Certifying Without Loss of Generality Reasoning in Solution-Improving Maximum Satisfiability

Jeremias Berg
Department of Computer Science, HIIT, Helsinki, Finland
University of Helsinki, Finland

Bart Bogaerts
Vrije Universiteit Brussel, Belgium

Jakob Nordström
University of Copenhagen, Denmark
Lund University, Sweden

Andy Oertel
Lund University, Sweden
University of Copenhagen, Denmark

Tobias Paxian
University of Freiburg, Germany

Dieter Vandesande
Vrije Universiteit Brussel, Belgium

Abstract

Proof logging has long been the established method to certify correctness of Boolean satisfiability (SAT) solvers, but has only recently been introduced for SAT-based optimization (MaxSAT). The focus of this paper is solution-improving search (SIS), in which a SAT solver is iteratively queried for increasingly better solutions until an optimal one is found. A challenging aspect of modern SIS solvers is that they make use of complex "without loss of generality" arguments that are quite involved to understand even at a human meta-level, let alone to express in a simple, machine-verifiable proof.

In this work, we develop pseudo-Boolean proof logging methods for solution-improving MaxSAT solving, and use them to produce a certifying version of the state-of-the-art solver Pacose with VeriPB proofs. Our experimental evaluation demonstrates that this approach works in practice. We hope that this is yet another step towards general adoption of proof logging in MaxSAT solving.

2012 ACM Subject Classification Theory of computation → Automated reasoning; Theory of computation → Constraint and logic programming; Mathematics of computing → Combinatorial optimization

Keywords and phrases proof logging, certifying algorithms, MaxSAT, solution-improving search, SAT-UNSAT, maximum satisfiability, combinatorial optimization, certification, pseudo-Boolean

Digital Object Identifier 10.4230/LIPIcs.CP.2024.4

Supplementary Material *Dataset (experimental data and source code)*: `https://zenodo.org/records/12591387` [8]

Funding *Jeremias Berg*: Research Council of Finland under grants 342145.
Jakob Nordström: Swedish Research Council grant 2016-00782 and Independent Research Fund Denmark grant 9040-00389B.
Andy Oertel: Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation.
Dieter Vandesande: Fonds Wetenschappelijk Onderzoek – Vlaanderen (project G070521N).

30th International Conference on Principles and Practice of Constraint Programming (CP 2024).
Editor: Paul Shaw; Article No. 4; pp. 4:1–4:28

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Acknowledgements We want to thank Florian Pollitt and Mathias Fleury for their assistance with the CaDiCaL proof tracer and for fuzzing VeriPB within CaDiCaL. Their contributions were very helpful to further improve the robustness of the VeriPB toolchain. We also wish to acknowledge useful discussions with participants of the Dagstuhl workshop 23261 *SAT Encodings and Beyond*. The computational experiments were enabled by resources provided by LUNARC at Lund University.

1 Introduction

Thanks to tremendous progress over the last decades on algorithms for combinatorial search and optimization, today *NP*-hard problems are routinely solved in many practical applications. Unfortunately, as these algorithms get more and more sophisticated, it also gets more and more challenging to avoid errors sneaking in during algorithm design and implementation. It is well-known that modern combinatorial solving algorithms in different paradigms can sometimes produce "solutions" that violate hard constraints, claim that suboptimal solutions are optimal, or declare that feasible problems lack solutions [9, 15, 16, 19, 30, 39].

Although there are many ways to address this problem, including software testing techniques such as fuzzing [15, 50], and design of formally verified software [28], the most promising approach appears to be the use of *certifying algorithms* [1, 48] with so-called *proof logging*. What this means is the algorithm should not only produce an answer, but also a *proof* that this answer is correct. Such proofs should follow simple rules, as specified by a formal *proof system*, so that they can easily be verified by an independent *proof checker*. In addition to guaranteeing correctness, proof logging brings many other advantages: it enables advanced *testing* (since one can detect correct answers found for invalid reasons, and also test instances for which the answer is not known), detailed *debugging* (since invalid proof steps pinpoint where errors happened), *auditability* (since proofs can be stored and verified independently of which algorithm was used), and *performance analysis* (since proofs can be mined for insights on which reasoning steps were crucial for reaching the final conclusion).

Proof logging has been particularly successful in the domain of Boolean satisfiability (SAT) solving [11], where a large variety of proof systems has seen the light of day [4, 10, 35, 63]. Using proof logging has long been mandatory in the main track of the SAT competitions, and it is hard to overestimate the impact this has had on improving overall correctness and reliability of SAT solvers. This has stimulated the spread of proof logging into other combinatorial solving paradigms, including SAT modulo theories (SMT) [7, 57], automated planning [25–27, 56], and mixed integer linear programming [21, 24].

1.1 Proof Logging for MaxSAT Solving

In view of the above discussion, it is interesting to compare the developments in other combinatorial optimization paradigms to the state of affairs in maximum satisfiability (MaxSAT), the optimization version of the SAT problem. Without loss of generality, MaxSAT can be described as the problem of maximizing a linear objective O subject to satisfying a Boolean formula F in conjunctive normal form (CNF). Although MaxSAT is arguably the one optimization paradigm closest to SAT, and although several proof systems for formalizing MaxSAT reasoning have been proposed [14, 42, 49, 53–55], for a long time there has been no practically feasible proof logging method for state-of-the-art MaxSAT solvers. This changed only recently when pseudo-Boolean proof logging using VeriPB [12, 34] was proposed for MaxSAT [59, 60], a proposal that was followed by the successful design and implementation of VeriPB proof logging for modern core-guided MaxSAT solvers [9].

