Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence (IJCAI-23)
Sister Conferences Best Papers Track

Certified CNF Translations for Pseudo-Boolean Solving (Extended Abstract)*

Stephan Gocht[1,2], **Ruben Martins**[3], **Jakob Nordström**[2,1] and **Andy Oertel**[1,2]
[1]Lund University
[2]University of Copenhagen
[3]Carnegie Mellon University
{stephan.gocht, andy.oertel}@cs.lth.se, jn@di.ku.dk, rubenm@andrew.cmu.edu

Abstract

The dramatic improvements in Boolean satisfiability (SAT) solving since the turn of the millennium have made it possible to leverage conflict-driven clause learning (CDCL) solvers for many combinatorial problems in academia and industry, and the use of proof logging has played a crucial role in increasing the confidence that the results these solvers produce are correct. However, the fact that SAT proof logging is performed in conjunctive normal form (CNF) clausal format means that it has not been possible to extend guarantees of correctness to the use of SAT solvers for more expressive combinatorial paradigms, where the first step is an unverified translation of the input to CNF.

In this work, we show how cutting-planes-based reasoning can provide proof logging for solvers that translate pseudo-Boolean (a.k.a. 0-1 integer linear) decision problems to CNF and then run CDCL. We are hopeful that this is just a first step towards providing a unified proof logging approach that will extend to maximum satisfiability (MaxSAT) solving and pseudo-Boolean optimization in general.

1 Introduction

Boolean satisfiability (SAT) solving has witnessed striking improvements over the last decades, starting with the introduction of *conflict-driven clause learning (CDCL)* [Marques-Silva and Sakallah, 1999; Moskewicz *et al.*, 2001], and this has led to a wide range of applications to large-scale problems in both academia and industry [Biere *et al.*, 2021]. The conflict-driven paradigm has also been successfully exported to other areas such as *maximum satisfiability (MaxSAT)*, *pseudo-Boolean (PB) solving*, *constraint programming (CP)*, and *mixed integer programming (MIP)*. As combinatorial solvers are used to attack ever more challenging problems, the question arises whether we can trust the results they produce. Sadly, it is well-documented that state-of-the-art CP and MIP solvers can return incorrect solutions [Akgün *et al.*, 2018; Cook *et al.*, 2013; Gillard *et al.*, 2019]. For SAT solvers, however, analogous problems [Brummayer *et al.*, 2010] have been successfully addressed by the introduction of *proof logging*, requiring that solvers should be *certifying* [McConnell *et al.*, 2011] in the sense that they output machine-verifiable proofs of their claims.

A number of different proof formats have been developed for SAT solving, and since 2013 the SAT competitions require solvers to be certifying, with *DRAT* [Wetzler *et al.*, 2014] established as the standard format. Such proof logging would be highly desirable also for stronger combinatorial solving paradigms, but while methods such as *DRAT* are extremely powerful in theory, the limitation to a clausal format makes it hard to capture more advanced forms of reasoning concisely. A more fundamental concern is how these proof logging methods should deal with input that is not presented in conjunctive normal form (CNF). One way to address this problem could be to extend the *DRAT* format [Baek *et al.*, 2021], but another approach pursued in recent years is to develop stronger proof logging methods based on more expressive formalisms, such as binary decision diagrams [Barnett and Biere, 2021], algebraic reasoning [Kaufmann *et al.*, 2022; Kaufmann and Biere, 2021], pseudo-Boolean reasoning [Elffers *et al.*, 2020; Gocht *et al.*, 2020; Gocht and Nordström, 2021; Bogaerts *et al.*, 2022], and integer linear programming [Cheung *et al.*, 2017; Eifler and Gleixner, 2021].

In this work, we consider the use of CDCL for pseudo-Boolean solving, where the pseudo-Boolean input (i.e., a 0-1 integer linear program) is translated to CNF and passed to a SAT solver, as pioneered in MINISAT+ [Eén and Sörensson, 2006]. While *DRAT* proof logging can certify unsatisfiability of the translated formula, it cannot prove correctness of the translation, not only since there is no known method of carrying out PB reasoning efficiently in *DRAT* (except for constraints with small coefficients [Bryant *et al.*, 2022]), but also, and more fundamentally, because the input is not in CNF.

We demonstrate how to instead use the *cutting planes* proof system [Cook *et al.*, 1987], enhanced with a rule for introducing extension variables [Gocht and Nordström, 2021], to show that the CNF formula resulting from the translation can be derived from the original pseudo-Boolean constraints. Since this method is a strict extension of *DRAT*, we can combine the proof for the translation with the SAT solver *DRAT* proof log. In this way, we achieve end-to-end

*This is an abbreviated version of a paper published in the *Proceedings of the 25th International Conference on Theory and Applications of Satisfiability Testing (SAT '22)*.

