB`	26	8	2 (1468.05)	0 (0)	1 (921.49)	1 (759.35)	1 (712.53)	1 (956.24)	1 (977.85)	1 (723.64)	1 (961.44)	0 (0)
`liu`	20	16	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)
`lopes`	193	0	3 (587.4)	3 (535.85)	3 (578.6)	3 (618.59)	3 (591.26)	3 (776)	3 (848.41)	3 (829.79)	3 (575.29)	2 (1210.13)
`nossum`	180	3	**14 (7586.05)**	9 (3430.31)	6 (2456.01)	13 (7288.31)	9 (2572.05)	5 (2465.26)	5 (2433.79)	4 (2049.99)	9 (3839.22)	7 (3353.44)
`oliveras`	4080	3077	109 (43683.71)	105 (36726.47)	112 (43756.88)	109 (39191.79)	114 (41526.67)	115 (40787.05)	117 (41650.06)	118 (44734.24)	110 (43914.46)	91 (33994.67)
`ppp-problems`	6	0	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)
`rand6reg`	33	10	1 (15.52)	1 (1111.41)	0 (0)	2 (173.56)	2 (868.38)	2 (231)	2 (240.26)	2 (1206.99)	2 (83.28)	1 (17.68)
`robin`	6	2	1 (384.61)	1 (1157.23)	1 (528.43)	1 (1189.44)	1 (500.4)	1 (235.26)	1 (227.41)	0 (0)	1 (153.39)	0 (0)
`roussel`	40	20	2 (81.04)	2 (295.37)	2 (296.67)	2 (83.22)	2 (84.05)	2 (97.73)	2 (107.47)	2 (86.44)	2 (284.95)	0 (0)
`sroussel`	122	0	1 (227.52)	1 (378.6)	6 (2928.54)	3 (1178.33)	2 (377.13)	2 (556.94)	2 (543.24)	1 (90.49)	1 (92.03)	2 (326.76)
`subsetcard`	56	56	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)
`SUMINEQ`	24	1	**14 (2054.03)**	**12 (940.97)**	8 (1876.38)	8 (420.67)	6 (640.95)	3 (177.78)	3 (191.17)	3 (337.45)	3 (107.09)	2 (843.71)
`tsp`	100	8	18 (7264.84)	**28 (9770.74)**	**24 (8740.88)**	**22 (7893.11)**	17 (4808.65)	14 (5855.8)	14 (6015.16)	13 (4291.77)	9 (2992.37)	6 (2065.73)
`uclid_pb_benchmarks`	50	33	7 (2408.85)	7 (1621.16)	7 (1667.34)	6 (1655.5)	6 (1901.72)	6 (2191.74)	6 (2200.19)	6 (2235.44)	6 (2411.15)	6 (2180.67)
`vertexcover-instances`	107	93	11 (2225.2)	11 (2411.71)	11 (2600.06)	12 (3462.08)	11 (2482.88)	11 (2360.55)	11 (2230.41)	11 (2590.9)	10 (3379.93)	9 (2967.86)
`wnqueen`	100	99	1 (70.31)	1 (112.69)	1 (152.03)	1 (83.15)	1 (74.7)	1 (82.3)	1 (101.93)	1 (82.3)	1 (79.02)	1 (79.79)

Table 7.7: Table summarizing, for each of the considered learned constraint deletion strategies, the number of non-easy instances solved by *Sat4j-RoundingSat*when using this strategy for each of the considered families. The numbers in parentheses correspond to the total runtime (in seconds) needed to solve the instances. Bold numbers highlight strategies that perform well on a given family compared to the others.

Family	Number of instances in the family	Number of solved easy instances	*delete-degree-size*	*delete-degree*	*delete-slack*	*delete-lbd-s*	*delete-lbd-f*	*delete-lbd-d*	*delete-lbd-a*	*delete-lbd-e*	*no-deletion*	*delete-activity*
`Aardal_1`	14	14	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)
`armies`	12	5	1 (197.77)	0 (0)	1 (321.54)	3 (1836.38)	1 (105.86)	1 (518.26)	1 (256.31)	1 (102.13)	1 (108)	1 (141.8)
`caixa`	1	1	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)
`d_n_k`	234	160	19 (8074.48)	19 (7972.77)	19 (8431.84)	20 (8098.71)	20 (8718.21)	21 (8368.51)	20 (7917.45)	20 (8677.05)	16 (7579.37)	12 (4920.11)
`d-equals-n_k`	70	37	4 (839.17)	5 (1010.23)	4 (1037.59)	4 (808.81)	5 (906.8)	4 (1105.44)	4 (847.69)	5 (872.32)	4 (1241.37)	4 (1170.77)
`EC_ODD_GRIDS`	25	5	7 (894.35)	7 (396.4)	7 (563.15)	6 (62.63)	6 (73.99)	6 (77.13)	6 (79.56)	6 (71.33)	6 (100.87)	1 (91.77)
`EC_RANDOM_GRAPHS`	22	7	4 (1685.78)	5 (529.84)	5 (1372.46)	2 (1271.9)	2 (724.81)	2 (802.3)	1 (12.97)	2 (732.66)	2 (990.1)	1 (20.83)
`FPGA_SAT05`	57	38	5 (1149.78)	4 (1418.29)	5 (2352.06)	5 (712.45)	5 (1709)	4 (838.17)	3 (225.37)	5 (1640.56)	3 (446.11)	3 (572.67)
`heinz`	4	1	0 (0)	1 (766.56)	1 (96.26)	1 (566.52)	1 (627.23)	1 (248.33)	1 (80.71)	1 (594.22)	1 (76.61)	0 (0)
`Instances3col_OPB`	26	8	0 (0)	1 (770.72)	1 (983.21)	0 (0)	1 (649.89)	1 (759.64)	0 (0)	1 (572.55)	1 (976.44)	0 (0)
`liu`	20	16	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)
`lopes`	193	0	3 (1006.71)	3 (498.49)	3 (878.76)	3 (668.71)	3 (1136.45)	3 (771.32)	3 (760.26)	3 (1144.47)	3 (1393.52)	2 (1422.57)
`nossum`	180	0	9 (3373.64)	10 (5565.24)	**15 (5600.36)**	10 (4424.4)	12 (6414.11)	9 (3455.46)	10 (3939.39)	12 (6419.2)	8 (2005.93)	6 (3932.72)
`oliveras`	4080	3077	110 (40658.75)	100 (34261.33)	100 (38644.36)	113 (40206.34)	108 (35969.66)	108 (36817.84)	114 (43782.5)	108 (36521.12)	102 (40135.62)	85 (31863.74)
`ppp-problems`	6	0	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)
`rand6reg`	33	10	0 (0)	0 (0)	1 (819.55)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)
`robin`	6	2	0 (0)	1 (575.84)	1 (295.05)	1 (260.35)	0 (0)	1 (642.08)	0 (0)	0 (0)	0 (0)	1 (1190.61)
`roussel`	40	20	2 (325.81)	2 (299.73)	2 (76.93)	2 (84.11)	2 (99.85)	2 (83.54)	2 (88.66)	2 (102.25)	2 (288.34)	0 (0)
`sroussel`	122	1	1 (651.76)	2 (1224.25)	2 (1393.42)	1 (588.52)	2 (368.83)	2 (716.71)	0 (0)	2 (373.59)	2 (903.59)	1 (593.29)
`subsetcard`	56	56	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)
`SUMINEQ`	24	2	**11 (1354.42)**	9 (1535.89)	**12 (1254.37)**	7 (2438.3)	4 (2144.3)	9 (1021.84)	3 (263.02)	4 (2103.87)	3 (380.5)	1 (995.12)
`tsp`	100	21	**34 (10598.94)**	**36 (12086.48)**	24 (5572.63)	24 (8818.01)	26 (12054.4)	23 (6573.69)	29 (12303.64)	26 (12002.15)	17 (3755.73)	14 (2600.12)
`uclid_pb_benchmarks`	50	34	5 (1235.01)	6 (2205.86)	5 (760.6)	4 (543.37)	5 (1642.87)	5 (2100.95)	5 (1475.85)	5 (1532.46)	5 (1832.99)	5 (2159.07)
`vertexcover-instances`	107	91	14 (2596.07)	14 (2092.61)	14 (2563.37)	14 (2633.58)	14 (2331.19)	14 (2402.69)	14 (2332.09)	14 (2369.2)	13 (2500.73)	12 (2493.08)
`wnqueen`	100	99	1 (49.58)	1 (72.11)	1 (54.27)	1 (49.23)	1 (50.78)	1 (46.58)	1 (50.36)	1 (51.95)	1 (48.74)	1 (66.94)

Table 7.8: Table summarizing, for each of the considered learned constraint deletion strategies, the number of non-easy instances solved by *Sat4j-PartialRoundingSat* when using this strategy for each of the considered families. The numbers in parentheses correspond to the total runtime (in seconds) needed to solve the instances. Bold numbers highlight strategies that perform well on a given family compared to the others.

7.2.3 Application to Restarts

Another possible use of the quality measures described above is to design a restart policy similar to that of *Glucose* [AS18]: when the quality of the most recently learned pseudo-Boolean constraints decreases, a restart should be performed (see Subsection 4.1.3). We thus define the following strategies:

- *restart-slack*, based on the *slack* measure,
- *restart-degree*, based on the *degree* measure,
- *restart-degree-size*, based on the *degree-size* measure,
- *restart-lbd-a*, based on the LBD_a measure,
- *restart-lbd-s*, based on the LBD_s measure,
- *restart-lbd-d*, based on the LBD_d measure,
- *restart-lbd-f*, based on the LBD_f measure, and
- *restart-lbd-e*, based on the LBD_e measure.

Let us study the practical impact of these strategies in different configurations of the pseudo-Boolean solver *Sat4j*, with the same experimental setting as before. In *Sat4j*, using restart strategies based on the quality of learned constraints is tightly linked to the learned constraint deletion strategy, especially when considering LBD-based restarts. As such, to evaluate the impact of the restarts strategies, we had to deactivate learned constraint deletion in the experiments. The results of these experiments are given in Figures 7.12, 7.13 and 7.14.

From these results, we can see that the adaptive restarts based on the new quality measure do not perform well compared to restart schemes implemented in SAT solvers. This is particularly true for *RoundingSat*-based solvers, in which *Picosat*'s restart policy [Bie08b] has the best performance. This suggests that the quality measures defined above are not suitable for adaptive restarts, even though the degree-based measure has quite good performance in *Sat4j-GeneralizedResolution*. In practice, it is worth noting that the degree of the constraints learned by this configuration may become very large, while that of the constraints derived in *RoundingSat* based solvers remains relatively small because of the application of the division rule, which may explain why degree-based strategies do not perform well in these solvers.

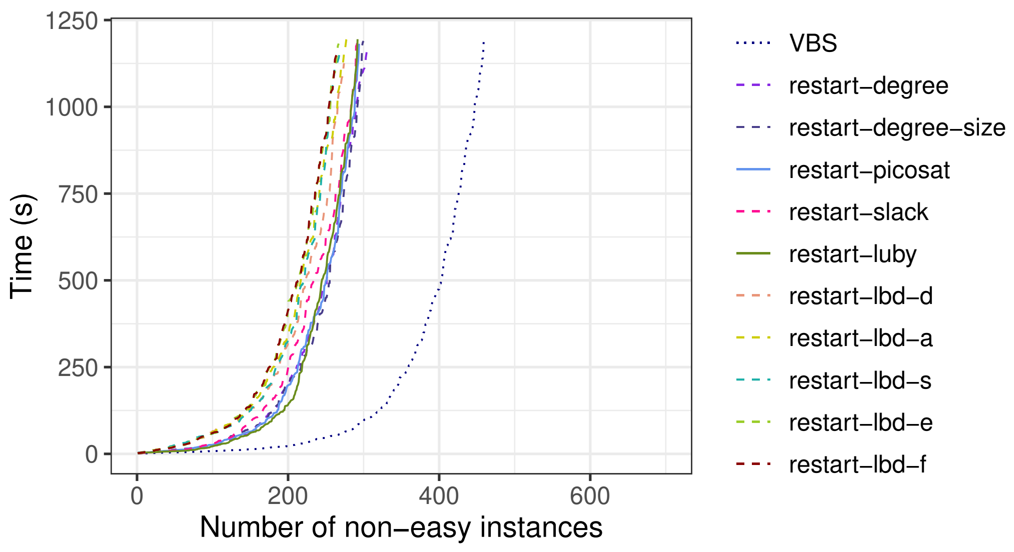


Figure 7.12: Cactus plot of the various restart strategies implemented in *Sat4j-GeneralizedResolution*

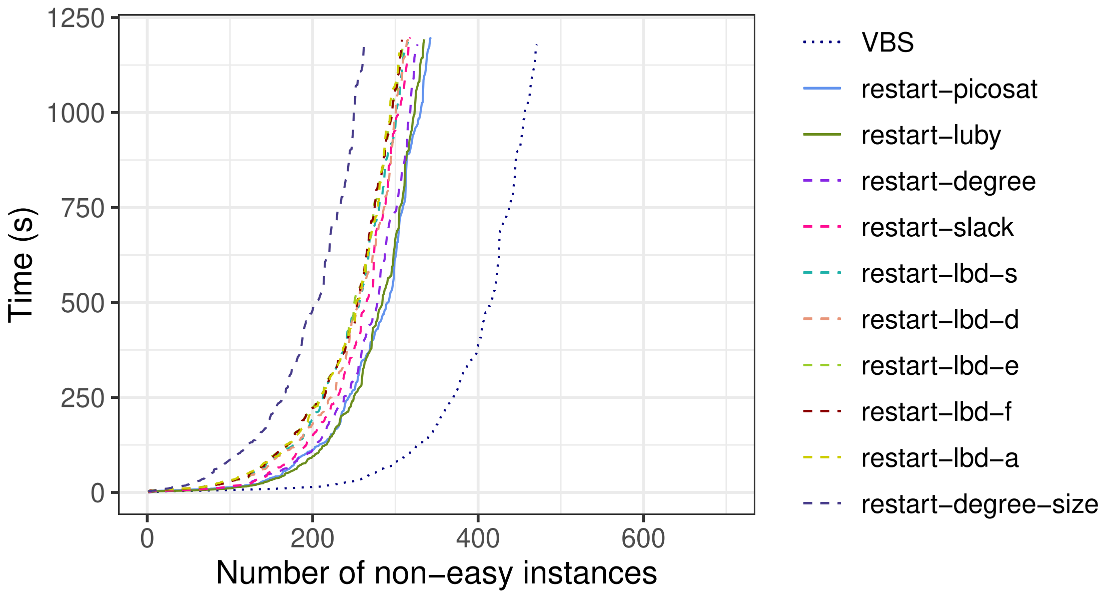


Figure 7.13: Cactus plot of the various restart strategies implemented in *Sat4j-RoundingSat*.

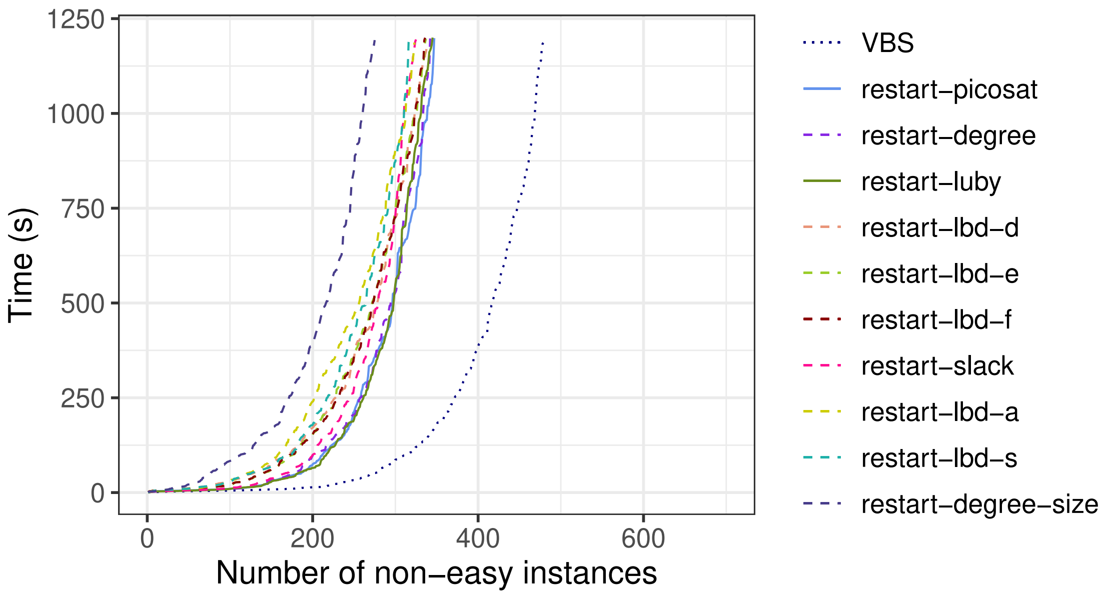


Figure 7.14: Cactus plot of the various restart strategies implemented in *Sat4j-PartialRoundingSat*.

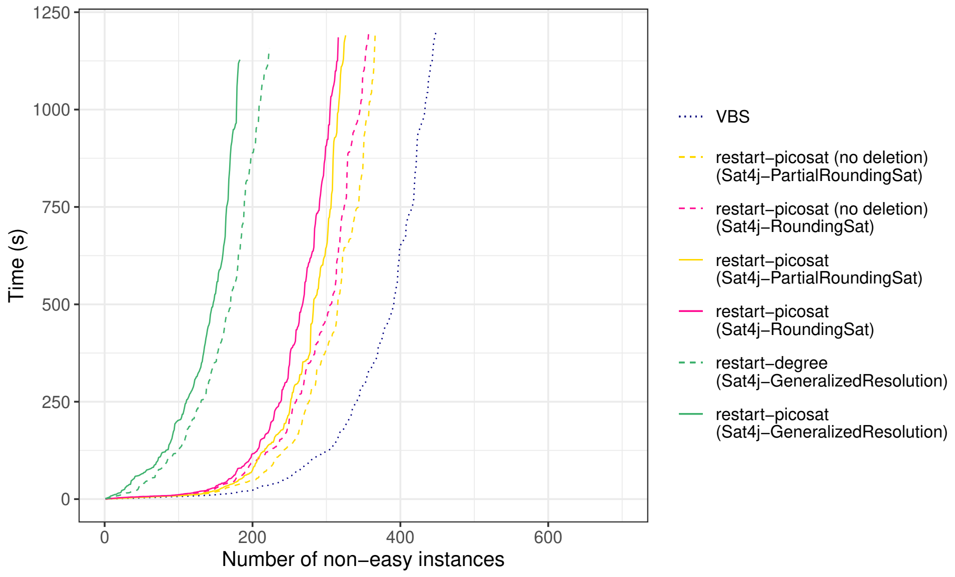


Figure 7.15: Cactus plot of the best restart strategies implemented in several configurations of *Sat4j*.