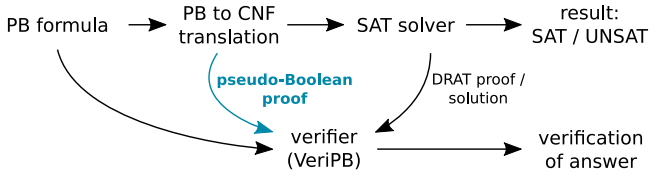


Figure 1: Proof logging workflow for pseudo-Boolean solving with our contribution highlighted in blue boldface.

verification of the pseudo-Boolean solving process using the proof checker VERIPB [Gocht and Nordström, 2021; Bogaerts *et al.*, 2022] as illustrated in Figure 1.

One challenge when certifying PB-to-CNF translations is that there are many different ways of encoding pseudo-Boolean constraints into CNF (as catalogued in, e.g., [Philipp and Steinke, 2015]), and it is time-consuming (and error-prone) to code up proof logging for every single encoding. However, many of the encodings can be understood as first designing a circuit to evaluate whether the PB constraint is satisfied, and then writing down a CNF formula enforcing the computation of this circuit. An important part of our contribution is that we develop a general proof logging method for a wide class of such circuits. The PB format makes it easy to derive 0-1 linear inequalities describing the circuit computations, and once this has been done the clauses in the CNF translation can simply be obtained by so-called *reverse unit propagation (RUP)* [Goldberg and Novikov, 2003; Van Gelder, 2008], obviating the need for complicated syntactic proofs. We apply this method to the *sequential counter* [Sinz, 2005], *totalizer* [Bailleux and Boufkhad, 2003], *generalized totalizer* [Joshi *et al.*, 2015] and *binary adder network* [Eén and Sörensson, 2006; Warners, 1998] encodings, and report results from an empirical evaluation of the efficiency of proof generation and verification.

This paper is only an abbreviated version of the *SAT '22* conference paper [Gocht *et al.*, 2022]. We refer the reader to the *SAT '22* conference paper or the upcoming full-length version for all the details omitted below.

2 Example: Sequential Counter Encoding

In the interest of brevity, let us give a concrete example of how to certify CNF translations of cardinality constraints using the *sequential counter* encoding [Sinz, 2005], introducing the necessary terminology and notation as we go along (and referring the reader to, e.g., [Buss and Nordström, 2021; Gocht and Nordström, 2021] for more detailed background).

A *literal* ℓ over a Boolean variable x is x itself or its negation $\overline{x}$, where variables can be assigned values 0 (false) or 1 (true), so that $\overline{x} = 1 - x$. A *pseudo-Boolean (PB) constraint* C is a 0-1 linear inequality $\sum_i a_i \ell_i \geq A$, which, without loss of generality, can be assumed to be in *normalized form* [Barth, 1995]; i.e., all literals ℓ_i are over distinct variables and the coefficients a_i and the *degree (of falsity)* A are non-negative integers. We use equality constraints $\sum_i a_i \ell_i = A$ as syntactic sugar for the corresponding pair of inequalities. A *pseudo-Boolean formula* is a conjunction of PB constraints. A *cardinality constraint* is a PB constraint with all coefficients equal to 1. If the degree is also 1, then the constraint $\ell_1 + \cdots + \ell_k \geq 1$ is equivalent to the *(disjunctive) clause* $\ell_1 \vee \cdots \vee \ell_k$, and so a CNF formula is just a special case of a pseudo-Boolean formula.

To convert a cardinality constraint $\sum_{i=1}^{n} \ell_i \bowtie k$ (where $\bowtie$ denotes $\geq$, $\leq$, or $=$) to CNF using the sequential counter encoding, one constructs a circuit summing up the input bits one by one, with intermediate variables $s_{i,j}$ for $i \in [n]$ and $j \in [i]$ being true if and only if $\sum_{t=1}^{i} \ell_t \geq j$ holds. The variables $s_{i,j}$ can be computed as in Figure 2a by the formula