The cactus plot given by Figure 7.15 shows that the improvements brought to the strategies do not have a big impact in the configurations of *Sat4j* (especially compared to the improvements we observed in previous experiments). This is also confirmed by the results of Table 7.9. Note that, in both the cactus plot and in the table, the strategies are compared with the default configuration of *Sat4j*, which uses the *activity*-based deletion strategy. As such, the small improvement we observe may also be due to the bad results of this strategy.

As for learned constraint deletion strategies, and as suggested by the cactus plots, the number of instances solved with the learned constraint deletion strategies is quite similar for all solvers. This is true for almost all families, as shown in Tables 7.10, 7.11 and 7.12. In the case of *Sat4j-RoundingSat* and *Sat4j-PartialRoundingSat*, we can see that the *delete-slack* strategy particularly improves the performance of the solvers on the `nossum` and `SUMINEQ` families, while the *delete-degree* and *delete-degree-size* allow such an improvement for the `tsp` family.

Restart Strategy	Solved Instances	SOTA Contribution	1-second Contribution	10-second Contribution
restart-picosat (no deletion) (Sat4j-PartialRoundingSat)	3895	15	92	61
restart-picosat (no deletion) (Sat4j-RoundingSat)	3884	23	106	75
restart-picosat (default) (Sat4j-PartialRoundingSat)	3855	13	43	24
restart-picosat (default) (Sat4j-RoundingSat)	3843	7	30	17
restart-degree (Sat4j-GeneralizedResolution)	3751	7	45	20
restart-picosat (default) (Sat4j-GeneralizedResolution)	3711	4	33	16

Table 7.9: Table summarizing the results of the best quality measures used by a Glucose-like restart policy in different configurations of *Sat4j*. Columns display, from left to right, the quality measure, the number of instances solved by the corresponding strategy, its state-of-the-art contribution, the number of instances for which the configuration was at least 1 second faster than all the others and the number of instances for which the configuration was at least 10 seconds faster than all the others

Family	Number of instances in the family	Number of solved easy instances	*restart-degree*	*restart-degree-size*	*restart-picosat*	*restart-slack*	*restart-luby*	*restart-lbd-d*	*restart-lbd-a*	*restart-lbd-s*	*restart-lbd-e*	*restart-lbd-f*
Aardal_1	14	14	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)
armies	12	2	5 (1190.02)	5 (1077.58)	4 (2088)	4 (478.03)	6 (1245.82)	3 (405.08)	4 (1073.66)	3 (448.46)	3 (241.17)	3 (240.59)
caixa	1	1	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)
d_n_k	234	149	24 (9589.52)	22 (5826.93)	22 (5529.05)	22 (6277.26)	22 (6865.77)	20 (7494.58)	18 (6630.49)	18 (6869.11)	20 (8230.27)	20 (8084.05)
d-equals-n_k	70	16	**24 (2449.84)**	**25 (2698.79)**	**26 (2811.04)**	**24 (1021.86)**	**24 (1717.33)**	11 (2054.44)	11 (2350.36)	12 (3126.93)	11 (2583.68)	11 (2631.42)
EC_ODD_GRIDS	25	5	6 (608.83)	6 (113.72)	6 (120.33)	6 (87.84)	6 (121.72)	6 (206.05)	6 (134.77)	6 (148.01)	7 (1890.81)	7 (1887.06)
EC_RANDOM_GRAPHS	22	4	0 (0)	1 (117.79)	0 (0)	1 (556.57)	1 (154.44)	1 (117.47)	2 (922.63)	2 (706.92)	1 (57.46)	1 (53.5)
FPGA_SAT05	57	33	7 (1618.47)	10 (366.26)	8 (321.9)	8 (895.28)	**13 (1381.3)**	9 (1851.56)	10 (991.3)	9 (865.3)	6 (160.87)	6 (158.98)
heinz	4	1	1 (10.84)	0 (0)	1 (301.94)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	1 (1108.74)	1 (1078.01)
Instances3col_OPB	26	7	1 (33.03)	1 (59.38)	1 (42.35)	1 (61.32)	1 (26.52)	3 (1536.25)	3 (1594.68)	2 (483.04)	3 (1619.74)	3 (1651.83)
liu	20	16	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)
lopes	193	0	2 (1321.23)	2 (580.58)	1 (227.32)	1 (159.89)	3 (2133.32)	2 (410.37)	3 (1884.22)	2 (918.35)	1 (186.1)	1 (194.63)
nossum	180	0	6 (1327.82)	5 (1155.88)	4 (1145.5)	0 (0)	5 (762.9)	2 (589.16)	2 (647.13)	3 (382.8)	1 (866.29)	1 (894.58)
oliveras	4080	3004	145 (36555.35)	135 (33338.84)	141 (34608.64)	133 (37643.1)	137 (33252.11)	139 (36141.08)	137 (43601.27)	140 (38745.11)	144 (39593.33)	142 (37519.52)
ppp-problems	6	0	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)
rand6reg	33	6	4 (1281.47)	3 (1018.5)	2 (915.44)	4 (810.43)	3 (561.82)	3 (978.15)	4 (1965.2)	3 (875.37)	3 (636.25)	3 (720.96)
robin	6	1	1 (15.43)	1 (127.4)	1 (8.36)	1 (18.51)	1 (22.97)	2 (713.35)	2 (797.84)	1 (13.2)	1 (28.78)	1 (29.13)
roussel	40	20	1 (188.53)	1 (194.73)	2 (294.45)	2 (494.68)	2 (89.95)	2 (195.29)	2 (190.38)	2 (195.26)	2 (115.38)	2 (134.87)
sroussel	122	0	2 (696.85)	1 (220.65)	3 (313.66)	0 (0)	0 (0)	0 (0)	1 (858.58)	0 (0)	0 (0)	0 (0)
subsetcard	56	56	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)
SUMINEQ	24	0	0 (0)	0 (0)	1 (5.63)	1 (193.77)	0 (0)	1 (21.11)	0 (0)	1 (5.27)	0 (0)	0 (0)
tsp	100	0	12 (4318.31)	**22 (8365.58)**	13 (3580.05)	**20 (6806.38)**	9 (2855.84)	12 (3101.69)	14 (2864.07)	9 (4264.29)	13 (3366.47)	13 (3536.75)
uclid_pb_benchmarks	50	30	7 (2296.79)	6 (1100.75)	6 (995.98)	6 (1722)	5 (575.45)	7 (2569.6)	7 (1985.37)	7 (1780.66)	8 (2769.15)	8 (2825.64)
vertexcover-instances	107	78	14 (1778.81)	13 (1400.14)	17 (2200.78)	15 (2744)	23 (2407.8)	14 (1235.13)	13 (2335.97)	14 (2943.88)	12 (1487.49)	12 (1485.88)
wnqueen	100	5	42 (5729.7)	41 (6245.77)	35 (4984.75)	43 (9087.39)	31 (3671.74)	37 (5411.42)	39 (5806.1)	36 (4364.58)	30 (4984.77)	30 (5017.79)

Table 7.10: Table summarizing, for each of the considered restart policies, the number of non-easy instances solved by *Sat4j-GeneralizedResolution* when using this strategy for each of the considered families. The numbers in parentheses correspond to the total runtime (in seconds) needed to solve the instances. Bold numbers highlight strategies that perform well on a given family compared to the others.

Family	Number of instances in the family	Number of solved easy instances	*restart-picosat*	*restart-luby*	*restart-degree*	*restart-slack*	*restart-lbd-s*	*restart-lbd-d*	*restart-lbd-e*	*restart-lbd-f*	*restart-lbd-a*	*restart-degree-size*
`Aardal_1`	14	13	1 (1.31)	1 (14.43)	1 (965.25)	1 (2.02)	1 (5.52)	1 (6.47)	1 (5.51)	1 (5.89)	1 (6.08)	1 (235.4)
`armies`	12	0	6 (188.93)	7 (1254.85)	5 (1258.86)	6 (1927.46)	6 (2263.31)	5 (777.65)	6 (1543.13)	6 (1538.48)	7 (604.45)	3 (567.21)
`caixa`	1	1	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)
`d_n_k`	234	150	27 (9152.74)	24 (6284.6)	25 (7714.34)	23 (5634.34)	28 (13534.35)	24 (9733.4)	22 (8433.22)	22 (8469.72)	24 (8975.4)	25 (6415.14)
`d-equals-n_k`	70	14	**28 (2426.83)**	**27 (2887.5)**	**28 (1597.9)**	**25 (849.69)**	18 (3721.12)	17 (3489.41)	17 (6498.51)	16 (5410.36)	17 (5252.65)	26 (3110.5)
`EC_ODD_GRIDS`	25	10	2 (501.24)	2 (162.39)	2 (727.01)	2 (994.25)	2 (372.86)	2 (772.49)	2 (437.59)	2 (439.05)	2 (264.21)	2 (837.64)
`EC_RANDOM_GRAPHS`	22	4	4 (1051.09)	6 (771.28)	6 (97.86)	3 (887.06)	4 (710.89)	1 (12.29)	4 (2141.71)	4 (2156.41)	2 (93.35)	5 (2181.75)
`FPGA_SAT05`	57	36	6 (1578.26)	10 (366.83)	7 (1176.06)	7 (2376.58)	7 (1534.01)	6 (485.28)	7 (1356.3)	7 (1442.05)	8 (813.82)	7 (2744.7)
`heinz`	4	1	0 (0)	0 (0)	1 (861.89)	1 (83.65)	1 (458.93)	1 (118.28)	0 (0)	0 (0)	0 (0)	0 (0)
`Instances3col_OPB`	26	7	2 (978.46)	1 (27.9)	1 (22.54)	1 (60.83)	3 (885.46)	2 (558.47)	5 (2452.9)	5 (2510.51)	3 (763.82)	2 (495.09)
`liu`	20	15	1 (7.42)	1 (5.35)	1 (7.32)	1 (6.87)	1 (6.01)	1 (6.84)	2 (346.96)	2 (337.93)	1 (7.1)	2 (931.25)
`lopes`	193	0	3 (575.29)	3 (1136.5)	5 (1298.64)	4 (2251.29)	4 (1866.54)	4 (1666.37)	4 (1919.4)	4 (1916.03)	3 (1670.97)	2 (657.66)
`nossum`	180	0	**12 (3864.58)**	5 (3142.27)	**13 (3977.29)**	**12 (5555.82)**	4 (1623.52)	5 (2966.38)	1 (29.36)	1 (36.81)	1 (67.47)	**11 (2910.48)**
`oliveras`	4080	2992	195 (45032.02)	181 (39035.23)	171 (33919.19)	180 (35693.21)	179 (39246.93)	189 (43851.84)	180 (37052.66)	178 (34476.16)	176 (38301.34)	126 (43531.78)
`ppp-problems`	6	0	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)
`rand6reg`	33	10	2 (83.28)	1 (5.98)	2 (429.72)	2 (1802.97)	2 (109.56)	1 (2.1)	2 (1049.95)	2 (1002.4)	2 (579.74)	2 (52.71)
`robin`	6	2	1 (153.39)	2 (1234.56)	1 (992.49)	1 (64.45)	1 (317.98)	1 (29.91)	1 (316.13)	1 (291.51)	0 (0)	1 (518.71)
`roussel`	40	20	2 (284.95)	2 (83.94)	1 (191.22)	2 (485.66)	2 (184.19)	2 (187.35)	2 (126.47)	2 (116.18)	2 (195.43)	1 (193.46)
`sroussel`	122	0	1 (92.03)	3 (584.89)	3 (92.44)	0 (0)	1 (86.37)	2 (1359.11)	0 (0)	0 (0)	0 (0)	2 (1496.99)
`subsetcard`	56	56	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)
`SUMINEQ`	24	0	4 (110.95)	8 (1517.52)	2 (1167.71)	3 (250.86)	4 (667.28)	2 (21.74)	4 (1397.61)	4 (1309.18)	5 (1333.63)	3 (667.52)
`tsp`	100	0	17 (3075.43)	17 (3646.37)	24 (6744.79)	17 (6353.31)	20 (5722.8)	22 (4909.6)	25 (6885.91)	26 (7953.31)	24 (5760.75)	20 (2812.72)
`uclid_pb_benchmarks`	50	32	7 (2424.94)	9 (2140.72)	8 (2342.35)	8 (1209.78)	7 (1823.65)	8 (1370.86)	7 (2179.01)	7 (2113.91)	7 (2143.12)	7 (1971.55)
`vertexcover-instances`	107	84	19 (3551.98)	22 (2732.46)	17 (1719.78)	16 (3832.95)	17 (3121.28)	17 (3357.13)	15 (2575.63)	15 (2640.66)	17 (3614.85)	12 (2689.51)
`wnqueen`	100	97	3 (105.94)	3 (75.42)	3 (170.45)	3 (259.84)	3 (137.46)	3 (114.92)	3 (66.31)	3 (63.82)	3 (265.77)	3 (428)

Table 7.11: Table summarizing, for each of the considered restart policies, the number of non-easy instances solved by *Sat4j-RoundingSat* when using this strategy for each of the considered families. The numbers in parentheses correspond to the total runtime (in seconds) needed to solve the instances. Bold numbers highlight strategies that perform well on a given family compared to the others.

Family	Number of instances in the family	Number of solved easy instances	*restart-picosat*	*restart-degree*	*restart-luby*	*restart-lbd-d*	*restart-lbd-e*	*restart-lbd-f*	*restart-slack*	*restart-lbd-a*	*restart-lbd-s*	*restart-degree-size*
`Aardal_1`	14	14	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)
`armies`	12	2	4 (144.17)	5 (702.75)	4 (2509.26)	4 (405.93)	3 (1297.83)	3 (1354.7)	4 (41.76)	5 (1538.98)	4 (576.29)	0 (0)
`caixa`	1	1	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)
`d_n_k`	234	151	25 (7728.36)	24 (7443.92)	23 (7052.38)	24 (9485.62)	25 (10681.28)	25 (10568.07)	24 (7049.08)	25 (12207.77)	22 (8020.47)	21 (7027.32)
`d-equals-n_k`	70	14	27 (1496.51)	27 (1699.49)	26 (1748.97)	19 (3323.93)	22 (6329.65)	22 (6243.78)	25 (975.74)	19 (4229.1)	18 (3274.73)	27 (5006.96)
`EC_ODD_GRIDS`	25	10	1 (48.93)	2 (566.54)	2 (290.06)	2 (1067.91)	1 (14.69)	1 (13.47)	3 (1256.7)	2 (1068.75)	1 (49.13)	2 (211.54)
`EC_RANDOM_GRAPHS`	22	4	5 (1005.79)	5 (1189.63)	3 (834.62)	3 (708.46)	2 (232.55)	2 (222.13)	5 (177.72)	4 (1002.29)	3 (1028.55)	3 (630.43)
`FPGA_SAT05`	57	36	5 (423.62)	8 (1323.78)	11 (337.52)	5 (596.36)	8 (2907.72)	8 (2828.61)	7 (464.77)	7 (2875.52)	6 (634.3)	6 (841.18)
`heinz`	4	1	1 (76.61)	0 (0)	1 (40.81)	1 (992.17)	1 (46.41)	1 (47.44)	0 (0)	1 (67.76)	0 (0)	0 (0)
`Instances3col_OPB`	26	7	2 (998.36)	1 (38.32)	1 (38.87)	3 (1236.17)	4 (1865.11)	4 (1793.14)	1 (51.12)	3 (1112.95)	4 (1835.43)	3 (1746.54)
`liu`	20	14	2 (12.73)	2 (14.65)	2 (11.03)	2 (13.21)	2 (12.24)	2 (12.15)	2 (12.38)	2 (14.81)	2 (15.61)	1 (184.85)
`lopes`	193	0	3 (1393.52)	4 (2098.15)	5 (2849.31)	5 (3598.25)	5 (1875.5)	5 (1853.81)	5 (2917.81)	4 (2236.48)	4 (1606.2)	3 (1092.63)
`nossum`	180	0	8 (2005.93)	9 (1832.67)	8 (3234.8)	3 (763.2)	4 (915.29)	4 (857.63)	8 (3229.79)	4 (1464.7)	2 (293.95)	10 (2500.33)
`oliveras`	4080	2986	193 (41328.86)	175 (32930.24)	180 (34599.68)	197 (45835.3)	186 (39292.45)	186 (38704.46)	178 (35437.46)	178 (38413.45)	183 (42692.63)	132 (46393.33)
`ppp-problems`	6	0	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)
`rand6reg`	33	10	0 (0)	3 (100.03)	2 (43.3)	1 (156.39)	1 (1196.84)	0 (0)	0 (0)	2 (1696.03)	0 (0)	1 (573.61)
`robin`	6	2	0 (0)	1 (824.52)	1 (465.3)	1 (638.91)	1 (169.89)	1 (143.7)	1 (210.4)	1 (8.32)	1 (7.28)	1 (139.58)
`roussel`	40	20	2 (288.34)	1 (200.53)	2 (94.61)	2 (209.6)	2 (115.4)	2 (115.31)	2 (503.86)	2 (210.04)	2 (189.64)	1 (212.83)
`sroussel`	122	0	3 (940.39)	2 (1262.36)	1 (574.65)	1 (841.1)	2 (539.85)	2 (518.95)	2 (1126.12)	1 (448.64)	1 (163.04)	1 (107.35)
`subsetcard`	56	56	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)
`SUMINEQ`	24	0	5 (390.26)	2 (479.32)	4 (1088.91)	5 (1304.03)	4 (170.05)	4 (182.26)	1 (32.53)	5 (2230.24)	4 (87.57)	4 (554.56)
`tsp`	100	7	31 (3845.08)	**45 (8917.87)**	36 (6824.19)	33 (8097.27)	37 (9980.48)	37 (9979.12)	32 (4514.26)	32 (9429.98)	33 (7350.12)	37 (5896.78)
`uclid_pb_benchmarks`	50	32	7 (1862.18)	8 (1364.98)	7 (2132.12)	9 (1894.85)	9 (1680.39)	9 (1673.87)	7 (2755.25)	7 (1591.28)	8 (1663.24)	7 (1728.53)
`vertexcover-instances`	107	86	18 (2527.99)	13 (1572.68)	**21 (2069.05)**	14 (3829.65)	13 (3334.64)	13 (3438.54)	13 (1668.57)	14 (4031.91)	14 (2877.89)	10 (2919.75)
`wnqueen`	100	95	5 (164.33)	5 (190.59)	5 (67.81)	5 (334.41)	5 (220.75)	5 (212.26)	5 (373.24)	5 (264.62)	5 (269.7)	5 (167.73)

Table 7.12: Table summarizing, for each of the considered restart policies, the number of non-easy instances solved by *Sat4j-PartialRoundingSat* when using this strategy for each of the considered families. The numbers in parentheses correspond to the total runtime (in seconds) needed to solve the instances. Bold numbers highlight strategies that perform well on a given family compared to the others.

7.3 Putting Things Together

The previous sections showed that considering the specificities of pseudo-Boolean constraints in the branching heuristic and when evaluating the quality of learned constraints may improve the performance of the solver. Let us now consider combinations of these strategies.

7.3.1 Combining Learned Constraint Deletion and Restarts

As we mentioned above, in *Sat4j*, there is a tight link between the learned constraint deletion strategy and the restart policy, since they both use the same quality measures for learned constraints. Let us thus consider their combined use in this solver. For the strategies introduced in this section, both the restart and the learned constraint deletion strategies use the same measure. The results of these experiments are given in Figures 7.16, 7.17 and 7.18.

In these cactus plots, we observe that the different quality measures, despite improving the default configuration, do not have a strong impact on the solver (especially compared to that of the learned constraint deletion), as also illustrated in Figure 7.19.

The cactus plot and the results in Table 7.13 tend to confirm the observation made in the previous section regarding the impact of the quality measures in the restart policy: they do not work well for detecting when to trigger restarts. Indeed, even though all strategies improve the performance of the different solvers, the improvement does not reach the one achieved by the learned constraint deletion. This suggests that the improvement is mainly due to this strategy rather than the restart policy.

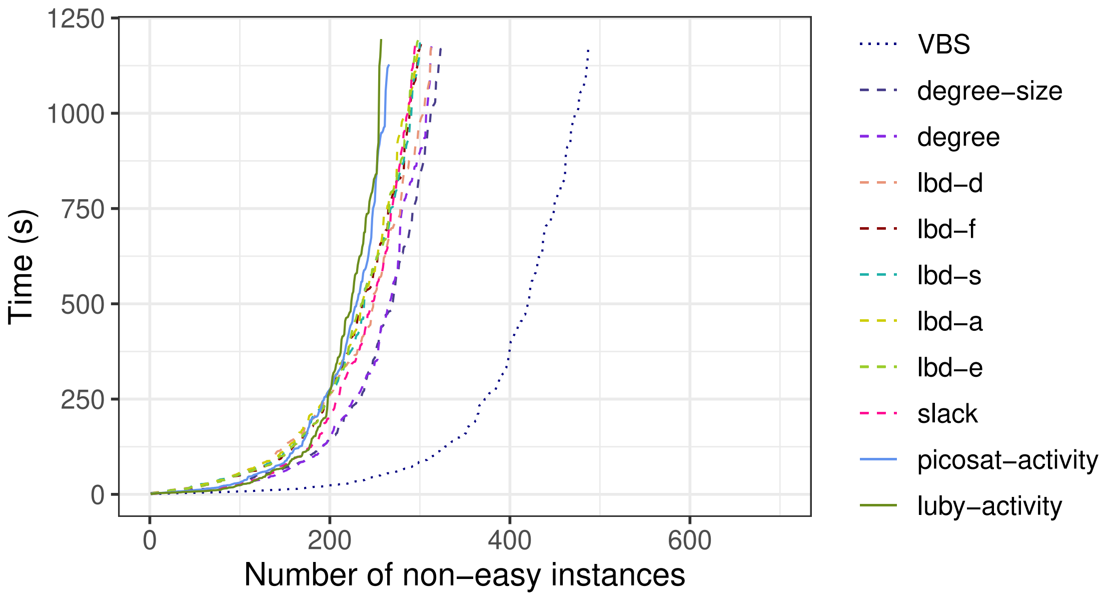


Figure 7.16: Cactus plot of the various quality measures used for learned constraint deletion and dynamic restarts in *Sat4j-GeneralizedResolution*.

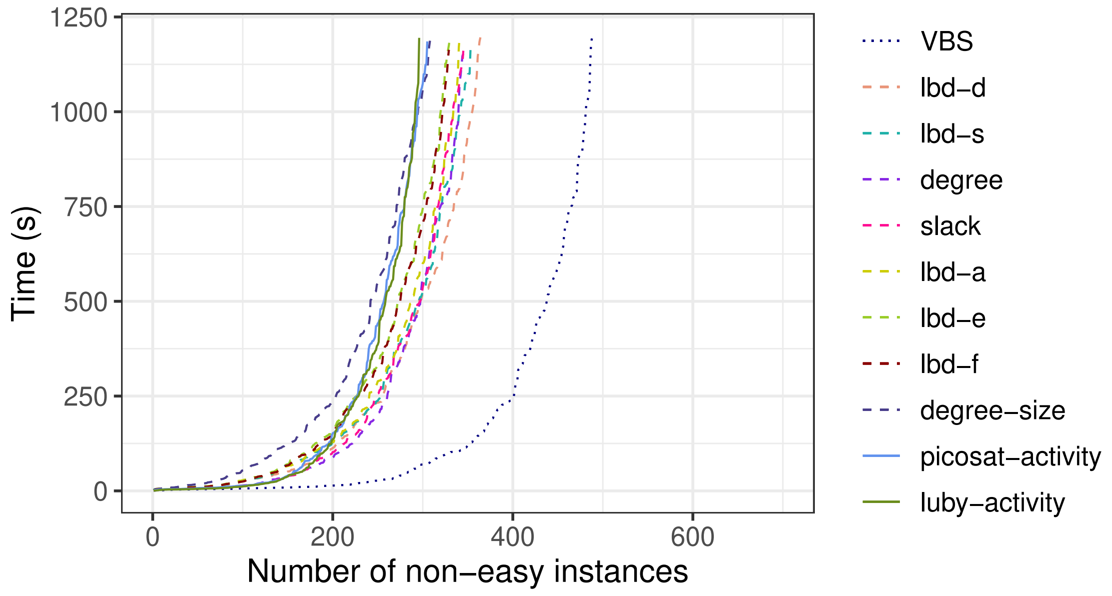


Figure 7.17: Cactus plot of the various quality measures used for learned constraint deletion and dynamic restarts in *Sat4j-RoundingSat*.

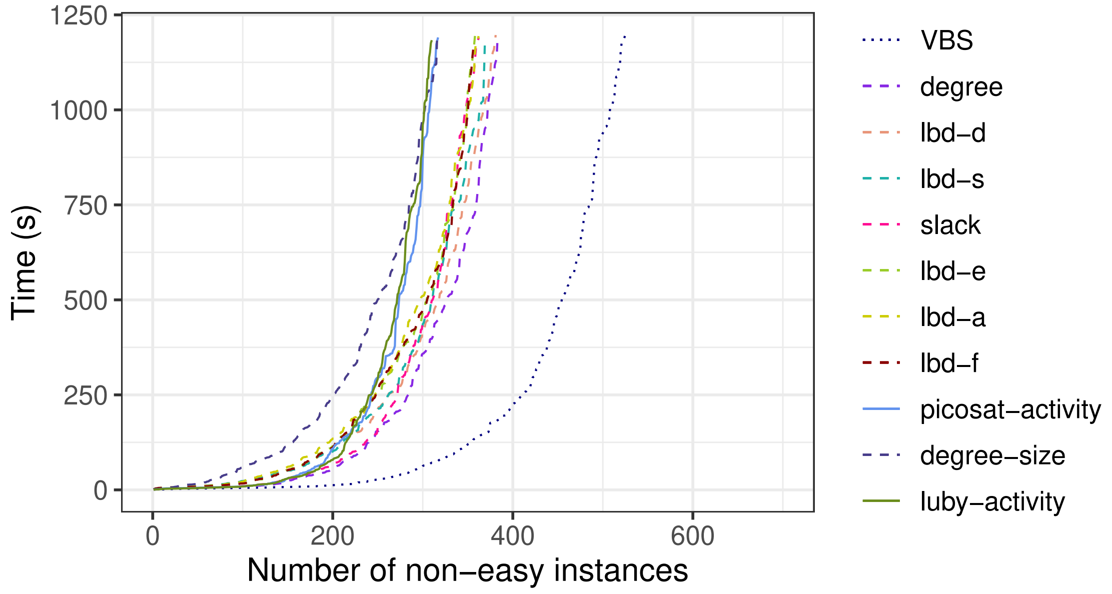


Figure 7.18: Cactus plot of the various quality measures used for learned constraint deletion and dynamic restarts in *Sat4j-PartialRoundingSat*.

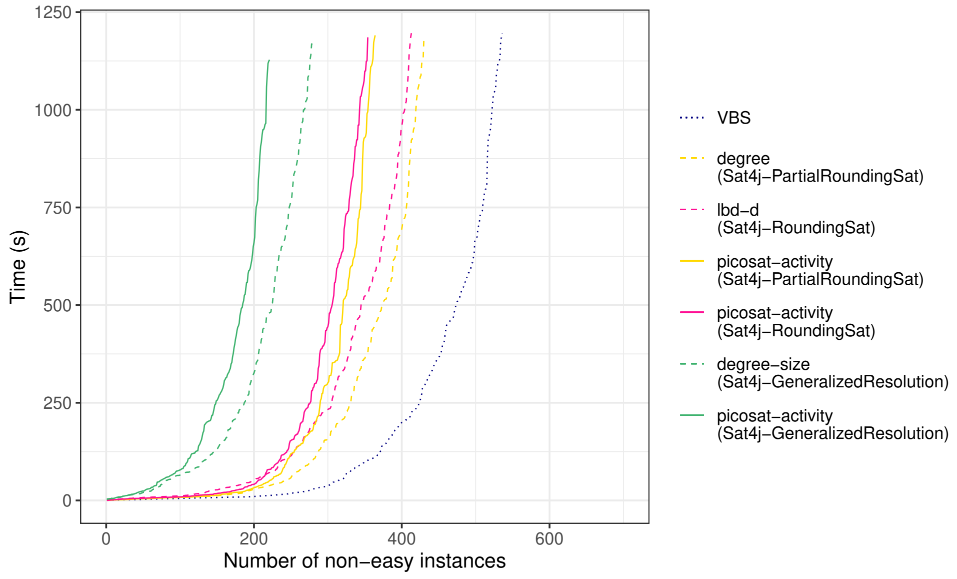


Figure 7.19: Cactus plot of the best learned constraint deletion strategies implemented in different *Sat4j* solvers.

LCD and Restart Strategy	Solved Instances	SOTA Contribution	1-second Contribution	10-second Contribution
degree (Sat4j-PartialRoundingSat)	3920	36	143	90
lbd-d (Sat4j-RoundingSat)	3902	40	153	118
picosat-activity (default) (Sat4j-PartialRoundingSat)	3855	9	106	32
picosat-activity (default) (Sat4j-RoundingSat)	3843	8	79	28
degree-size (Sat4j-GeneralizedResolution)	3769	7	34	24
picosat-activity (default) (Sat4j-GeneralizedResolution)	3711	3	29	15

Table 7.13: Table summarizing the results of the best quality measures used during learned constraint deletion and restarts for several configurations of *Sat4j*. Columns display, from left to right, the quality measure, the number of instances solved by the corresponding configuration, its state-of-the-art contribution, the number of instances for which the configuration was at least 1 second faster than all the others and the number of instances for which the configuration was at least 10 seconds faster than all the others

7.3.2 Combining All "Best" Strategies

In order to take advantage of the best strategies we identified empirically, we now evaluate the performance of *Sat4j* with their combination.

In the case of *Sat4j-GeneralizedResolution*, the best strategies are *bump-effective*, *delete-lbd-s* and *restart-degree*. We thus ran the solver using this configuration in the same experimental setting as before, and got the results given in Figures 7.20 and 7.21.

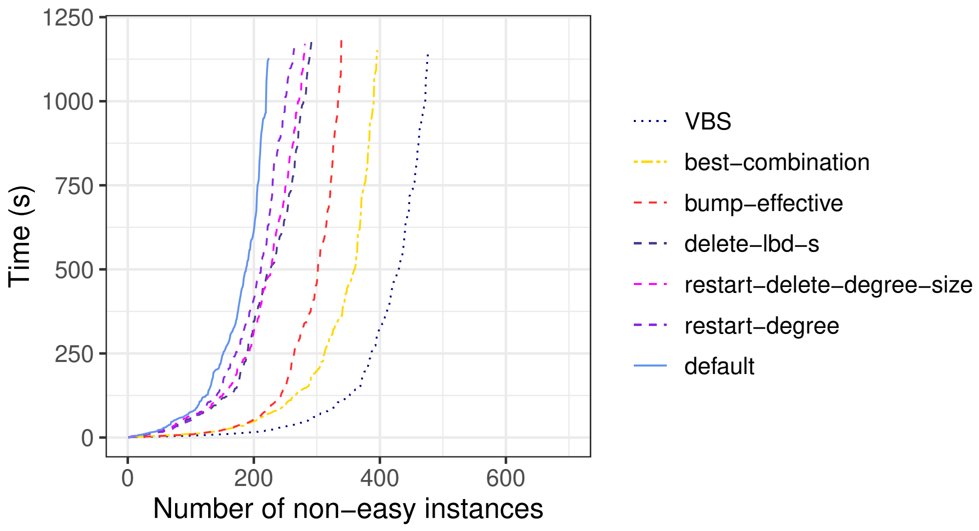


Figure 7.20: Cactus plot of the best strategies enabled in *Sat4j-GeneralizedResolution*.

Clearly, the combination of the best strategies allows to improve the overall performance of the solver, as also confirmed by the results given in Table 7.14. This suggests that combining the different strategies allows to get improvements from all of them. However, there is still room for additional benefits, as all configurations contribute to the VBS. Moreover, as shown in the scatter plot (Figure 7.21), the combination of all best strategies does not improve the solver on all benchmarks, and in particular not those of the `FPGA_SAT05` family.

Enabled Strategy	Solved Instances	SOTA Contribution	1-second Contribution	10-second Contribution
best-combination	3884	31	213	127
bump-effective	3826	5	130	53
delete-lbd-s	3779	8	42	28
restart-delete-degree-size	3769	11	68	42
restart-degree	3751	9	57	25
default	3711	3	14	5

Table 7.14: Table summarizing the results of the best strategies enabled in *Sat4j-GeneralizedResolution*. Columns display, from left to right, the configuration of the solver, the number of instances solved by this configuration, its state-of-the-art contribution, the number of instances for which the configuration was at least 1 second faster than all the others and the number of instances for which the configuration was at least 10 seconds faster than all the others.

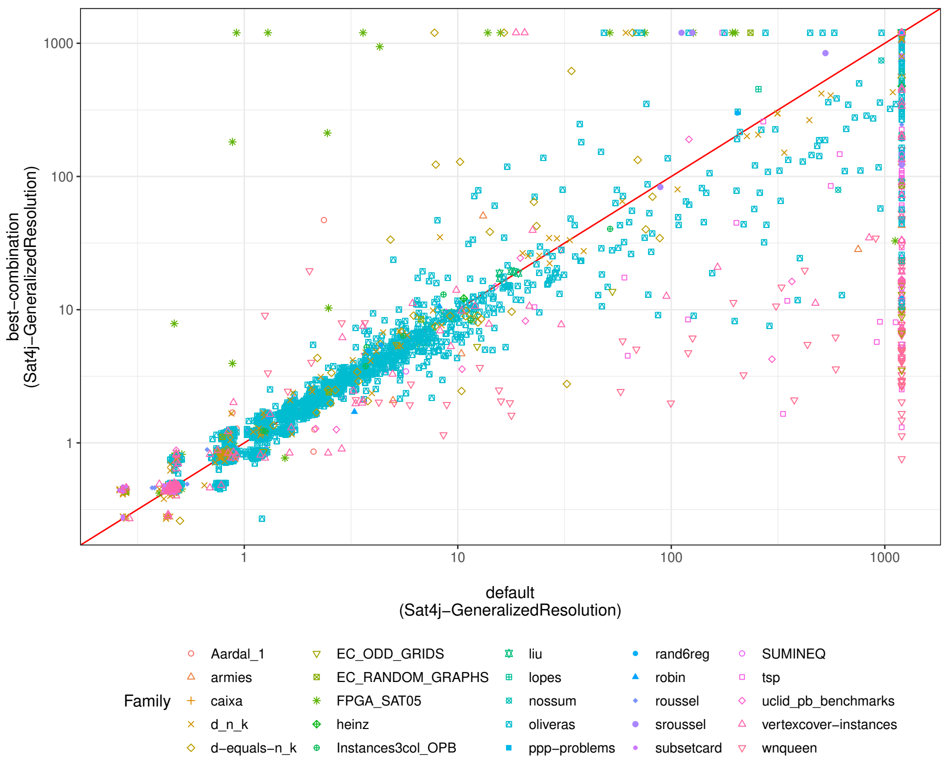


Figure 7.21: Scatter plot comparing the runtime (in seconds) of the default configuration of *Sat4j-GeneralizedResolution* and the combination of the best strategies of this solver (logarithmic scale).

In the case of *Sat4j-RoundingSat*, the best strategies are *bump-assigned*, *delete-slack* and *restart-picosat*. We thus ran the solver using this configuration in the same experimental setting as before, and got the results given in Figures 7.22 and 7.23.