$$s_{i,j} \leftrightarrow ((\ell_i \wedge s_{i-1,j-1}) \vee s_{i-1,j}) \tag{1}$$

saying that $s_{i,j}$ is true either if the first $i-1$ literals add up to $j-1$ and the ith literal is true, or if already the first $i-1$ literals add up to j. The circuit constructed in this way, shown in Figure 2b, can be partitioned into n blocks, where the ith block computes $s_{i,j}$ for $j \in [i]$ from the ith input bit ℓ_i and the variables $s_{i-1,j}$ in the previous block. One then obtains the CNF encoding by translating each component in Figure 2a (as described by Equation (1)) to the clausal constraints

$$\overline{\ell}_i + \overline{s}_{i-1,j-1} + s_{i,j} \geq 1 \tag{2a}$$
$$\overline{s}_{i-1,j} + s_{i,j} \geq 1 \tag{2b}$$
$$\ell_i + s_{i-1,j} + \overline{s}_{i,j} \geq 1 \tag{2c}$$
$$s_{i-1,j-1} + \overline{s}_{i,j} \geq 1 \tag{2d}$$

for $i \in [n]$ and $j \in [i]$ (where we write these disjunctive clauses in the pseudo-Boolean form that will be used in our proofs). For all i one sets $s_{i,0} = 1$, so that constraint (2a) simplifies to $\overline{\ell}_i + s_{i,1} \geq 1$ and constraint (2d) is satisfied and disappears. In the same way, $s_{i-1,i} = 0$ simplifies (2c) to $\ell_i + \overline{s}_{i,i} \geq 1$, and (2b) is satisfied and disappears. Once clauses (2a)–(2d) have been generated for all circuit components, one obtains a greater-than-or-equal-to-k constraint by adding the *unit clause* $s_{n,k} \geq 1$. Analogously, a less-than-or-equal-to-k constraint is enforced using the clause $\overline{s}_{n,k+1} \geq 1$.

Our goal is to produce a cutting planes proof that the resulting CNF formula correctly encodes the original cardinality constraint. Since $s_{i,j}$ is true if and only if $\sum_{t=1}^{i} \ell_t \geq j$ holds, for all $i \in [n]$ we should be able to deduce

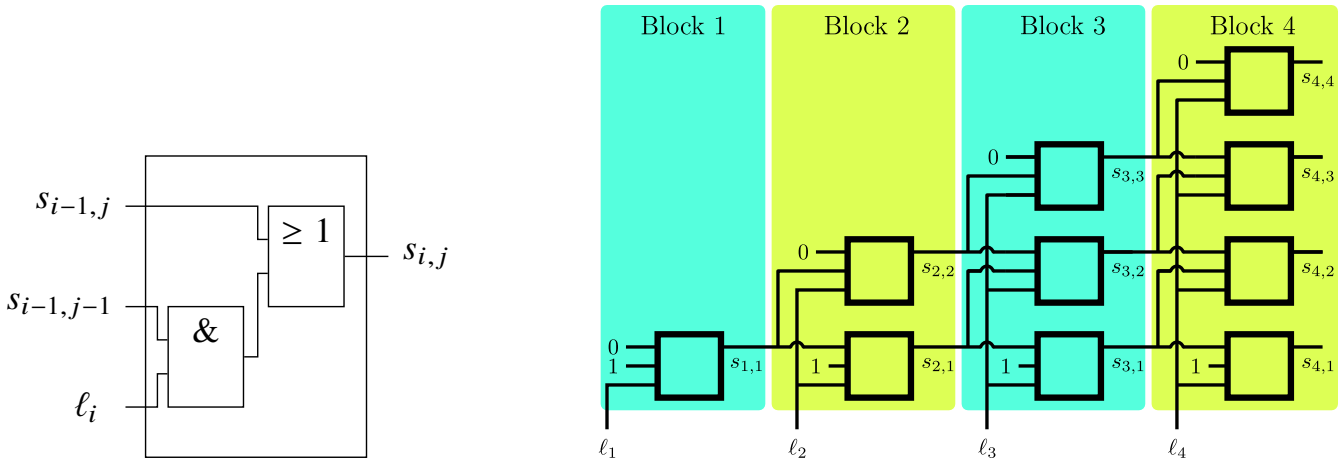
$$\textstyle\sum_{j=1}^{i} \ell_j = \sum_{j=1}^{i} s_{i,j} \,. \tag{3}$$

However, the sequential counter circuit computes the variables $s_{i,j}$ in the ith block using only the variables $s_{i-1,j}$ from the previous block and the literal ℓ_i, and so if we only reason locally about the ith block what we can derive is the equality

$$\ell_i + \textstyle\sum_{j=1}^{i-1} s_{i-1,j} = \sum_{j=1}^{i} s_{i,j} \,. \tag{4}$$

If we look at the variables on wires entering and exiting the ith block of the circuit, we see that Equation (4) specifies that the sum of the inputs is equal to the sum of the outputs. This means that if we traverse the blocks and do a telescoping sum, we can easily derive (3). From this, in turn, it is clear that a constraint on the input variables $\sum_{j=1}^{n} \ell_i \bowtie k$ implies the same constraint on the output variables, and formally this can be obtained by one final telescoping sum step combining $\sum_{j=1}^{n} \ell_i \bowtie k$ and $\sum_{j=1}^{n} \ell_i = \sum_{j=1}^{n} s_{n,j}$ to get

$$\textstyle\sum_{j=1}^{n} s_{n,j} \bowtie k \,. \tag{5}$$



(a) Logic circuit for single component.

(b) Circuit for 4 input literals counting up to 4.

Figure 2: Circuit representation of the sequential counter encoding.

Another important property of the variables $s_{i,j}$ is that they do not just take any values satisfying (4), but are ordered—since $s_{i,j}$ encodes $\sum_{t=1}^{i} \ell_t \geq j$, it follows that $s_{i,j}$ cannot be true unless also $s_{i,j'}$ is true for all $j' < j$. This can be expressed by *ordering constraints*

$$s_{i,j} - s_{i,j+1} \geq 0 \qquad i \in [n]\,, j \in [i-1]\,, \tag{6}$$

which are semantically implied by the circuit encoding.

Taking this view of the circuit encoding, the task of certifying the correctness of the CNF translation becomes surprisingly simple. If we can derive the pseudo-Boolean constraints (4)–(6), then it can be verified that the clauses of the sequential counter encoding (i.e., (2a)–(2d) plus $\overline{s}_{n,k+1} \geq 1$ and/or $s_{n,k} \geq 1$) all follow by reverse unit propagation, meaning that the clauses can just be claimed as true with the proof left to the proof checker. This is so since when asserting the clauses to false, the ordering constraints (6) will propagate enough variables $s_{i,j}$ for (4) to be falsified.

To see how to obtain the constraints (4)–(6), note that we already discussed above how to derive (5) by a telescoping sum over constraints (4), which is straightforward to do with standard cutting planes rules. To get constraints on the form (4), we can first use the *redundance-based strengthening rule* in [Gocht and Nordström, 2021] to derive the *reified constraints*

$$j \cdot \overline{s}_{i,j} + \ell_i + \textstyle\sum_{j=1}^{i-1} s_{i-1,j} \geq j \tag{7a}$$

$$(i-j+1) \cdot s_{i,j} + \overline{\ell}_i + \textstyle\sum_{j=1}^{i-1} \overline{s}_{i-1,j} \geq i-j+1 \tag{7b}$$

(which together can be seen to enforce the equivalence $s_{i,j} \Leftrightarrow \ell_i + \sum_{j=1}^{i-1} s_{i-1,j} \geq j$). If we do this in increasing order for i and j, then $s_{i,j}$ is a fresh variable for each new pair of constraints (7a)–(7b), which can be shown to imply that these are valid derivation steps. From the constraints (7a)–(7b) we can then derive (4) and (6) (though these steps are slightly trickier, and we refer to [Gocht *et al.*, 2022] for the details).

The fact that we can perform all the derivations on 0-1 linear inequalities using cutting planes reasoning make the steps very efficient (and quite elegant). Although the final constraints we need to derive to prove the correctness of the CNF translation are all clausal, it turns out to be very helpful to be able to use 0-1 inequalities in intermediate steps.

3 Experimental Evaluation

To evaluate the proof logging methods developed in this paper, we have implemented certified translations to CNF for the sequential counter, totalizer, generalized totalizer, and binary adder network encodings in the tool VERITASPBLIB. This tool takes a pseudo-Boolean formula in OPB format [Roussel and Manquinho, 2016] and returns a CNF translation together with a proof logging certificate. We have employed the verifier VERIPB [Gocht and Nordström, 2021; Bogaerts *et al.*, 2022] to check the certificate returned by VERITASPBLIB, and have used the SAT solver KISSAT (http://fmv.jku.at/kissat/) in a lightly modified version outputting *DRAT* proofs in pseudo-Boolean format, to solve the CNF formula. Finally, we have conjoined the certificates from the CNF translation and the SAT solving and verified the end-to-end pipeline with VERIPB.

Our evaluation aimed to answer the following questions:

1. Can we use our end-to-end framework to verify the results of CDCL-based pseudo-Boolean solving, and how efficient is the verification?
2. How long does the verification of the proof take when compared to the translation of the PB formula to CNF?