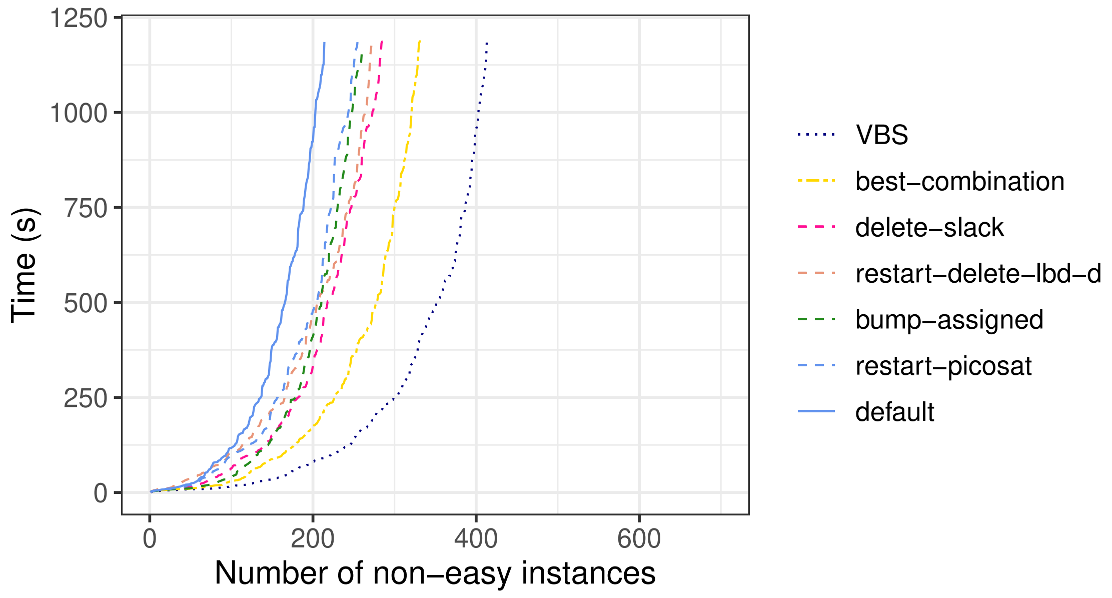


Figure 7.22: Cactus plot of the best strategies enabled in *Sat4j-RoundingSat*.

Once again, the combination of the best strategies allows to improve the overall performances of the solver, even though the improvement is not as large as that observed with *Sat4j-GeneralizedResolution*. This may be explained by the fact that the proof system used in *Sat4j-RoundingSat* already allows to focus on literals that are important with respect to the conflict being analyzed, which is one of the purposes of the strategies designed in this chapter. Yet, as may be observed in the scatter plot and in the results given by Table 7.15, the combination of the best strategies allows to more efficiently solve a large number of the considered instances.

Enabled Strategy	Solved Instances	SOTA Contribution	1-second Contribution	10-second Contribution
best-combination	3962	30	111	76
delete-slack	3918	12	51	41
restart-delete-lbd-d	3902	17	128	94
bump-assigned	3890	6	45	22
restart-picosat (no-deletion)	3884	6	33	23
default	3843	9	20	15

Table 7.15: Table summarizing the results of the best strategies enabled in *Sat4j-RoundingSat*. Columns display, from left to right, the configuration of the solver, the number of instances solved by this configuration, its state-of-the-art contribution, the number of instances for which the configuration was at least 1 second faster than all the others and the number of instances for which the configuration was at least 10 seconds faster than all the others.

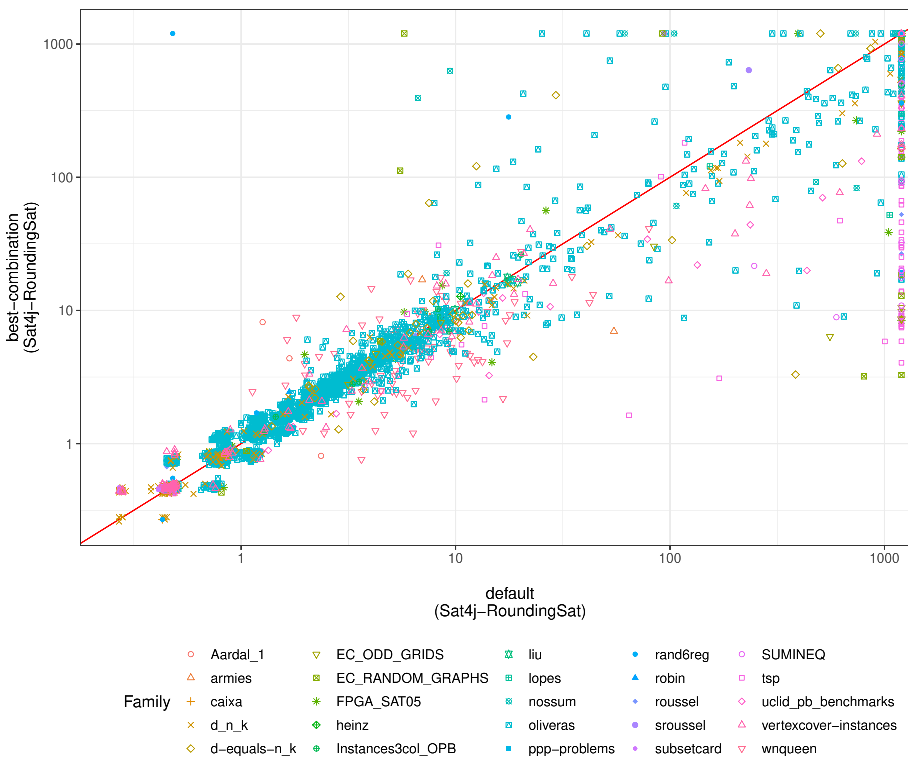


Figure 7.23: Scatter plot comparing the runtime (in seconds) of the default configuration of *Sat4j-RoundingSat* and the combination of the best strategies of this solver (logarithmic scale).

In the case of *Sat4j-PartialRoundingSat*, the best strategies are *bump-assigned*, *delete-degree-size* and *restart-picosat*. We thus ran the solver using this configuration in the same experimental setting as before, and got the results given in Figures 7.24 and 7.25.

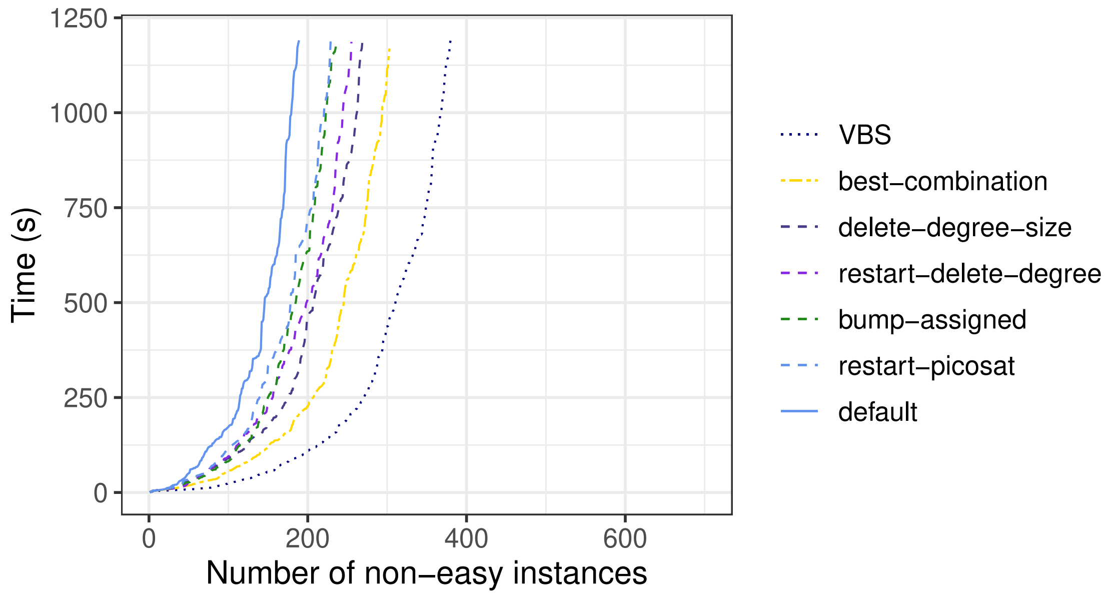


Figure 7.24: Cactus plot of the best strategies enabled in *Sat4j-PartialRoundingSat*.

From these figures and the results given in Table 7.16, we can make similar observations as the ones made with *Sat4j-RoundingSat*. In particular, the combination of the different strategies allows to improve the performance of the solver. However, as for *Sat4j-GeneralizedResolution*, the family `FPGA_SAT05` is solved faster with the default configuration of *Sat4j-PartialRoundingSat*.

Enabled Strategy	Solved Instances	SOTA Contribution	1-second Contribution	10-second Contribution
best-combination	3969	20	112	81
delete-degree-size	3935	8	52	44
restart-delete-degree	3920	19	125	69
bump-assigned	3899	11	48	25
restart-picosat (no-deletion)	3895	9	38	24
default	3855	7	22	13

Table 7.16: Table summarizing the results of the best strategies enabled in *Sat4j-PartialRoundingSat*. Columns display, from left to right, the configuration of the solver, the number of instances solved by this configuration, its state-of-the-art contribution, the number of instances for which the configuration was at least 1 second faster than all the others and the number of instances for which the configuration was at least 10 seconds faster than all the others.

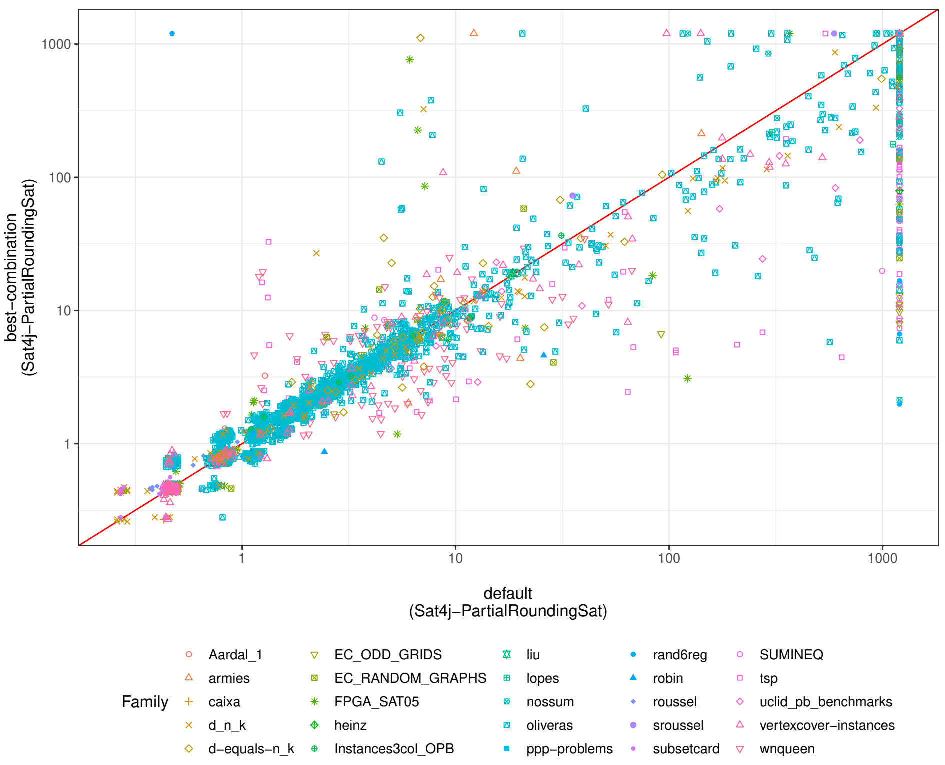


Figure 7.25: Scatter plot comparing the runtime (in seconds) of the default configuration of *Sat4j-PartialRoundingSat* and the combination of the best strategies of this solver (logarithmic scale).

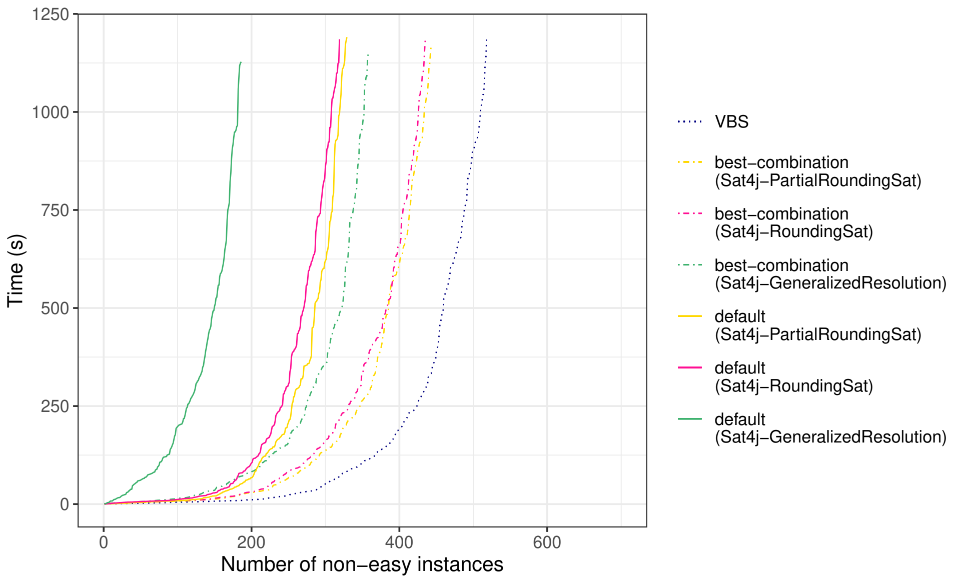


Figure 7.26: Cactus plot of the combinations of the best strategies in different configurations of *Sat4j*.

Let us now compare all the best configurations of *Sat4j* studied in this section together with the default configurations, considering the cactus plot in Figure 7.26. The cactus plot shows that all best configurations allow a significant improvement of the solver, which is also confirmed by the results given in Table 7.17. This is particularly true for *Sat4j-GeneralizedResolution*: by combining the best strategies, the solver solves now more instances than the default configuration of *Sat4j-RoundingSat* and *Sat4j-PartialRoundingSat*. This also illustrates the importance of the CDCL strategies implemented in pseudo-Boolean solvers, and their complementarity with the proof system used by the solver.

Even if the different strategies introduced in this chapter allow a significant improvement of *Sat4j* pseudo-Boolean solvers, these strategies are not enough to allow beating the original implementation of *RoundingSat*, as illustrated in Figure 7.27.

However, if instead of the slack-based approach, the conservative variant of the watched literal-based approach for detecting propagations is exploited, the performance improves significantly, as shown in Figure 7.28.

We also implemented new variants of *Sat4j-Both* taking advantage of the strategies described in this chapter. Recall that *Sat4j-Both* runs in parallel both *Sat4j-GeneralizedResolution* and *Sat4j-Resolution*. The new variants we consider replace the use of *Sat4j-GeneralizedResolution* in this solver by one of the *best-combinations* presented above. Figure 7.29 shows the results of these new variants. In this case, the difference between the solvers is not as clear as in the previous experiments, as *Sat4j-Resolution* is most of the time the fastest of the two solvers run in parallel by the different *Sat4j-Both* variants.

As shown in the cactus plot, *Sat4j-Resolution*, *Sat4j-Both* and even more significantly the original implementation of *RoundingSat* remain more efficient in practice than the different configurations of *Sat4j* studied in this chapter. However, if the improvements we observe in this solver could be brought to *RoundingSat*, for instance, they would allow to significantly improve the state-of-the-art of pseudo-Boolean solving.

Solver Configuration	Solved Instances	SOTA Contribution	1-second Contribution	10-second Contribution
best-combination (Sat4j-PartialRoundingSat)	3969	26	168	110
best-combination (Sat4j-RoundingSat)	3962	25	132	81
best-combination (Sat4j-GeneralizedResolution)	3884	10	95	56
default (Sat4j-PartialRoundingSat)	3855	5	66	16
default (Sat4j-RoundingSat)	3843	5	51	19
default (Sat4j-GeneralizedResolution)	3711	2	28	10

Table 7.17: Table summarizing the results of the combinations of the best strategies for several configurations of *Sat4j*. Columns display, from left to right, the configuration of the solver, the number of instances solved by this configuration, its state-of-the-art contribution, the number of instances for which the configuration was at least 1 second faster than all the others and the number of instances for which the configuration was at least 10 seconds faster than all the others.

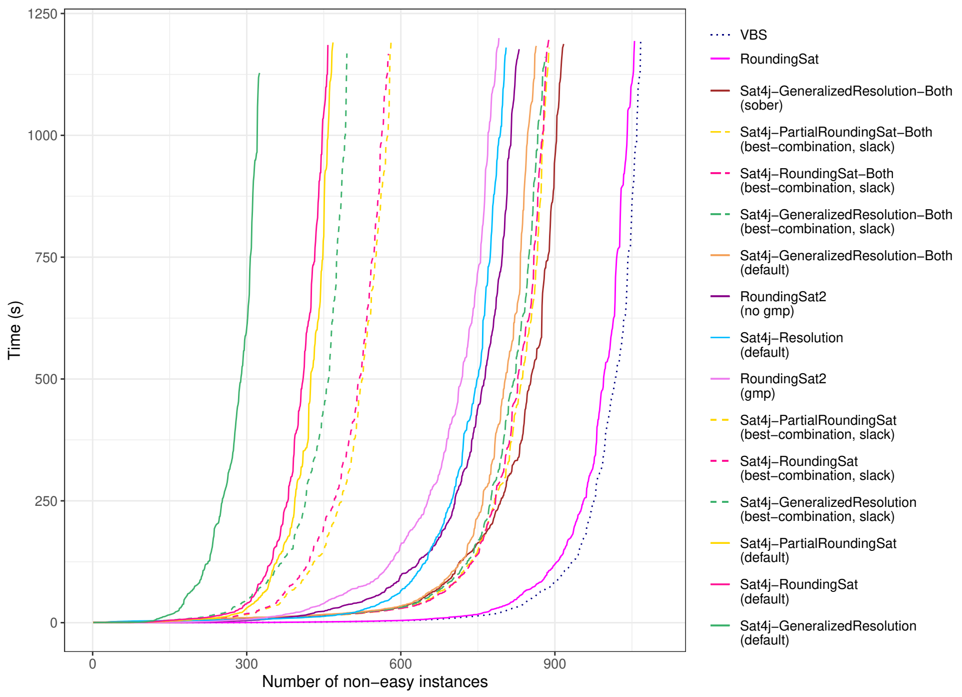


Figure 7.27: Cactus plot comparing several configurations of *Sat4j* with different versions of *RoundingSat*.