3.1 End-to-End Solving and Verification

We evaluated VERITASPBLIB on 1,534 formulas from the PB Evaluation 2016. Table 1 shows how VERITASPBLIB can be used to generate a CNF formula that can be solved by KISSAT and verified by VERIPB. For instances with cardinality constraints (*Card*), we use the sequential counter and totalizer encodings to translate those constraints to CNF. For instances with general PB constraints (*PB*), our translations use the adder network and generalized totalizer (*GTE*) encodings. Finally, for instances with both cardinality and general

Category	#Inst	Encoding	Translation		Solving			
			#CNF	#Veri	#Solved		#Verified	
					SAT	UNSAT	SAT	UNSAT
Card	772	Sequential	772	772	139	480	133	479
		Totalizer	772	772	139	475	130	474
PB	444	Adder	444	444	179	167	178	165
		GTE	425	414	164	162	150	151
Card+PB	308	Seq+Adder	306	296	134	152	128	151

Table 1: Number of translated, solved and verified instances for each encoding.

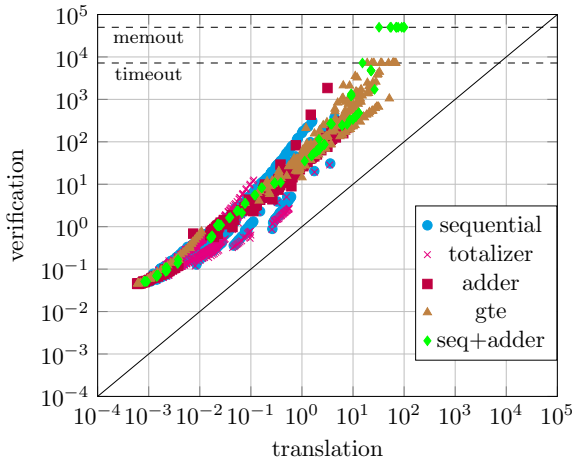


Figure 3: Comparison between translation and verification.

PB constraints (*Card+PB*), we use the sequential counter encoding for cardinality constraints and the adder network encoding for PB constraints, henceforth denoted by *Seq+Adder*.

The column *#CNF* shows for how many instances VERITASPBLIB successfully generated the CNF translation, which is almost all. The exceptions are 19 instances using *GTE* and 2 instances using the *Seq+Adder* encoding. In those cases, the number of clauses generated is too large and exceeds the resource limits used in our evaluation.

The column *#Veri* presents for how many instances VERIPB verified the translation certificate from VERITASPBLIB. Except for a few instances for *GTE* and *Seq+Adder* yielding large proofs, VERIPB is successful. If the translation check passes, then this guarantees that the CNF encoding does not remove any solutions of the pseudo-Boolean formula. The columns *#Solved* and *#Verified* under *Solving* show how many instances can be solved by KISSAT, and from those, how many can be verified by VERIPB. If a satisfiable formula is verified, then all clauses learned by KISSAT are also valid for the original PB formula, as is the satisfying assignment found, and for an unsatisfiable instance we know that the PB input was also unsatisfiable. We can verify 99% of the solved unsatisfiable instances and 95% of satisfiable instances, which shows the feasibility of our approach.

3.2 Translation and Verification

Turning our focus to the certified translation only, our experiments show that the average overhead in running time for proof logging is a factor of 2–3 for all encodings except *GTE*, which incurs around a factor 5 in overhead. However, since the translation to CNF is fast for the majority of instances, the additional overhead of proof logging is not an issue.

Figure 3 compares the time for VERITASPBLIB to generate the CNF translation and VERIPB to verify it. The verification overhead is far from negligible, but is not unreasonable. Over all encodings, for 75% of benchmarks verification takes at most 49 times longer than translation, and for 98% of benchmarks at most 100 times longer. While some overhead is natural, since the translation algorithm can just output a claimed proof while the verifier needs to perform the calculations to actually check it, our experiments do show that there is room for further improvements in efficiency both for the verifier and for the proof logging methods.

4 Concluding Remarks

In this work, we develop a general framework for certified translations of linear pseudo-Boolean constraints into CNF using cutting-planes-based proof logging. Since our method is a strict extension of *DRAT*, the proof for the PB-to-CNF translation can be combined with a SAT solver *DRAT* proof log to provide, for the first time, end-to-end verification for CDCL-based PB solvers. Our use of cutting planes is not only crucial to deal with the PB input format, but the expressivity of the 0-1 linear constraints also allows us to certify the correctness of the translation to CNF in a concise and elegant way. While there is still room for performance improvements in proof logging and verification, our evaluation shows that this approach is feasible in practice. One bottleneck is that the pseudo-Boolean proof checker VERIPB is comparatively slow at verifying *DRAT* proofs—this could be addressed by having the SAT solver output *LRAT* proofs instead.

We wish to stress that we view certified PB-to-CNF translations only as a first step. In the conference version of this paper, we expressed optimism that the techniques we have developed could also be extended to *core-guided MaxSAT solving* [Fu and Malik, 2006], and such results have very recently been announced in [Berg *et al.*, 2023]. While designing efficient proof logging for other MaxSAT approaches such as *implicit hitting sets (IHS)* [Davies and Bacchus, 2011] and *abstract cores* [Berg *et al.*, 2020] seems more challenging, we are hopeful that our work could lead to a unified proof logging method for all of modern MaxSAT solving, and also for pseudo-Boolean optimization using cutting-planes-based reasoning as in [Devriendt *et al.*, 2021a; Devriendt *et al.*, 2021b; Elffers and Nordström, 2018; Le Berre and Parrain, 2010; Smirnov *et al.*, 2021; Smirnov *et al.*, 2022].

Acknowledgements

The authors wish to acknowledge helpful and stimulating discussions with Bart Bogaerts and Ciaran McCreesh. We are particularly grateful to Bart for sharing the manuscript [Vandesande *et al.*, 2022] using a very elegant reification technique that we wish we would have thought of. We believe it would be worth exploring whether similar ideas could be used in our framework to improve the efficiency of verification.

Stephan Gocht and Jakob Nordström were supported by the Swedish Research Council grant 2016-00782, and Jakob Nordström also received funding from the Independent Research Fund Denmark grant 9040-00389B. Ruben Martins was supported by National Science Foundation award CCF-1762363 and an Amazon Research Award, and Andy Oertel was supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation.

References

[Akgün *et al.*, 2018] Özgür Akgün, Ian P. Gent, Christopher Jefferson, Ian Miguel, and Peter Nightingale. Metamorphic testing of constraint solvers. In *Proc. 24th International Conference on Principles and Practice of Constraint Programming (CP '18)*, pages 727–736, 2018.

[Baek *et al.*, 2021] Seulkee Baek, Mario Carneiro, and Marijn J. H. Heule. A flexible proof format for SAT solver-elaborator communication. In *Proc. 27th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS '21)*, pages 59–75, 2021.

[Bailleux and Boufkhad, 2003] Olivier Bailleux and Yacine Boufkhad. Efficient CNF encoding of Boolean cardinality constraints. In *Proc. 9th International Conference on Principles and Practice of Constraint Programming (CP '03)*, pages 108–122, 2003.

[Barnett and Biere, 2021] Lee A. Barnett and Armin Biere. Non-clausal redundancy properties. In *Proc. 28th International Conference on Automated Deduction (CADE-28)*, pages 252–272, 2021.

[Barth, 1995] Peter Barth. A Davis-Putnam based enumeration algorithm for linear pseudo-Boolean optimization. Technical Report MPI-I-95-2-003, Max-Planck-Institut für Informatik, 1995.

[Berg *et al.*, 2020] Jeremias Berg, Fahiem Bacchus, and Alex Poole. Abstract cores in implicit hitting set MaxSat solving. In *Proc. 23rd International Conference on Theory and Applications of Satisfiability Testing (SAT '20)*, pages 277–294, 2020.

[Berg *et al.*, 2023] Jeremias Berg, Bart Bogaerts, Jakob Nordström, Andy Oertel, and Dieter Vandesande. Certified core-guided MaxSAT solving. In *Proc. 29th International Conference on Automated Deduction (CADE-29)*, 2023. To appear.

[Biere *et al.*, 2021] Armin Biere, Marijn J. H. Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability*, volume 336 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2nd edition, 2021.

[Bogaerts *et al.*, 2022] Bart Bogaerts, Stephan Gocht, Ciaran McCreesh, and Jakob Nordström. Certified symmetry and dominance breaking for combinatorial optimisation. In *Proc. 36th AAAI Conference on Artificial Intelligence (AAAI '22)*, pages 3698–3707, 2022.

[Brummayer *et al.*, 2010] Robert Brummayer, Florian Lonsing, and Armin Biere. Automated testing and debugging of SAT and QBF solvers. In *Proc. 13th International Conference on Theory and Applications of Satisfiability Testing (SAT '10)*, pages 44–57, 2010.