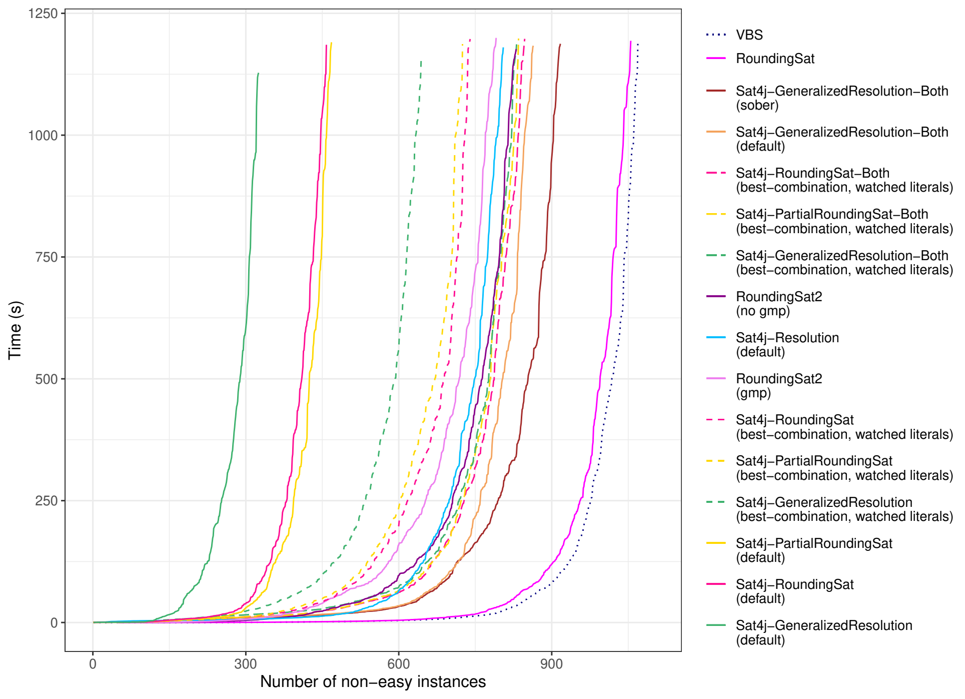


Figure 7.28: Cactus plot comparing several configurations of *Sat4j* with different versions of *RoundingSat* (using watched literals).

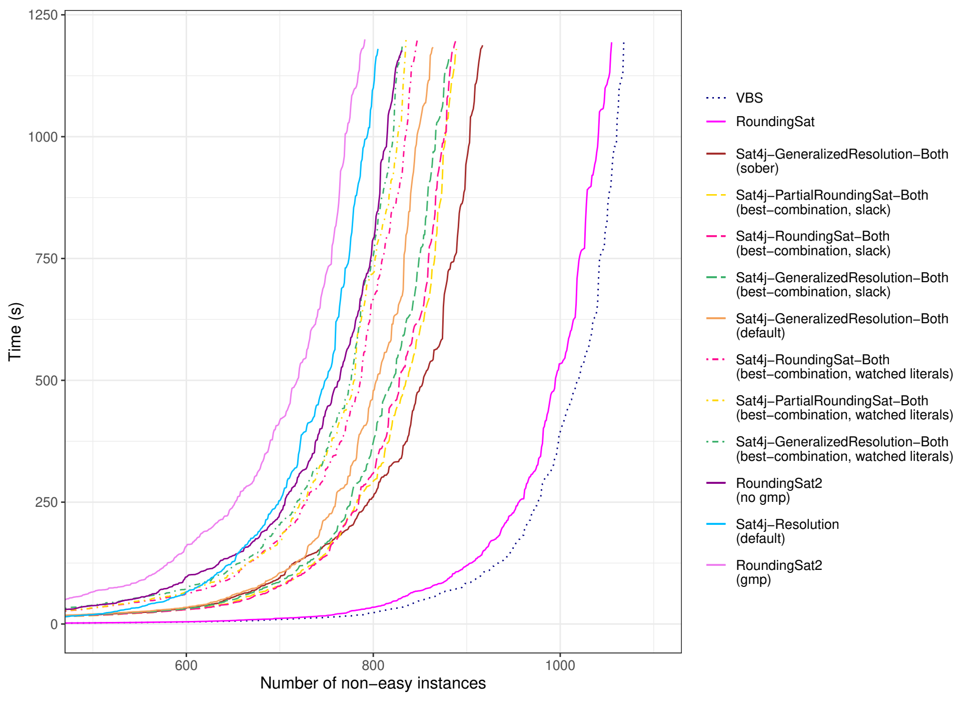


Figure 7.29: Cactus plot comparing several configurations of *Sat4j* with different versions of *RoundingSat* (many core solvers).

Conclusion

We have presented and evaluated a number of approaches aiming to improve the performance of pseudo-Boolean solvers based on the cutting planes proof system. In particular, we have shown that such solvers may inherently produce *irrelevant* literals during their conflict analysis, which in turns may lead to the inference of weaker constraints, and thus on longer unsatisfiability proofs. We have empirically observed this behavior on the pseudo-Boolean solver *Sat4j*, and shown that *RoundingSat*-based solvers also produce such literals. This work has been published in [LMMW20]. This result is important because it means that none of the existing proof systems implemented in pseudo-Boolean solvers prevents from producing irrelevant literals. An obvious perspective is thus to design a strategy for applying cutting planes rules that prevents the creation of irrelevant literals. We have not been able to devise such a strategy in the time spent for the preparation of this thesis. We provided however a way to detect irrelevant literals when they are produced. While our approach is incomplete and slow in practice (detecting irrelevant literals is NP-hard), we empirically identified families of instances in which irrelevant literals make the solver produce exponentially larger unsatisfiability proofs.

We have also designed weakening strategies that are used by the solver to guarantee the derivation of a conflicting constraint during conflict analysis. While none of the strategies we presented is better than all the others on all benchmarks, we empirically showed that applying *partial weakening*, which produces stronger constraints, allows to improve the performance of the solver. In relation with the previous result, we showed that one can weaken so-called *ineffective* literals to also remove irrelevant literals, while doing so forces to derive clauses only, and thus weak constraints. Quite interestingly, while the weakening rule has mostly been applied on the *reason side* during conflict analysis, our experiments also suggest that applying it on the *conflict side* may actually be preferable. Such a result opens new perspectives to improve the weakening process, by considering the conflicting constraint as amenable to weakening as the reason constraint. This work has been published in [LMW20].

Finally, we designed and implemented different strategies inspired by those used in modern SAT solvers that take into account the properties of pseudo-Boolean constraints to improve the performance of the solver. In particular, we have shown that considering the current assignment as well as the coefficients appearing in the pseudo-Boolean constraints encountered by the solver allows to significantly improve its runtime. The results of such a study allowed us to build new solvers in *Sat4j* with the best available strategies for each available proof system. We believe that the data gathered during this extensive experimental evaluation can bring additional hints about the pros and cons of each strategy by looking at the benchmark level (instead of the family level as in this document). Preliminary results concerning this work have been presented in [Wal20].

General Conclusion and Perspectives

In this thesis, we have studied different aspects of pseudo-Boolean reasoning, both from theoretical and practical perspectives.

First, we considered properties of pseudo-Boolean constraints as a propositional language [LMMW18]. We have in particular shown that this language is not suitable for knowledge compilation (checking the consistency of a pseudo-Boolean formula is not tractable). More precisely, considering the criteria of the knowledge compilation map [DM02], pseudo-Boolean languages do not offer additional queries compared to CNF, while some transformations offered by this language are no longer tractable when considering pseudo-Boolean constraints instead of clauses (for instance, singleton forgetting or closure under bounded disjunction). The main advantage of pseudo-Boolean constraints, from a knowledge representation perspective, is thus their succinctness: a single pseudo-Boolean constraint may represent exponentially many clauses.

As the succinctness criterion considers only *equivalent* formulae, it does not take into account *encodings* that allow the introduction of auxiliary variables. It is well-known that allowing the use of such variables may drastically reduce the size of a formula. However, we have shown that, if we bound the width of such encodings, their expressiveness may be drastically limited, restricting the formulae to those of low communication complexity [MW19, MW20].

From a practical viewpoint, we have investigated different approaches for improving the performance of pseudo-Boolean solvers. In particular, we have shown that irrelevant literals may be introduced by the application of cutting planes rules, and lead to the inference of weaker constraints [LMMW20]. We empirically showed that the unsatisfiability proof produced by the solver may be exponentially larger in the presence of irrelevant literals, but dealing with irrelevant literals is hard in practice: detecting them is NP-hard.

A possible counter-measure for removing efficiently irrelevant literals is to take advantage of the weakening rule to weaken away so-called *ineffective* literals (even though these literals may also be relevant). We studied different weakening strategies and showed that, despite none of the existing and new strategies is better than the others on all considered benchmarks, the way this rule is applied may have a significant impact on the performance of the solver [LMW20]. An interesting observation we made is that, while most implementations apply the weakening rule on the *reason* side of the cancellation rule (*RoundingSat* [EN18] is the first solver that applies it on *both* sides), it may actually be preferable to apply it on the *conflict* side to get better performance.

Finally, we presented different approaches for adapting CDCL strategies to pseudo-Boolean solving. Indeed, it is well-known that many features implemented in resolution-based SAT solvers are required to get the best from these solvers. This is particularly true for branching heuristics, learned constraint deletion strategies and restart policies. We presented a wide variety of adaptions to take into account the specific form of pseudo-Boolean constraints in the counterpart of these strategies implemented in pseudo-Boolean solvers. In particular, we showed that taking into account the current assignment and the coefficients appearing in the constraints may allow to drastically improve the performance of the solvers. All these strategies have been implemented in the pseudo-Boolean solver *Sat4j* [LP10] and are publicly available in its repository[6].

While studying the subjects above, some questions have been raised, that will need to be investigated further in order to improve the performance of pseudo-Boolean solvers. In particular, our approach for detecting irrelevant literals highlighted that one can exponentially reduce the size of the unsatisfiability proofs produced by the solver by removing irrelevant literals before they lead to the inference of weaker constraints. However, this approach is not efficient enough to be considered as a counter-measure to the production of irrelevant literals. Instead, the ideal solution would be to identify a strategy for applying cutting planes rules that guarantees to derive constraints that only contain relevant literals when the input only contains such literals.

Another avenue to explore is to consider different combinations of the strategies presented in this thesis, regarding the application of the weakening rule, branching heuristics, learned constraint deletion strategies and restart policies. In particular, all these strategies are often tightly linked in the solver, and understanding and identifying their interaction is hard in practice, as they may have side effects on each other. Moreover, the solver does not explore the same search space from one strategy to another, and it may thus learn different constraints, making harder the comparison of the behavior of the solver in its different configurations.

During the three years of the preparation of this thesis, we also observed some "strange" behaviors of pseudo-Boolean solvers that we did not investigate, and can definitely lead to improvements in pseudo-Boolean solving if they are further studied. For instance, it appears that the 1-UIP scheme inherited from SAT solving is not always optimal when considering pseudo-Boolean constraints. More precisely, even if an assertive constraint has already been derived, continuing conflict analysis may allow to identify a higher backtrack level, as in the following example.

Example 91

Suppose that, during the execution of the solver, a conflict is encountered with the constraint $4a(0@10) + 4b(1@30) + 3c(0@20) + 3d(0@30) + 2e(1@30) + f(0@40) + g(0@40) + z(0@40) \geq 8$, and that the reason for f and g is $3i(0@20) + 3j(0@40) + 2\bar{f}(1@40) + 2\bar{g}(1@40) + h(1@40) \geq 5$. A cancellation is applied between these two constraints to get $8a(0@10) + 8b(1@30) + 6c(0@20) + 6d(0@30) + 4e(1@30) + 2z(0@40) + 3i(0@20) + 3j(0@40) + h(1@40) \geq 17$. Observe that this constraint is assertive, and propagates b at decision level 20. This constraint is thus learned, and the conflict analysis stops.

However, suppose now that the reason for j is $6\bar{c}(1@20)+6\bar{d}(1@30)+3\bar{j}(1@40)+3k(0@40)+3l(0@30) \geq 15$. If we apply the cancellation rule between this constraint and the constraint that we learned, we obtain the constraint $8a(0@10) + 8b(1@30) + 4e(1@30) + 2z(0@40) + 3i(0@20) + 3k(0@40) + 3l(0@30) + h(1@40) \geq 17$, which is also assertive and propagates b at decision level 10. Stopping the analysis at the first decision level was here suboptimal.

Note that continuing the analysis does not always allow to improve the backjump, as if we instead resolve the learned constraint with the reason for z given by $10w(1@25) + 10x(0@25) + y(0@40) + \bar{z}(1@40) \geq 11$, the constraint we get is $20w(1@25) + 20x(0@25) + 8a(0@10) + 8b(1@30) + 6c(0@20) + 6d(0@30) + 4e(1@30) + 3i(0@20) + 3j(0@40) + 2y(0@40) + h(1@40) \geq 37$, which is still assertive, but now propagates b at decision level 25.

[6]https://gitlab.ow2.org/sat4j/sat4j

From this observation, several questions arise. First, what would be a criterion determining when to stop conflict analysis in the case of pseudo-Boolean solving, so as to identify the highest possible backtrack level? Second, is it *worth* computing it? Indeed, if this computation is too costly in practice, it may be preferable to settle for the current approach, even though the backjump level is not optimal. Also, the problem of the optimal backjump seems to be tightly linked to the order in which cancellations are applied (see the impact of resolving with the reason of j or that of z in the example). Currently, cancellation are applied in the reverse order of the propagations. Could we instead use a heuristic, to identify the best order to use?

Another perspective is to find a better algorithm for detecting propagations and conflicts. Indeed, during our experiments, we made a quite surprising observation: as pseudo-Boolean constraints can become conflictual after having propagated some literals, it may happen that a reason encountered during conflict analysis is actually conflicting. This behavior can be explained by the fact that multiple constraints may be conflictual at the same time, and the solver has thus to select one of them to perform conflict analysis, and one of them may be a reason for the propagation of some literals. As such, during conflict analysis, it may happen that the conflict is actually resolved against a conflicting reason, as in the following example.

Example 92

Consider the following pseudo-Boolean constraints:

- $\chi_1 \equiv a + \bar{b} + \bar{c} \geq 2$
- $\chi_2 \equiv 3b + 3d + e + f \geq 4$
- $\chi_3 \equiv 2c + \bar{e} + \bar{f} \geq 2$
- $\chi_4 \equiv b + \bar{d} + e + f \geq 1$

Now, suppose that the solver decides to assign a to 0. Then, unit propagations are triggered. In particular, χ_1 propagates both $\bar{b}$ and $\bar{c}$, and χ_2 propagates then d. Propagations continue, and we finally get:

- $\chi_1 \equiv a(0@1) + \bar{b}(1@1) + \bar{c}(1@1) \geq 2$
- $\chi_2 \equiv 3b(0@1) + 3d(1@1) + e(0@1) + f(0@1) \geq 4$
- $\chi_3 \equiv 2c(0@1) + \bar{e}(1@1) + \bar{f}(1@1) \geq 2$
- $\chi_4 \equiv b(0@1) + \bar{d}(0@1) + e(0@1) + f(0@1) \geq 1$

Observe that both χ_2 and χ_4 are now conflicting, and that the reason for d, which appears in χ_4, is χ_2. The solver has now to choose a conflict on which to perform the analysis. If the chosen conflict is χ_4, the conflict analysis produces the following:

$$\cfrac{\cfrac{\cfrac{\chi_4 \qquad \chi_3}{b(0@1) + c(0@1) + \bar{d}(0@1) \geq 1} \qquad \chi_2}{4b(0@1) + 3c(0@1) + e(0@1) + f(0@1) \geq 4} \qquad \chi_1}{3a(0@1) + b(0@1) + e(0@1) + f(0@1) \geq 4}$$

Observe that, at the second step, the conflict is resolved against the (conflicting) reason χ_2, which also reintroduces the two literals e and f that were cancelled by the first cancellation.

Now, if we select instead χ_2 as the conflict, we get instead the following conflict analysis:

$$\dfrac{\dfrac{\chi_2 \qquad \chi_3}{3b(0@1) + 3d(1@1) + 2c(0@1) \geq 4} \qquad \chi_1}{3d(1@1) + 2a(0@1) + b(0@1) \geq 4}$$

Note that the learned constraint obtained in this latter case is incomparable with that obtained in the previous case, so that it is not clear which of the two approaches is the best.

Currently, when a conflicting reason is encountered, the cancellation is performed silently, as if the reason was not conflictual (actually, the solver does not *know* that this reason is conflictual). Is this the best solution to handle conflictual reasons? For instance, could we simply "forget" what we have done with the previous conflict, and use the conflicting reason as a new conflict when encountered during the analysis?

Another way to improve pseudo-Boolean solvers is to investigate the implementation of other cutting planes rules, such as the *addition* rule. Indeed, doing so would allow to derive pseudo-Boolean constraints from clauses, and thus prevent the proof system internally used by the solver from degenerating to resolution when clauses are given as input. For instance, doing so is required to find a short unsatisfiability proof for pigeonhole principle formulae encoded as CNF. Different approaches for detecting cardinality constraints from clauses have been proposed [BLLM14, EN20]. However, none of them use the addition rule to derive cardinality constraints.

Thanks to all the improvements we proposed in the thesis, either already implemented or mentioned above, one may consider the use of pseudo-Boolean solvers in other settings. In particular, we focused here on *decision* problems. It would be interesting to evaluate the impact of these improvements on *optimization problems*, which aim to minimize or maximize the value of a given objective function (see [BH02] for more details). Typically, multiple calls to the solver must be performed for solving optimization problems, and each call uses the last model found to refine the solution until an optimal solution is identified. One could take advantage of these multiple calls to try the different strategies we presented so as to identify the most appropriate ones for solving the input. One could also extend these strategies to better adapt them to optimization problems, and consider the intermediate models to guide the search performed by the subsequent calls to the solver, as proposed in [Nad19] for the case of phase selection.

Another interesting use of pseudo-Boolean solvers is to consider them as *oracles* in more complex settings, for instance in knowledge compilation. Many compilers use internally a SAT solver to build, for instance, a d-DNNF circuit that is equivalent to a CNF formula given as input [Dar04, MMBH12, LM17]. Using a pseudo-Boolean solver would both allow to compile pseudo-Boolean formulae and to take advantage of their succinctness to compile formulae for which the CNF input would be too big to be compiled efficiently. This is the case, for instance, for formulae representing *neural networks*, and for which queries allowing to evaluate the robustness of the model learned by the neural network would be tractable on the compiled form [NKR+18].

Appendices

Appendix A

Description of the Solvers

Most of the solvers considered in this thesis have participated, in a certain form, to the Pseudo-Boolean Competition 2016 [Rou16]. A more specific attention is paid to the solvers that implement cutting planes-based reasoning, and thus we focus on *RoundingSat* [EN18] and *Sat4j* [LP10].

A.1 *RoundingSat*

The *RoundingSat* solver [EN18] is a pseudo-Boolean solver written in C++, and that supports cutting planes-based reasoning. This solver makes an heavy use of the weakening and division rules, so as to allow the use of fixed precision arithmetic and to be more efficient in practice. The proof system of this solver is further studied in Section 4.2.

Recently, a new version of *RoundingSat* has been released, named *RoundingSat2* in this document, with many new features, especially arbitrary precision arithmetic (either using the `boost` or the `gmp` library).

The considered version of *RoundingSat* is that corresponding to the commit of November 16th, 2019 (`e1c97a73`) while the version of *RoundingSat2* is that of September 4th, 2020 (`383cce49`) from *RoundingSat*'s repository[7].

A.2 *Sat4j*

The *Sat4j* library [LP10] is both a SAT and pseudo-Boolean solver written in Java, which implements different proof systems to solve pseudo-Boolean problems.

In particular, *Sat4j-Resolution* implements a resolution-based reasoning, as described in Section 4.3, that benefits both from the succinctness of pseudo-Boolean constraints and the efficiency of modern SAT solvers.

In *Sat4j-GeneralizedResolution*, the proof system is the generalized resolution one introduced in [Hoo88] and considered in Section 4.2. In practice, this solver is much slower than *Sat4j-Resolution*, as it has to deal with the complex operations related to the cutting planes proof system, and especially those relying on arbitrary precision arithmetic. However, *Sat4j-GeneralizedResolution* is faster than the resolution-based solver when dealing with problems that are hard for resolution, such as pigeonhole principle instances.

To benefit from the performance of both *Sat4j-Resolution* and *Sat4j-GeneralizedResolution*, *Sat4j-Both* runs both solvers in parallel, (so that, basically, the CPU time is equally divided between the two

[7] `https://gitlab.com/miao_research/roundingsat`

solvers). When one of the two solvers finds a solution or an unsatisfiability proof, *Sat4j-Both* returns the answer of this solver. As *Sat4j-GeneralizedResolution* is in practice much slower than *Sat4j-Resolution*, the *Sat4j-Both (sober)* variant has been developed, based on the following observation: when *Sat4j-GeneralizedResolution* is efficient, this is because of the strength of its underlying proof system. In this case, the answer is given almost immediately. *Sat4j-Both (sober)* runs thus *Sat4j-GeneralizedResolution* for 1 minute, while it runs *Sat4j-Resolution* for the entire runtime allocated to the solver.

Finally, *Sat4j-RoundingSat* is an implementation of *RoundingSat*'s proof system in *Sat4j*, which still uses arbitrary precision arithmetic and applies more parsimoniously the weakening and division rules (see the next section for a more detailed comparison of the two solvers).

Note that, because *Sat4j* is written in Java, it is basically 3 to 4 times slower than any equivalent C++ solver. In particular, *Sat4j-RoundingSat* is necessarily slower than *RoundingSat*.

During the preparation of this thesis, I participated in the development of different solvers and strategies in *Sat4j*. In particular, I have contributed to this solver by writing about $7,000$ lines of code, in more than 100 classes.

A.3 Implementation Details of *RoundingSat* vs *Sat4j*

The table on the following page compares the different features implemented by *RoundingSat* and by different solvers of *Sat4j*, in their default configurations. *Sat4j-Both* is not mentioned, as it runs in parallel *Sat4j-Resolution* and *Sat4j-GeneralizedResolution* in their default configurations.

A.4 Other Solvers

This thesis also mentions SAT-based pseudo-Boolean solvers, such as *MiniSat+* [ES06], *NaPS* [SN15] or *Open-WBO* [MML14]. As these solvers are based on the resolution proof system, we do not insist on their internal implementations.

The different encodings used to encode the input pseudo-Boolean problems into a CNF instances are further studied in Section 4.3.

	Sat4j-Resolution	*Sat4j-GeneralizedResolution*	*Sat4j-RoundingSat*	*RoundingSat*	*RoundingSat2*
Number representation	Arbitrary precision			Fixed precision	Arbitrary precision
Propagation detection	Watched literals (in clauses and cardinality constraints) Slack (in pseudo-Boolean constraints)			Watched literals for all constraints, the maximum coefficient is set once and for all	Watched literals (in clauses and cardinality constraints) *opt-watch* (in pseudo-Boolean constraints in which less than 30% of the literals must be watched) Slack (in other pseudo-Boolean constraints)
Weakening on reason	Reduction to clause	Reduction until conflict is ensured	Weaken all non-falsified literals non-divisible by the coefficient α of the pivot and division by α unless the pivot has coefficient 1 in the reason or in the conflict	Weaken all non-falsified literals non-divisible by the coefficient α of the pivot and division by α	Partially weaken all non-falsified literals non-divisible by the coefficient α of the pivot and division by α
Weakening on conflict	Reduction to clause	None	Weaken all non-falsified literals non-divisible by the coefficient α of the pivot and division by α unless the pivot has coefficient 1 in the reason or in the conflict	Weaken all non-falsified literals non-divisible by the coefficient α of the pivot and division by α	Partially weaken all non-falsified literals non-divisible by the coefficient α of the pivot and division by α
Cancellation	Classical resolution (all constraints are clauses)	Cancellation			
Rounding on learned constraint	None (all constraints are clauses)	None		Round to cardinality constraint if the coefficients reach 10^9	Round coefficients to 29 bits if their size reaches 62 bits by applying partial weakening and division (as for the reduction of the conflict above)
Post-processing on learned constraint	None (all constraints are clauses)	None		Simplify literals assigned at decision level 0 Weaken non-falsified literals that are not propagated Weaken falsified literals that preserve the propagations	Simplify literals assigned at decision level 0 Weaken non-falsified literals that are not propagated Weaken falsified literals that preserve the propagations Divide the coefficients by their GCD Reduce to an equivalent cardinality constraint if any
Branching heuristic	EVSIDS, with each literal encountered during conflict analysis bumped each time it appears			EVSIDS, with each literal encountered during conflict analysis bumped once, and cancelling literals bumped twice	
Learned constraint deletion	Based on LBD	Based on activity		Based on LBD of assigned literals, and on the activity in case of equal LBD	
Restarts	Based on recent LBDs	Static *PicoSAT*'s restart policy		Luby series with factor 100	

Appendix B

Description of the Experimental Settings

This appendix presents the experimental settings used throughout the thesis to evaluate the performance of the algorithms.

B.1 Machine Configuration

All experiments have been executed on a cluster of computers equipped with quadcore bi-processors Intel XEON X5550 (2.66 GHz, 8 MB cache) and 32 GB of memory. The nodes of this cluster run Linux CentOS 7 (x86_64), and use GCC 4.8.5 for C/C++-based programs and the JDK 11.0.1 for Java-based programs.

B.2 Benchmarks

Unless otherwise specified, our experiments consider the whole set of decision benchmarks containing “small” integers used in the pseudo-Boolean competitions since the first edition [MR06] as input, for a total of 5582 instances. From these instances, 4080 are from the `oliveras` family. To limit the bias of this large number of problems from the same family, we only consider instances that are *not easy*, i.e., for which at least one of the solvers took more than 1 minute to get a result. Other instances (i.e., instances solved in less than 1 minute by all solvers) are thus removed from the plots and tables presented in this thesis, while their number is specified in the analysis of the corresponding experiments.

Note that only instances using “small” integers are considered here. This limitation allows to execute solvers that do not support arbitrary precision arithmetic.

B.3 Description of the Benchmark Families

In this section, we provide the description of different benchmark families that have been used, when they are available.

B.3.1 Families `EC_ODD_GRIDS` and `EC_RANDOM_GRAPHS`

These families contain pseudo-Boolean encodings of the *even-colouring* problem [EGNV18]. Instances from these families encode that, given a graph G, there exists a black-and-white colouring of the edges of G such that every vertex is connected to the same number of black and white edges. These formulae are satisfiable if and only if G contains an even number of edges.

In the case of EC_ODD_GRIDS, the graphs G encoded in the formulae represent complete grids, in which every vertex is connected to its 4 neighbors. One of the edges is split to ensure that G contains an odd number of vertices, making the formulae unsatisfiable.

In the case of EC_RANDOM_GRAPH, the graph G encoded in the formulae is a random graph containing an even number of edges, with one of them split to make G contain an odd number of edges.

B.3.2 Family `liu`

This family contains instances of the *degree bounded spanning tree* problem. This problem, described in [Rou06], is as follows.

Let $G = (V, E)$ be an undirected graph, and let $w : V \times V \rightarrow \mathbb{N}$ be a function assigning non-negative integer weights to the edges of G. A *spanning tree* $T = (V, E')$ is a d*-bounded spanning tree* if, for every vertex $v \in V$, the following holds:

$$\sum_{(v,u)\in E} w(v, u) \leq d$$

The 15 instances of this problem are composed of 15 randomly generated graphs. More precisely:

- 5 graphs having 30 vertices and 350 edges with $d = 15$,
- 5 graphs having 40 vertices and 600 edges with $d = 20$, and
- 5 graphs having 50 vertices and 1000 edges with $d = 25$.

The weights of edges are between 1 and 9.

B.3.3 Family `tsp`

This family contains instances of the *travelling salesperson* problem. This problem, described in [Rou06], is as follows. There are n cities that are connected to each other. Each pair of cities is assigned a weight w, with $1 \leq w \leq n$, which is the cost associated with the travel between them. A satisfying assignment involves choosing the order in which the n cities are visited so that the sum of the weights associated with each adjacent pairs of cities that are visited is less than a given weight W.

This family contains both satisfiable and unsatisfiable instances. All problems are such that $n = 11$ and $W = 25$.

B.3.4 Family `vertexcover-instances`

This family contains instances of the *vertex cover* problem, used for instance in [EGNV18]. Formulae from this family encode that a graph $G = (V, E)$ has a vertex cover of size s, with different values of s. The graphs represented by these formulae are either grids or complete graphs.

B.3.5 Family `wnqueen`

This family contains instances of the *weighted* n*-queens* problem. This problem, described in [Rou06], is as follows. In the weighted n-queens problem, every square of a $n \times n$ chess board is assigned a weight w. Let w_i be the weight of the square assigned to the i-th queen, where $1 \leq i \leq n$. The n queens must be assigned to the squares so that the sum of the weights of the assigned squares must be less than or equal to a weight W, i.e., $\sum_{i=1}^{n} w_i \leq W$.

Two queens attack each other if and only if:

- they are in the same row, or
- they are in the same column, or
- they are in the same ascending diagonal, or
- they are in the same descending diagonal.

This family contains both satisfiable and unsatisfiable instances. All problems are such that $n = 13$ and $W = 38$.

B.3.6 Other Families

For the following families, we only briefly described what they encode.

- `FPGA_SAT05` and `uclid_pb_benchmarks` are industrial instances.
- `heinz` are pseudo-Boolean encodings of MIPLIB2010 problems.
- `ppp-problems` are pseudo-Boolean encodings of the progressive party problem described in [GH99].
- `lopes` are multiple constant multiplication problems.
- `oliveras` are pseudo-Boolean encodings of resource-constrained project scheduling problem.
- `robin` are pseudo-Boolean encodings of the travelling tournament problem, as described in [ENT01].
- `roussel` are pigeonhole principle formulae, either encoded using cardinality constraints (20 instances) or clauses (20 instances).
- `sroussel` are pseudo-Boolean constraints modelling visits of museums.
- `subsetcard` are *subset cardinality formulae*, as described in [EGNV18].
- `SUMINEQ` are *linearized pebbling formulae*, as described in [EGNV18].
- `nossum` are cryptography instances for SHA-1 made with Vegard Nossum's generator[8] with 21 to 23 rounds and 80 to 160 bits.
- `quimper` are pseudo-Boolean instances from [VQD16].

[8] https://github.com/vegard/sha1-sat

Index

Bibliography

[ABH$^+$08] G. AUDEMARD, L. BORDEAUX, Y. HAMADI, S. JABBOUR, and L. SAIS. « A Generalized Framework for Conflict Analysis ». In Hans KLEINE BÜNING and Xishun ZHAO, editors, *Theory and Applications of Satisfiability Testing – SAT 2008*, pages 21–27, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg. 6

[ACMS18] Antoine AMARILLI, Florent CAPELLI, Mikaël MONET, and Pierre SENELLART. « Connecting Knowledge Compilation Classes and Width Parameters ». *CoRR*, abs/1811.02944, 2018. 3.3, 3.3

[AFT11] Albert ATSERIAS, Johannes Klaus FICHTE, and Marc THURLEY. « Clause-Learning Algorithms with Many Restarts and Bounded-Width Resolution ». *J. Artif. Intell. Res.*, 40:353–373, 2011. 4.1.3

[AM05] Carlos ANSÓTEGUI and Felip MANYÀ. « Mapping Problems with Finite-Domain Variables to Problems with Boolean Variables ». In Holger H. HOOS and David G. MITCHELL, editors, *Theory and Applications of Satisfiability Testing*, pages 1–15, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg. 4.3

[AMS18] Antoine AMARILLI, Mikaël MONET, and Pierre SENELLART. « Connecting Width and Structure in Knowledge Compilation ». In *21st International Conference on Database Theory, ICDT 2018, March 26-29, 2018, Vienna, Austria*, 2018. 3.3

[ANORC11] Roberto ACHÁ, Robert NIEUWENHUIS, Albert OLIVERAS, and Enric RODRÍGUEZ-CARBONELL. « Cardinality Networks: A theoretical and empirical study ». *Constraints*, 16, 04 2011. 4.3

[AS09] Gilles AUDEMARD and Laurent SIMON. « Predicting Learnt Clauses Quality in Modern SAT Solvers ». In *Proceedings of IJCAI'09*, pages 399–404, 2009. 4.1.3, 7.2.1

[AS12] Gilles AUDEMARD and Laurent SIMON. « Refining Restarts Strategies for SAT and UNSAT ». In Michela MILANO, editor, *Principles and Practice of Constraint Programming*, pages 118–126, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. 4.1.3

[AS14] Ignasi ABÍO and Peter J. STUCKEY. « Encoding Linear Constraints into SAT ». In Barry O'SULLIVAN, editor, *Principles and Practice of Constraint Programming*, pages 75–91, Cham, 2014. Springer International Publishing. 4.3

[AS18] Gilles AUDEMARD and Laurent SIMON. « On the Glucose SAT Solver ». *Int. J. Artif. Intell. Tools*, 27(1):1840001:1–1840001:25, 2018. 7.2.3

[Bar95] P. BARTH. « A Davis-Putnam based Enumeration Algorithm for Linear Pseudo-Boolean Optimization ». Technical report MPI-I-95-2, Max-Planck Institut Für Informatik, 1995. 1, 4

[BB92] Michael BURO and Hans BÜNING. « Report on a SAT competition ». Technical report, Universität Paderborn, 11 1992. 4.1.3

[BCMS15] Simone BOVA, Florent CAPELLI, Stefan MENGEL, and Friedrich SLIVOVSKY. « On Compiling CNFs into Structured Deterministic DNNFs ». In *Theory and Applications of Satisfiability Testing - SAT 2015 - 18th International Conference*, 2015. 3, 3.3, 3.4, 3.4

[BCMS16] Simone BOVA, Florent CAPELLI, Stefan MENGEL, and Friedrich SLIVOVSKY. « Knowledge Compilation Meets Communication Complexity ». In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016*, 2016. 1.2, 2.2, 3, 3.3

[BF15] Armin BIERE and Andreas FRÖHLICH. « Evaluating CDCL Variable Scoring Schemes ». In Marijn HEULE and Sean WEAVER, editors, *Theory and Applications of Satisfiability Testing – SAT 2015*, pages 405–422, Cham, 2015. Springer International Publishing. 4.1.3

[BH02] Endre BOROS and Peter L. HAMMER. « Pseudo-Boolean optimization ». *Discrete Applied Mathematics*, 123(1):155 – 225, 2002. 7.3.2

[BHvMW09] Armin BIERE, Marijn HEULE, Hans van MAAREN, and Toby WALSH, editors. *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2009. I, 3.2, 3.5.1

[Bie08a] Armin BIERE. « Adaptive Restart Strategies for Conflict Driven SAT Solvers ». In Hans KLEINE BÜNING and Xishun ZHAO, editors, *Theory and Applications of Satisfiability Testing – SAT 2008*, pages 28–33, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg. 4.1.3

[Bie08b] Armin BIERE. « PicoSAT Essentials ». *J. Satisf. Boolean Model. Comput.*, 4(2-4):75–97, 2008. 4.1.3, 4.2.3, 7.2.3

[Bie09] Armin BIERE. Bounded Model Checking. In Armin BIERE, Marijn HEULE, Hans van MAAREN, and Toby WALSH, editors, *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, pages 457–481. IOS Press, 2009. (document)

[Bie16] Armin BIERE. « Splatz, Lingeling, Plingeling, Treengeling, YalSAT Entering the SAT Competition 2016 ». In Tomáš BALYO, Marijn HEULE, and Matti JÄRVISALO, editors, *Proc. of SAT Competition 2016 – Solver and Benchmark Descriptions*, volume B-2016-1 of *Department of Computer Science Series of Publications B*, pages 44–45. University of Helsinki, 2016. 4.1.3

[BJ10] Eli BEN-SASSON and Jan JOHANNSEN. « Lower Bounds for Width-Restricted Clause Learning on Small Width Formulas ». In Ofer STRICHMAN and Stefan SZEIDER, editors, *Theory and Applications of Satisfiability Testing - SAT 2010, 13th International Conference, SAT 2010, Edinburgh, UK, July 11-14, 2010. Proceedings*, volume 6175 of *Lecture Notes in Computer Science*, pages 16–29. Springer, 2010. 4.1.3

[BKM11] Irénée BRIQUEL, Pascal KOIRAN, and Klaus MEER. « On the expressive power of CNF formulas of bounded tree- and clique-width ». *Discrete Applied Mathematics*, 159(1):1–14, 2011. 3, 3.5.2, 3.5.2, 3.5.2

[BLLM14] Armin BIERE, Daniel LE BERRE, Emmanuel LONCA, and Norbert MANTHEY. « Detecting Cardinality Constraints in CNF ». In *Theory and Applications of Satisfiability Testing*, pages 285–301, 2014. 1.1.3, II, 7.3.2

[BM84a] Egon BALAS and Joseph B. MAZZOLA. « Nonlinear 0–1 programming: I. Linearization techniques ». *Mathematical Programming*, 30(1):1–21, 1984. 1

[BM84b] Egon BALAS and Joseph B. MAZZOLA. « Nonlinear 0–1 programming: II. Dominance relations and algorithms ». *Mathematical Programming*, 30(1):22–45, 1984. 1

[Bry86] Randal E. BRYANT. « Graph-Based Algorithms for Boolean Function Manipulation ». *IEEE Trans. Comput.*, 35(8):677–691, August 1986. 2.2

[BS94] Belaid BENHAMOU and Lakhdar SAIS. « Tractability Through Symmetries in Propositional Calculus ». *J. Autom. Reasoning*, 12(1):89–102, 1994. II

[BS17] Simone BOVA and Stefan SZEIDER. « Circuit Treewidth, Sentential Decision, and Query Compilation ». In *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2017*, pages 233–246, 2017. 3.3