[Bryant *et al.*, 2022] Randal E. Bryant, Armin Biere, and Marijn J. H. Heule. Clausal proofs for pseudo-Boolean reasoning. In *Proc. 28th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS '22)*, pages 443–461, 2022.

[Buss and Nordström, 2021] Samuel R. Buss and Jakob Nordström. Proof complexity and SAT solving. In Biere et al. [2021], chapter 7, pages 233–350.

[Cheung *et al.*, 2017] Kevin K. H. Cheung, Ambros M. Gleixner, and Daniel E. Steffy. Verifying integer programming results. In *Proc. 19th International Conference on Integer Programming and Combinatorial Optimization (IPCO '17)*, pages 148–160, 2017.

[Cook *et al.*, 1987] William Cook, Collette Rene Coullard, and György Turán. On the complexity of cutting-plane proofs. *Discrete Applied Mathematics*, 18(1):25–38, 1987.

[Cook *et al.*, 2013] William Cook, Thorsten Koch, Daniel E. Steffy, and Kati Wolter. A hybrid branch-and-bound approach for exact rational mixed-integer programming. *Mathematical Programming Computation*, 5(3):305–344, 2013.

[Davies and Bacchus, 2011] Jessica Davies and Fahiem Bacchus. Solving MAXSAT by solving a sequence of simpler SAT instances. In *Proc. 17th International Conference on Principles and Practice of Constraint Programming (CP '11)*, pages 225–239, 2011.

[Devriendt *et al.*, 2021a] Jo Devriendt, Ambros Gleixner, and Jakob Nordström. Learn to relax: Integrating 0-1 integer linear programming with pseudo-Boolean conflict-driven search. *Constraints*, 26(1–4):26–55, 2021.

[Devriendt *et al.*, 2021b] Jo Devriendt, Stephan Gocht, Emir Demirović, Jakob Nordström, and Peter Stuckey. Cutting to the core of pseudo-Boolean optimization: Combining core-guided search with cutting planes reasoning. In *Proc. 35th AAAI Conference on Artificial Intelligence (AAAI '21)*, pages 3750–3758, 2021.

[Eén and Sörensson, 2006] Niklas Eén and Niklas Sörensson. Translating pseudo-Boolean constraints into SAT. *Journal on Satisfiability, Boolean Modeling and Computation*, 2(1-4):1–26, 2006.

[Eifler and Gleixner, 2021] Leon Eifler and Ambros Gleixner. A computational status update for exact rational mixed integer programming. In *Proc. 22nd International Conference on Integer Programming and Combinatorial Optimization (IPCO '21)*, pages 163–177, 2021.

[Elffers and Nordström, 2018] Jan Elffers and Jakob Nordström. Divide and conquer: Towards faster pseudo-Boolean solving. In *Proc. 27th International Joint Conference on Artificial Intelligence (IJCAI '18)*, pages 1291–1299, 2018.

[Elffers *et al.*, 2020] Jan Elffers, Stephan Gocht, Ciaran McCreesh, and Jakob Nordström. Justifying all differences using pseudo-Boolean reasoning. In *Proc. 34th AAAI Conference on Artificial Intelligence (AAAI '20)*, pages 1486–1494, 2020.

[Fu and Malik, 2006] Zhaohui Fu and Sharad Malik. On solving the partial MAX-SAT problem. In *Proc. 9th International Conference on Theory and Applications of Satisfiability Testing (SAT '06)*, pages 252–265, 2006.

[Gillard *et al.*, 2019] Xavier Gillard, Pierre Schaus, and Yves Deville. SolverCheck: Declarative testing of constraints. In *Proc. 25th International Conference on Principles and Practice of Constraint Programming (CP '19)*, pages 565–582, 2019.

[Gocht and Nordström, 2021] Stephan Gocht and Jakob Nordström. Certifying parity reasoning efficiently using pseudo-Boolean proofs. In *Proc. 35th AAAI Conference on Artificial Intelligence (AAAI '21)*, pages 3768–3777, 2021.

[Gocht *et al.*, 2020] Stephan Gocht, Ciaran McCreesh, and Jakob Nordström. Subgraph isomorphism meets cutting planes: Solving with certified solutions. In *Proc. 29th International Joint Conference on Artificial Intelligence (IJCAI '20)*, pages 1134–1140, 2020.