[BSS94] Belaid BENHAMOU, Lakhdar SAIS, and Pierre SIEGEL. « Two Proof Procedures for a Cardinality Based Language in Propositional Calculus ». In Patrice ENJALBERT, Ernst W. MAYR, and Klaus W. WAGNER, editors, *STACS 94, 11th Annual Symposium on Theoretical Aspects of Computer Science, Caen, France, February 24-26, 1994, Proceedings*, volume 775 of *Lecture Notes in Computer Science*, pages 71–82. Springer, 1994. 2

[CCT87] W. COOK, C. R. COULLARD, and G. TURÁN. « On the Complexity of Cutting-plane Proofs ». *Discrete Appl. Math.*, 18(1):25–38, November 1987. (document), II, 30, 56

[CD97] Marco CADOLI and Francesco M. DONINI. « A Survey on Knowledge Compilation ». *AI Commun.*, 10(3,4):137–150, December 1997. (document), 1.3

[Che04] Hubie CHEN. « Quantified Constraint Satisfaction and Bounded Treewidth ». In *Proceedings of the 16th Eureopean Conference on Artificial Intelligence, ECAI 2004*, 2004. (document), 3

[CK05] Donald CHAI and Andreas KUEHLMANN. « A fast pseudo-Boolean constraint solver ». *IEEE Trans. on CAD of Integrated Circuits and Systems*, pages 305–317, 2005. (document), I, II, 4.2.1, 4.2.1, 28, 28, 7, 36, 19, 4.2.3, 4.2.3, 4.2.3

[CLH11] Yves CRAMA and Peter L. HAMMER. *Boolean Functions: Theory, Algorithms, and Applications*. Cambridge University Press, 2011. I, 1.1.2, 17, 5.3.3

[CLRS09] Thomas H. CORMEN, Charles E. LEISERSON, Ronald L. RIVEST, and Clifford STEIN. *Introduction to Algorithms, Third Edition*. MIT Press, 2009. 1.2, 4, 5.4.3

[CM19] Florent CAPELLI and Stefan MENGEL. « Tractable QBF by Knowledge Compilation ». In *36th International Symposium on Theoretical Aspects of Computer Science, STACS 2019*, volume 126, pages 18:1–18:16, 2019. 3, 3.1.4, 15, 3.4

[Coo71] Stephen A. COOK. « The Complexity of Theorem-proving Procedures ». In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, STOC '71, pages 151–158, New York, NY, USA, 1971. ACM. (document), 2, 4.1

[Cou12] Bruno COURCELLE. « On the model-checking of monadic second-order formulas with edge set quantifications ». *Discrete Applied Mathematics*, 160(6):866–887, 2012. 87

[CR79] Stephen A. COOK and Robert A. RECKHOW. « The Relative Efficiency of Propositional Proof Systems ». *J. Symb. Log.*, 44(1):36–50, 1979. 56

[DABC93] Olivier DUBOIS, Pascal ANDRÉ, Yacine BOUFKHAD, and Jacques CARLIER. « SAT versus UNSAT ». In David S. JOHNSON and Michael A. TRICK, editors, *Cliques, Coloring, and Satisfiability, Proceedings of a DIMACS Workshop, New Brunswick, New Jersey, USA, October 11-13, 1993*, volume 26 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 415–436. DIMACS/AMS, 1993. 4.1.3

[Dar04] Adnan DARWICHE. « New Advances in Compiling CNF to Decomposable Negation Normal Form ». In *Proceedings of the 16th European Conference on Artificial Intelligence*, ECAI'04, pages 318–322, NLD, 2004. IOS Press. 7.3.2

[DBBD16] Jo DEVRIENDT, Bart BOGAERTS, Maurice BRUYNOOGHE, and Marc DENECKER. « Improved Static Symmetry Breaking for SAT ». In *Proceedings of SAT'16*, pages 104–122, 2016. II

[Dev20] Jo DEVRIENDT. « Watched Propagation of 0-1 Integer Linear Constraints ». In Helmut SIMONIS, editor, *Principles and Practice of Constraint Programming - 26th International Conference, CP 2020, Louvain-la-Neuve, Belgium, September 7-11, 2020, Proceedings*, volume 12333 of *Lecture Notes in Computer Science*, pages 160–176. Springer, 2020. 28

[DG02] Heidi E. DIXON and Matthew L. GINSBERG. « Inference Methods for a Pseudo-Boolean Satisfiability Solver ». In *AAAI'02*, pages 635–640, 2002. (document), II, 4.2.1, 28, 4.2.3, 4.2.3, 7.1

[DGP04] Heidi E. DIXON, Matthew L. GINSBERG, and Andrew J. PARKES. « Generalizing Boolean Satisfiability I: Background and Survey of Existing Work ». *Journal of Artificial Intelligence Research*, pages 193–243, 2004. (document), I, 3

[DHJ$^+$04] Pavol DURIS, Juraj HROMKOVIC, Stasys JUKNA, Martin SAUERHOFF, and Georg SCHNITGER. « On multi-partition communication complexity ». *Inf. Comput.*, 194(1):49–75, 2004. 3

[Die12] Reinhard DIESTEL. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012. 3.1.1, 3.4

[DIM93] DIMACS. « Satisfiability: Suggested Format ». *DIMACS Challenge. DIMACS*, 1993. I

[Dix04] Heidi DIXON. « *Automating Pseudo-Boolean Inference Within a DPLL Framework* ». PhD thesis, University of Oregon, 2004. I, 34, 4.2.3, 63, 8, 7.1, 7.1

[DLL62] Martin DAVIS, George LOGEMANN, and Donald LOVELAND. « A Machine Program for Theorem-Proving ». *Commun. ACM*, 5(7):394–397, July 1962. 19

[DM02] Adnan DARWICHE and Pierre MARQUIS. « A Knowledge Compilation Map ». *J. Artif. Int. Res.*, 17(1):229–264, September 2002. (document), I, 1.3.1, 1.3.2, 1.2, 2, 2.1, 2.1, 2.2, 2.2, 2.2, 2.2, 2.3, 7.3.2

[DP60] Martin DAVIS and Hilary PUTNAM. « A Computing Procedure for Quantification Theory ». *J. ACM*, 7(3):201–215, July 1960. 4, 4.1.2, 19

[EGG$^+$18] Jan ELFFERS, Jesús GIRÁLDEZ-CRU, Stephan GOCHT, Jakob NORDSTRÖM, and Laurent SIMON. « Seeking Practical CDCL Insights from Theoretical SAT Benchmarks ». In Jérôme LANG, editor, *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden*, pages 1300–1308. ijcai.org, 2018. 4.1.3, 4.1.3, 5.4.3

[EGNV18] Jan ELFFERS, Jesús GIRÁLDEZ-CRÚ, Jakob NORDSTRÖM, and 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. II, 5.4.3, B.3.1, B.3.4, B.3.6

[EN18] Jan ELFFERS and 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. (document), I, II, 28, 19, 14, 4.2.3, 4.2.3, 5.2, 5.4.1, 6.1, 6.2, 42, 7.2.1, 7.2.1, 7.3.2, A, A.1

[EN20] Jan ELFFERS and Jakob NORDSTRÖM. « A Cardinal Improvement to Pseudo-Boolean Solving ». In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 1495–1503. AAAI Press, 2020. 1.1.3, II, 7.3.2

[ENT01] Kelly EASTON, George NEMHAUSER, and Michael TRICK. « The Traveling Tournament Problem Description and Benchmarks ». In Toby WALSH, editor, *Principles and Practice of Constraint Programming — CP 2001*, pages 580–584, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg. B.3.6

[ES04] Niklas EÉN and Niklas SÖRENSSON. « An Extensible SAT-solver ». In *Theory and Applications of Satisfiability Testing*, pages 502–518, 2004. (document), II, 4, 4.1, 4.1.1, 4.1.3, 4.1.3, 4.2.3

[ES06] Niklas EEN and Niklas SÖRENSSON. « Translating Pseudo-Boolean Constraints into SAT ». *JSAT*, 2:1–26, 03 2006. 1.1.3, 4.3, A.4

[FHZ19] Johannes Klaus FICHTE, Markus HECHER, and Markus ZISSER. « An Improved GPU-Based SAT Model Counter ». In Thomas SCHIEX and Simon de GIVRY, editors, *Principles and Practice of Constraint Programming - 25th International Conference, CP 2019, Stamford, CT, USA, September 30 - October 4, 2019, Proceedings*, volume 11802 of *Lecture Notes in Computer Science*, pages 491–509. Springer, 2019. 3.5.2

[FM09] John FRANCO and John MARTIN. A History of Satisfiability. In Armin BIERE, Marijn HEULE, Hans van MAAREN, and Toby WALSH, editors, *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, pages 3–74. IOS Press, 2009. 4

[FMR08] E. FISCHER, J.A. MAKOWSKY, and E.V. RAVVE. « Counting truth assignments of formulas of bounded tree-width or clique-width ». *Discrete Applied Mathematics*, 156(4):511–529, 2008. (document), 3, 3.1.2, 3.3, 3.4

[Fre95] Jon William FREEMAN. « *Improvements to Propositional Satisfiability Search Algorithms* ». PhD thesis, University of Pennsylvania, USA, 1995. 4.1.3

[Gel02] Allen Van GELDER. « Generalizations of Watched Literals for Backtracking Search ». In *International Symposium on Artificial Intelligence and Mathematics, AI&M 2002, Fort Lauderdale, Florida, USA, January 2-4, 2002*, 2002. 4.1.1

[GH99] Philippe GALINIER and Jin-Kao HAO. « *Solving the Progressive Party Problem by Local Search* », pages 419–432. Springer US, Boston, MA, 1999. B.3.6

[Gin93] Matthew L. GINSBERG. « Dynamic Backtracking ». *J. Artif. Intell. Res.*, 1:25–46, 1993. 4.1.3

[GJ79] Michael R. GAREY and David S. JOHNSON. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., USA, 1979. 1.2

[GKPS95] Goran GOGIC, Henry KAUTZ, Christos PAPADIMITRIOU, and Bart SELMAN. « The Comparative Linguistics of Knowledge Representation ». In *IJCAI'95*, pages 862–869, 1995. (document), I

[GMT02] Enrico GIUNCHIGLIA, Marco MARATEA, and Armando TACCHELLA. « Dependent and Independent Variables in Propositional Satisfiability ». In *Logics in Artificial Intelligence, JELIA 2002*, 2002. 68

[GN04] Ian P GENT and Peter NIGHTINGALE. « A new encoding of alldifferent into SAT ». In *International Workshop on Modelling and Reformulating Constraint Satisfaction Problems*, 2004. 3.2

[Gom58] Ralph E. GOMORY. « Outline of an algorithm for integer solutions to linear programs ». *Bulletin of the American Mathematical Society*, pages 275–278, 1958. (document), 3.5.2, II

[GPW10] Georg GOTTLOB, Reinhard PICHLER, and Fang WEI. « Bounded treewidth as a key to tractability of knowledge representation and reasoning ». *Artif. Intell.*, 174(1):105–132, 2010. 3

[GSK98] Carla P. GOMES, Bart SELMAN, and Henry KAUTZ. « Boosting Combinatorial Search through Randomization ». In *Proceedings of the Fifteenth National/Tenth Conference on Artificial Intelligence/Innovative Applications of Artificial Intelligence*, AAAI '98/IAAI '98, pages 431–437, USA, 1998. American Association for Artificial Intelligence. 4.1.3

[Hak85] Armin HAKEN. « The intractability of resolution ». *Theoretical Computer Science*, pages 297–308, 1985. (document), II, 4.1.3, 56

[HMS12] Steffen HÖLLDOBLER, Norbert MANTHEY, and Peter STEINKE. « A Compact Encoding of Pseudo-Boolean Constraints into SAT ». In Birte GLIMM and Antonio KRÜGER, editors, *KI 2012: Advances in Artificial Intelligence*, pages 107–118, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. 4.3

[Hoo88] John N. HOOKER. « Generalized resolution and cutting planes ». *Annals of Operations Research*, pages 217–239, 1988. (document), 3.5.2, II, 4.2.2, A.2

[Hro97] Juraj HROMKOVIC. *Communication Complexity and Parallel Computing*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 1997. 3

[Hua07] Jinbo HUANG. « The Effect of Restarts on the Efficiency of Clause Learning ». In Manuela M. VELOSO, editor, *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*, pages 2318–2323, 2007. 4.1.3, 4.2.3

[JLBR11] Matti JÄRVISALO, Daniel LE BERRE, and Olivier ROUSSEL. « *SAT Competition 2011* », 2011. 12

[JLRS12] Matti JÄRVISALO, Daniel LE BERRE, Olivier ROUSSEL, and Laurent SIMON. « The International SAT Solver Competitions ». *AI Magazine*, 33(1), 2012. (document), II, 4.1

[JPW09] Michael JAKL, Reinhard PICHLER, and Stefan WOLTRAN. « Answer-Set Programming with Bounded Treewidth ». In *Proceedings of the 21st International Joint Conference on Artificial Intelligence, IJCAI 2009*, 2009. 3

[Juk12] Stasys JUKNA. *Boolean Function Complexity - Advances and Frontiers*, volume 27 of *Algorithms and combinatorics*. Springer, 2012. 3

[JW90] Robert G. JEROSLOW and Jinchang WANG. « Solving Propositional Satisfiability Problems ». *Annals of Mathematics and Artificial Intelligence*, 1(1–4):167–187, September 1990. 4.1.3

[Kal17] Nikolay KALEYSKI. « PI is not at least as succinct as MODS ». In *2nd International Workshop on Boolean Functions and their Applications (BFA)*, 2017. 1.2

[KN97] Eyal KUSHILEVITZ and Noam NISAN. *Communication complexity*. Cambridge University Press, 1997. 3, 3.1.3, 36

[Knu08] Donald E. KNUTH. *The Art of Computer Programming, Volume IV, Fascicle 0: Introduction to Combinatorial Algorithms and Boolean Functions*. Addison-Wesley Professional, 1 edition, 2008. 1.1.2

[Knu16] Donald E. KNUTH. *The Art of Computer Programming, Volume IV, Fascicle 6: Satisfiability*. Addison-Wesley, 2016. 4.1.3

[Kra88] Matthias KRAUSE. « Exponential Lower Bounds on the Complexity of Local and Real-time Branching Programs ». *Elektronische Informationsverarbeitung und Kybernetik*, 24(3):99–110, 1988. 3.5.2

[Kro09] Daniel KROENING. Software Verification. In Armin BIERE, Marijn HEULE, Hans van MAAREN, and Toby WALSH, editors, *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, pages 505–532. IOS Press, 2009. (document)

[LGPC16] Jia Hui LIANG, Vijay GANESH, Pascal POUPART, and Krzysztof CZARNECKI. « Learning Rate Based Branching Heuristic for SAT Solvers ». In Nadia CREIGNOU and Daniel LE BERRE, editors, *Theory and Applications of Satisfiability Testing – SAT 2016*, pages 123–140, Cham, 2016. Springer International Publishing. 4.1.3

[LM12] Mark H. LIFFITON and Jordyn C. MAGLALANG. « A Cardinality Solver: More Expressive Constraints for Free ». In Alessandro CIMATTI and Roberto SEBASTIANI, editors, *Theory and Applications of Satisfiability Testing – SAT 2012*, pages 485–486, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. 19

[LM17] Jean-Marie LAGNIEZ and Pierre MARQUIS. « An Improved Decision-DNNF Compiler ». In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, pages 667–673, 2017. 7.3.2

[LMMW18] Daniel LE BERRE, Pierre MARQUIS, Stefan MENGEL, and Romain WALLON. « Pseudo-Boolean Constraints from a Knowledge Representation Perspective ». In Jérôme LANG, editor, *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden*, pages 1891–1897. ijcai.org, 2018. (document), I, 7.3.2

[LMMW20] Daniel LE BERRE, Pierre MARQUIS, Stefan MENGEL, and Romain WALLON. « On Irrelevant Literals in Pseudo-Boolean Constraint Learning ». In Christian BESSIERE, editor, *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI 2020*, pages 1148–1154. ijcai.org, 2020. (document), II, 5, 7.3.2, 7.3.2

[LMW20] Daniel LE BERRE, Pierre MARQUIS, and Romain WALLON. « On Weakening Strategies for PB Solvers ». In Luca PULINA and Martina SEIDL, editors, *Theory and Applications of Satisfiability Testing - SAT 2020 - 23rd International Conference, Alghero, Italy, July 3-10, 2020, Proceedings*, volume 12178 of *Lecture Notes in Computer Science*, pages 322–331. Springer, 2020. (document), II, 6, 6.4, 6.1, 6.1, 7.2.1, 7.3.2, 7.3.2

[LP10] Daniel LE BERRE and Anne PARRAIN. « The SAT4J library, Release 2.2, System Description ». *Journal on Satisfiability, Boolean Modeling and Computation*, pages 59–64, 2010. (document), I, II, 4.1.3, 4.2.1, 28, 28, 14, 4.2.3, 4.3, 5.2, 5.4.2, 6.1, 7.3.2, A, A.2

[LSZ93] Michael LUBY, Alistair SINCLAIR, and David ZUCKERMAN. « Optimal Speedup of Las Vegas Algorithms ». *Inf. Process. Lett.*, 47(4):173–180, September 1993. 4.1.3, 4.2.3

[Mar99] João P. MARQUES-SILVA. « The Impact of Branching Heuristics in Propositional Satisfiability Algorithms ». In *Proceedings of the 9th Portuguese Conference on Artificial Intelligence: Progress in Artificial Intelligence*, EPIA ’99, pages 62–74, Berlin, Heidelberg, 1999. Springer-Verlag. 4.1.3

[MBK19] Hakan METIN, Souheib BAARIR, and Fabrice KORDON. « Composing Symmetry Propagation and Effective Symmetry Breaking for SAT Solving ». In *Proceedings of NFM’19*, pages 316–332, 2019. II

[MMBH12] Christian MUISE, Sheila A. MCILRAITH, J. Christopher BECK, and Eric I. HSU. « Dsharp: Fast d-DNNF Compilation with sharpSAT ». In Leila KOSSEIM and Diana INKPEN, editors, *Advances in Artificial Intelligence*, pages 356–361, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. 7.3.2

[MML14] Ruben MARTINS, Vasco MANQUINHO, and Inês LYNCE. « Open-WBO: A Modular MaxSAT Solver ». In Carsten SINZ and Uwe EGLY, editors, *SAT’14*, pages 438–445, 2014. 1.1.3, 4.3, A.4

[MMZ+01] Matthew W. MOSKEWICZ, Conor F. MADIGAN, Ying ZHAO, Lintao ZHANG, and Sharad MALIK. « Chaff: Engineering an Efficient SAT Solver ». In *Proceedings of the 38th Annual Design Automation Conference*, pages 530–535, 2001. (document), II, 4, 4.1, 4.1.1, 4.1.1, 6, 4.1.3

[MR06] Vasco MANQUINHO and Olivier ROUSSEL. « The First Evaluation of Pseudo-Boolean Solvers (PB'05) ». *JSAT*, pages 103–143, 2006. 28, B.2

[MS97] João P. MARQUES-SILVA and Karem A. SAKALLAH. « Robust Search Algorithms for Test Pattern Generation ». In *Digest of Papers: FTCS-27, The Twenty-Seventh Annual International Symposium on Fault-Tolerant Computing, Seattle, Washington, USA, June 24-27, 1997*, pages 152–161. IEEE Computer Society, 1997. I, 4

[MS99] João P. MARQUES-SILVA and Karem A. SAKALLAH. « GRASP: A Search Algorithm for Propositional Satisfiability ». *IEEE Trans. Computers*, pages 220–227, 1999. (document), II, 4, 4.1, 13, 13

[MW19] Stefan MENGEL and Romain WALLON. « Revisiting Graph Width Measures for CNF-Encodings ». In Mikolás JANOTA and Inês LYNCE, editors, *Theory and Applications of Satisfiability Testing - SAT 2019 - 22nd International Conference, SAT 2019, Lisbon, Portugal, July 9-12, 2019, Proceedings*, volume 11628 of *Lecture Notes in Computer Science*, pages 222–238. Springer, 2019. (document), I, 7.3.2

[MW20] Stefan MENGEL and Romain WALLON. « Graph Width Measures for CNF-Encodings with Auxiliary Variables ». *J. Artif. Intell. Res.*, 67:409–436, 2020. (document), I, 7.3.2

[Nad19] Alexander NADEL. « Anytime Weighted MaxSAT with Improved Polarity Selection and Bit-Vector Optimization ». In Clark W. BARRETT and Jin YANG, editors, *2019 Formal Methods in Computer Aided Design, FMCAD 2019, San Jose, CA, USA, October 22-25, 2019*, pages 193–202. IEEE, 2019. 7.3.2

[NKR+18] Nina NARODYTSKA, Shiva Prasad KASIVISWANATHAN, Leonid RYZHYK, Mooly SAGIV, and Toby WALSH. « Verifying Properties of Binarized Deep Neural Networks ». In Sheila A. MCILRAITH and Kilian Q. WEINBERGER, editors, *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 6615–6624. AAAI Press, 2018. 7.3.2

[Nor15] Jakob NORDSTRÖM. « On the Interplay Between Proof Complexity and SAT Solving ». *ACM SIGLOG News*, 2(3):19–44, August 2015. II, 56

[Nor20] Jakob NORDSTRÖM. « *Jakob Nordström's Video Seminars* », 2020. II

[Pap94] Christos H. PAPADIMITRIOU. *Computational complexity.* Addison-Wesley, 1994. 1.2

[PD07] Knot PIPATSRISAWAT and Adnan DARWICHE. « RSat 2.0: SAT Solver Description ». Technical report, University of California, Los Angeles, 2007. 4.1.3

[PD08] Knot PIPATSRISAWAT and Adnan DARWICHE. « New Compilation Languages Based on Structured Decomposability ». In *Proceedings of the 23rd National Conference on Artificial Intelligence - Volume 1*, AAAI'08, pages 517–522. AAAI Press, 2008. 81

[PD10] Thammanit PIPATSRISAWAT and Adnan DARWICHE. « A Lower Bound on the Size of Decomposable Negation Normal Form ». In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010*, 2010. 3, 3.3, 3.3

[PD11] Knot PIPATSRISAWAT and Adnan DARWICHE. « On the power of clause-learning SAT solvers as resolution engines ». *Artif. Intell.*, 175(2):512–525, 2011. 4.1.3

[PG86] David A. PLAISTED and Steven GREENBAUM. « A Structure-preserving Clause Form Translation ». *Journal of Symbolic Computation*, 2(3):293 – 304, 1986. (document), 1.1.3

[PS15] Tobias PHILIPP and Peter STEINKE. PBLib – A Library for Encoding Pseudo-Boolean Constraints into CNF. In Marijn HEULE and Sean WEAVER, editors, *Theory and Applications of Satisfiability Testing – SAT 2015*, volume 9340 of *Lecture Notes in Computer Science*, pages 9–16. Springer International Publishing, 2015. 4.3

[PSS13] Daniël PAULUSMA, Friedrich SLIVOVSKY, and Stefan SZEIDER. « Model Counting for CNF Formulas of Bounded Modular Treewidth ». In *30th International Symposium on Theoretical Aspects of Computer Science, STACS 2013*, 2013. 3.3

[PSS16] Daniël PAULUSMA, Friedrich SLIVOVSKY, and Stefan SZEIDER. « Model Counting for CNF Formulas of Bounded Modular Treewidth ». *Algorithmica*, 76(1):168–194, 2016. (document), 3

[Rin09] Jussi RINTANEN. Planning and SAT. In Armin BIERE, Marijn HEULE, Hans van MAAREN, and Toby WALSH, editors, *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, pages 483–504. IOS Press, 2009. (document)

[RM09a] Olivier ROUSSEL and Vasco M. MANQUINHO. Pseudo-Boolean and Cardinality Constraints. In Armin BIERE, Marijn HEULE, Hans van MAAREN, and Toby WALSH, editors, *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, pages 695–733. IOS Press, 2009. (document), II, 4, 39

[RM09b] Olivier ROUSSEL and Vasco M. MANQUINHO. Pseudo-Boolean and Cardinality Constraints. In Armin BIERE, Marijn HEULE, Hans van MAAREN, and Toby WALSH, editors, *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, pages 695–733. IOS Press, 2009. 1

[Rou06] Olivier ROUSSEL. « *Pseudo-Boolean Competition 2006* », 2006. B.3.2, B.3.3, B.3.5

[Rou16] Olivier ROUSSEL. « *Pseudo-Boolean Competition 2016* », 2016. A

[Rya04] L. RYAN. « Efficient algorithms for clause-learning SAT solvers », 2004. 4.1.3

[SAT99] SATLIB. « *The Satisfiability Library* », 1999. http://www.cs.ubc.ca/ hoos/SATLIB/index-ubc.html. I

[Sin05] Carsten SINZ. « Towards an Optimal CNF Encoding of Boolean Cardinality Constraints ». In *Proceedings of the 11th International Conference on Principles and Practice of Constraint Programming*, CP'05, pages 827–831, Berlin, Heidelberg, 2005. Springer-Verlag. 3.5.1, 3.5.1

[SM20] Arijit SHAW and Kuldeep S. MEEL. « Designing New Phase Selection Heuristics ». In Luca PULINA and Martina SEIDL, editors, *Theory and Applications of Satisfiability Testing - SAT 2020 - 23rd International Conference, Alghero, Italy, July 3-10, 2020, Proceedings*, volume 12178 of *Lecture Notes in Computer Science*, pages 72–88. Springer, 2020. 4.1.3

[Sma07] Jan-Georg SMAUS. « On Boolean Functions Encodable as a Single Linear Pseudo-Boolean Constraint ». In Pascal VAN HENTENRYCK and Laurence WOLSEY, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 288–302, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg. 2.1

[SN15] Masahiko SAKAI and Hidetomo NABESHIMA. « Construction of an ROBDD for a PB-Constraint in Band Form and Related Techniques for PB-Solvers ». *IEICE Transactions on Information and Systems*, E98.D:1121–1127, 06 2015. 1.1.3, 4.3, A.4

[SS06] Hossein M. SHEINI and Karem A. SAKALLAH. « Pueblo: A Hybrid Pseudo-Boolean SAT Solver ». *JSAT*, pages 165–189, 2006. (document), I, II, 28, 28, 4.2.3, 4.2.3, 4.2.3, 7.1

[SS10a] Marko SAMER and Stefan SZEIDER. « Algorithms for propositional model counting ». *J. Discrete Algorithms*, 8(1):50–64, 2010. (document), 3, 3.1.2, 3.3

[SS10b] Marko SAMER and Stefan SZEIDER. « Constraint satisfaction with bounded treewidth revisited ». *J. Comput. Syst. Sci.*, 76(2):103–114, 2010. 3.4

[SS13] Friedrich SLIVOVSKY and Stefan SZEIDER. « Model Counting for Formulas of Bounded Clique-Width ». In *Algorithms and Computation - 24th International Symposium, ISAAC 2013*, 2013. (document), 3, 3.1.2, 3.1.2, 3.3, 3.4

[Stu10] Peter J. STUCKEY. « Lazy Clause Generation: Combining the Power of SAT and CP (and MIP?) Solving ». In Andrea LODI, Michela MILANO, and Paolo TOTH, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 5–9, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. 4.3

[STV15] Sigve Hortemo SÆTHER, Jan Arne TELLE, and Martin VATSHELLE. « Solving #SAT and MAXSAT by Dynamic Programming ». *J. Artif. Intell. Res.*, 54:59–82, 2015. (document), 3, 3.1.2, 3.3

[Tse68] G. S. TSEITIN. « On the complexity of derivation in propositional calculus ». *Structures in Constructive Mathematics and Mathematical Logic*, pages 115–125, 1968. (document), 1.1.3

[Vat12] Martin VATSHELLE. « *New Width Parameters of Graphs* ». PhD thesis, University of Bergen, 2012. 3.1.2, 3.4

[VEG+18] Marc VINYALS, Jan ELFFERS, Jesús GIRÁLDEZ-CRÚ, Stephan GOCHT, and 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. (document), II, 4.1.3, 4.3

[Vol99] Heribert VOLLMER. Complexity Measures and Reductions. In *Introduction to Circuit Complexity: A Uniform Approach*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1999. 2.2

[VQD16] Dany VOHL, Claude-Guy QUIMPER, and Danny DUBÉ. « Finding Synchronization Codes to Boost Compression by Substring Enumeration ». *CoRR*, abs/1605.08102, 2016. B.3.6

[Wal97] Joachim WALSER. « Solving Linear Pseudo-Boolean Constraint Problems with Local Search ». In *Proceedings of the National Conference on Artificial Intelligence*, pages 269–274, 01 1997. 4

[Wal20] Romain WALLON. « On Adapting CDCL Strategies for PB Solvers ». In *11th Workshop on Pragmatics of SAT - POS 2020*, 2020. II, 7.3.2

[Weg00] Ingo WEGENER. *Branching Programs and Binary Decision Diagrams*. SIAM, 2000. 3, 3.5.2, 3.5.2

[WS01] Jesse WHITTEMORE and Joonyoung Kim Karem A. SAKALLAH. « SATIRE: A New Incremental Satisfiability Engine ». In *Proceedings of the 38th Design Automation Conference, DAC 2001, Las Vegas, NV, USA, June 18-22, 2001*, pages 542–545, 2001. I, 4, 4.3, 6.1

[Zha09] Hantao ZHANG. Combinatorial Designs by SAT Solvers. In Armin BIERE, Marijn HEULE, Hans van MAAREN, and Toby WALSH, editors, *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, pages 533–568. IOS Press, 2009. (document)

[ZMMM01] Lintao ZHANG, Conor F. MADIGAN, Matthew W. MOSKEWICZ, and Sharad MALIK. « Efficient Conflict Driven Learning in Boolean Satisfiability Solver ». In Rolf ERNST, editor, *Proceedings of the 2001 IEEE/ACM International Conference on Computer-Aided Design, ICCAD 2001, San Jose, CA, USA, November 4-8, 2001*, pages 279–285. IEEE Computer Society, 2001. 6

[ZS96] Hantao ZHANG and Mark E. STICKEL. « An Efficient Algorithm for Unit Propagation ». In *In Proceedings of the Fourth International Symposium on Artificial Intelligence and Mathematics (AI-MATH'96), Fort Lauderdale (Florida USA*, pages 166–169, 1996. 4.1.1

[ZS00] Hantao ZHANG and Mark STICKEL. « Implementing the Davis–Putnam Method ». *J. Autom. Reason.*, 24(1–2):277–296, February 2000. 4.1.1

Résumé

Cette thèse porte sur le langage des conjonctions de contraintes pseudo-booléennes, formées d'équations ou inéquations linéaires en variables booléennes. Ce format généralise le format CNF très répandu. Notre contribution à l'étude des contraintes pseudo-booléennes se compose de deux parties.

Dans une première partie, nous étudions les contraintes pseudo-booléennes du point de vue de la représentation des connaissances. Nous comparons leur langage à différents formats propositionnels largement utilisés en évaluant chacun d'eux à l'aune des critères de la carte de compilation proposée par Darwiche et Marquis. Cette comparaison montre, d'un côté, l'un des avantages des contraintes pseudo-booléennes par rapport aux formules CNF : elles peuvent être beaucoup plus concises. Malheureusement, d'un autre côté, les opérations qui sont NP-difficiles pour les contraintes pseudo-booléennes sont plus nombreuses que pour les formules CNF. En conséquence, il est parfois préférable d'utiliser des encodages CNF (au lieu d'utiliser des représentations) de ces contraintes. De cette manière, l'efficacité en pratique des solveurs SAT modernes peut être mise à profit pour raisonner sur de tels encodages. Nous étudions également les propriétés de ces encodages, et montrons que borner leur largeur peut fortement réduire leur expressivité.

L'une des raisons pour lesquelles il est intéressant d'utiliser des contraintes pseudo-booléennes en plus des formules CNF est qu'elles permettent d'utiliser le système de preuve des plans coupes, qui est plus puissant que celui de la résolution implanté dans les solveurs SAT classiques pour raisonner sur des formules CNF. En particulier, des preuves d'incohérence produites par ce système de preuve peuvent être exponentiellement plus courtes que leur équivalent dans le système de la résolution pour certains ensembles de contraintes. Ainsi, d'un point de vue théorique, les solveurs pseudo-booléens, capables de prendre en compte nativement les contraintes pseudo-booléennes, pourraient être plus efficaces que les solveurs SAT. Cependant, en pratique, les solveurs pseudo-booléens ne tiennent pas cette promesse, et il est important de comprendre pourquoi.

Comme une étape dans cette direction, la deuxième partie de cette thèse est consacrée à la résolution pratique de problèmes pseudo-booléens. Nous identifions plusieurs faiblesses des solveurs pseudo-booléens, et proposons des améliorations. En particulier, nous montrons que toutes les règles implantées par les solveurs pseudo-booléens peuvent produire, pendant l'analyse de conflit, des littéraux non pertinents, c'est-à-dire des littéraux dont la valeur de vérité n'influe pas sur celle des contraintes dans lesquelles ils apparaissent. Ce comportement est problématique, puisque ces littéraux conduisent à l'inférence de contraintes plus faibles, et donc à la production de preuves d'incohérence plus longues. Nous proposons plusieurs approches pour corriger ces faiblesses, comme l'utilisation de nouvelles stratégies d'affaiblissement, visant à établir un compromis entre la taille des contraintes produites et leur force. Ces stratégies peuvent permettre d'éliminer efficacement des littéraux non pertinents, même si elles produisent des contraintes plus faibles. Nous montrons par ailleurs qu'un affaiblissement plus parcimonieux (mais inévitable) peut permettre d'améliorer les performances du solveur.

De plus, nous adaptons au cadre pseudo-booléen un certain nombre de variantes de stratégies implantées par les solveurs SAT classiques. En particulier, nous présentons des heuristiques et des mesures pour la qualité des contraintes apprises dédiées aux contraintes pseudo-booléennes. Nous montrons qu'un choix avisé parmi ces stratégies peut permettre d'améliorer considérablement les performances du solveur.

Toutes les approches pratiques décrites dans cette thèse ont été implantées dans le solveur pseudo-booléen *Sat4j* et sont publiquement accessibles.

Abstract

This thesis is about the language of pseudo-Boolean constraints, i.e., of conjunctions of linear equations or inequations over Boolean variables, that generalizes the well-known CNF format. Our contribution to the study of this language is twofold.

In the first part of the thesis, the language of pseudo-Boolean constraints is investigated from a knowledge representation perspective. It is compared to a number of well-known propositional formats using the criteria considered in the so-called knowledge compilation map pointed out by Darwiche and Marquis. On the one hand, an advantage of using pseudo-Boolean constraints as a representation language over CNF formulae is that they can be far more succinct. On the other hand, unfortunately, more operations of interest are NP-hard when dealing with pseudo-Boolean constraints compared to CNF formulae. As a consequence, it is sometimes preferable to consider CNF encodings (instead of CNF representations) of such constraints. Doing so, the practical efficiency of modern SAT solvers can be leveraged to reason with these encodings. We also investigate the properties of such encodings, and show that bounding their width may drastically reduce their expressivity.

One reason for considering pseudo-Boolean constraints instead of CNF formulae is that they allow the use of the cutting planes proof system, which is stronger than the resolution proof system implemented in classical SAT solvers based on CNF. In particular, unsatisfiability proofs that may be exponentially shorter than their resolution counterpart can be derived for some sets of pseudo-Boolean constraints. Thus, from the theory side, pseudo-Boolean solvers, i.e., solvers that use pseudo-Boolean constraints natively instead of encodings, could be more efficient than SAT solvers. However, practical solver implementations fail to keep this promise, and understanding why this is the case is an important issue.

As a step in this direction, the second part of this thesis deals with the practical resolution of pseudo-Boolean problems. Several weaknesses of current implementations of pseudo-Boolean solvers are identified and for some of them improvements are proposed. In particular, we show that all rules implemented in pseudo-Boolean solvers may produce irrelevant literals during conflict analysis, i.e., literals that have no effect on the truth value of the constraints in which they appear. This is problematic as such literals lead to the inference of constraints that can be weaker than expected, and in turn to the production of longer unsatisfiability proofs. We point out several approaches to deal with those weak points, such as the use of new weakening strategies, that are designed in such a way that a tradeoff between the size of the derived constraints and their strength is obtained. Those weakening strategies may allow to efficiently eliminate irrelevant literals, at the price of deriving weaker constraints. In addition, we show that a more parsimonious (but inevitable) application of the weakening rule may allow improving the performance of the solver.

We also lift many variants of strategies that are classically implemented in SAT solvers to the pseudo-Boolean setting. In particular, branching heuristics and quality measures for learned constraints that are adapted to pseudo-Boolean constraints are presented. We show that a careful choice among those strategies may drastically improve the performance of the solver.

All practical approaches described in this thesis have been implemented in the pseudo-Boolean solver *Sat4j* and are publicly available.