[Gocht *et al.*, 2022] Stephan Gocht, Ruben Martins, Jakob Nordström, and Andy Oertel. Certified CNF translations for pseudo-Boolean solving. In *Proc. 25th International Conference on Theory and Applications of Satisfiability Testing (SAT '22)*, pages 16:1–16:25, 2022.

[Goldberg and Novikov, 2003] Evgueni Goldberg and Yakov Novikov. Verification of proofs of unsatisfiability for CNF formulas. In *Proc. Conference on Design, Automation and Test in Europe (DATE '03)*, pages 886–891, 2003.

[Joshi *et al.*, 2015] Saurabh Joshi, Ruben Martins, and Vasco M. Manquinho. Generalized totalizer encoding for pseudo-Boolean constraints. In *Proc. 21st International Conference on Principles and Practice of Constraint Programming (CP '15)*, pages 200–209, 2015.

[Kaufmann and Biere, 2021] Daniela Kaufmann and Armin Biere. AMulet 2.0 for verifying multiplier circuits. In *Proc. 27th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS '21)*, pages 357–364, 2021.

[Kaufmann *et al.*, 2022] Daniela Kaufmann, Paul Beame, Armin Biere, and Jakob Nordström. Adding dual variables to algebraic reasoning for circuit verification. In *Proc. 25th Design, Automation and Test in Europe Conference (DATE '22)*, pages 1435–1440, 2022.

[Le Berre and Parrain, 2010] Daniel Le Berre and Anne Parrain. The Sat4j library, release 2.2. *Journal on Satisfiability, Boolean Modeling and Computation*, 7:59–64, 2010.

[Marques-Silva and Sakallah, 1999] João P. Marques-Silva and Karem A. Sakallah. GRASP: A search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, 48(5):506–521, 1999.

[McConnell *et al.*, 2011] Ross M. McConnell, Kurt Mehlhorn, Stefan Näher, and Pascal Schweitzer. Certifying algorithms. *Computer Science Review*, 5(2):119–161, 2011.

[Moskewicz *et al.*, 2001] Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an efficient SAT solver. In *Proc. 38th Design Automation Conference (DAC '01)*, pages 530–535, 2001.

[Philipp and Steinke, 2015] Tobias Philipp and Peter Steinke. PBLib – a library for encoding pseudo-Boolean constraints into CNF. In *Proc. 18th International Conference on Theory and Applications of Satisfiability Testing (SAT '15)*, pages 9–16, 2015.

[Roussel and Manquinho, 2016] Olivier Roussel and Vasco M. Manquinho. Input/output format and solver requirements for the competitions of pseudo-Boolean solvers. Revision 2324. Available at http://www.cril.univ-artois.fr/PB16/format.pdf, 2016.

[Sinz, 2005] Carsten Sinz. Towards an optimal CNF encoding of Boolean cardinality constraints. In *Proc. 11th International Conference on Principles and Practice of Constraint Programming (CP '05)*, pages 827–831, 2005.

[Smirnov *et al.*, 2021] Pavel Smirnov, Jeremias Berg, and Matti Järvisalo. Pseudo-Boolean optimization by implicit hitting sets. In *Proc. 27th International Conference on Principles and Practice of Constraint Programming (CP '21)*, pages 51:1–51:20, 2021.

[Smirnov *et al.*, 2022] Pavel Smirnov, Jeremias Berg, and Matti Järvisalo. Improvements to the implicit hitting set approach to pseudo-Boolean optimization. In *Proc. 25th International Conference on Theory and Applications of Satisfiability Testing (SAT '22)*, pages 13:1–13:18, 2022.

[Van Gelder, 2008] Allen Van Gelder. Verifying RUP proofs of propositional unsatisfiability. In *10th International Symposium on Artificial Intelligence and Mathematics (ISAIM '08)*, 2008.

[Vandesande *et al.*, 2022] Dieter Vandesande, Wolf De Wulf, and Bart Bogaerts. QMaxSATpb: A certified MaxSAT solver. In *Proc. 16th International Conference on Logic Programming and Non-monotonic Reasoning (LPNMR '22)*, pages 429–442, 2022.

[Warners, 1998] Joost P. Warners. A linear-time transformation of linear inequalities into conjunctive normal form. *Information Processing Letters*, 68(2):63–69, 1998.

[Wetzler *et al.*, 2014] Nathan Wetzler, Marijn J. H. Heule, and Warren A. Hunt Jr. DRAT-trim: Efficient checking and trimming using expressive clausal proofs. In *Proc. 17th International Conference on Theory and Applications of Satisfiability Testing (SAT '14)*, pages 422–429, 2014.