In this paper, we revisit proof logging work for *solution-improving search (SIS)* [59,60], also referred to as *model-improving search* or *linear SAT-UNSAT (LSU) search*, and consider state-of-the-art solving techniques. In the SIS approach – which is much simpler to explain than, e.g., core-guided [29] or implicit hitting set [20] search – a SAT solver is repeatedly called on the formula F, each time with an added *solution-improving constraint* asking for increasingly better solutions with respect to the objective O, and the problem turns infeasible when the last solution found was optimal. In the work by Vandesande et al. [59, 60], the main technical challenge was to certify correctness of the CNF encodings of these solution-improving constraints, which could then essentially be concatenated with the proof logging generated by the SAT solver (modulo some non-trivial engineering).

At first sight, it seems that implementing pseudo-Boolean proof logging in a state-of-the-art MaxSAT solver using solution-improving search would mostly be a matter of carefully transferring already developed techniques [59,60], perhaps combining them with proof logging ideas developed for other CNF encodings [31]. After all, the distinguishing feature of a top-of-the-line SIS solver is the choice of CNF translation for reasoning about the objective function, such as, in the case of Pacose, the *polynomial watchdog (DPW)* encoding [6]. Once proof logging for such a CNF encoding is in place, it seems reasonable to expect that the rest should be plain sailing.

It is all the more surprising, then, that it turns out nothing could be further from the truth. To minimize the time the MaxSAT solver spends on generating PW encodings, an essential step is to introduce completely unconstrained variables that can be used to perform different comparisons with a single CNF encoding; this is referred to as the *dynamic polynomial watchdog encoding (DPW)* [52]. Loosely speaking, if we know that the best possible objective value lies in the range $[lo, hi]$, then instead of generating repeated encodings $O \geq V$ to probe different possible objective values V in this range, one can introduce free variables t_i encoding a tare sum T taking values between 0 and $hi - lo$ and try to maximize the value $T = T^*$ for which one single DPW-encoded constraint $O - T \geq lo$ holds. Once the maximum T^* has been found, it is clear that $O = lo + T^*$ is the best possible objective value, since without loss of generality T could be set to any value. But how can such a meta-argument be expressed in simple propositional logic reasoning?

In what follows, we provide a brief, if still high-level, discussion of some of the challenges that arise when trying to design simple proofs to certify such fairly complex "without loss of generality" arguments, and then outline how such challenges can be overcome.

1.2 Solution-Improving "Without Loss of Generality" Reasoning

As already discussed above, the key aspect in which different solution-improving MaxSAT solvers differ is how they encode the solution-improving constraints. In order to compute the value of a linear expression L over 0–1 variables of interest, Pacose uses the polynomial watchdog encoding to describe a Boolean circuit BC with output variables z_k such that $z_k = 0$ implies $L \geq 1 + k \cdot 2^P$ (for some fixed integer P). If we chose L to be the objective function O that we are maximizing, this would allow to find the interval $\left[1 + k^* \cdot 2^P, (k^* + 1) \cdot 2^P\right]$ in which the optimal value lies by calling the SAT solver with the prechosen partial assignment $z_k = 0$ (referred to as an *assumption*) for increasing values of k until the solver returns that there is no satisfying assignment. To determine the exact location of the optimum in this interval, additional, completely unconstrained, variables t_i, called *tare variables*, are used to encode an integer $T = \sum_{i=0}^{P-1} 2^i t_i$ in the range $\left[0, 2^P - 1\right]$. The actual circuit in the encoding uses the linear form $L = O - T$, so that $z_k = 0$ means $O - T \geq 1 + k \cdot 2^P$. By making SAT solver calls with suitable assumptions on the unconstrained t_i-variables, the optimal value of the objective function can be computed.

Given the CNF encoding of a circuit $\mathsf{BC}\big(O - T \geq 1 + k \cdot 2^P\big)$ evaluating the inequality $O - T \geq 1 + k \cdot 2^P$ as outlined above, the solution-improving search proceeds in two phases:

(i) The *coarse convergence phase* identifies the largest k for which $z_k = 0$ is possible.

(ii) The *fine convergence phase* then maximizes the tare variable sum T.

Let us discuss this process in slightly more detail, and explain why it presents challenges from a proof logging point of view.

If during the coarse convergence phase a SAT solver call with assumption $z_k = 0$ returns a satisfying assignment α achieving objective value at least $1 + k \cdot 2^P$, the solver stores the information $z_k = 0$ (in the form of a unit clause $\overline{z}_k$), which enforces that any future solutions found have to be at least this good. The SAT solver is then called again with $z_{k'} = 0$ for some $k' > k$ to probe whether a solution exists with value at least $1 + k' \cdot 2^P$. Here it is relevant to note that fixing $z_k = 0$ could remove assignments corresponding to optimal solutions. For instance, if the optimal value is $V = V^* + 1 + k \cdot 2^P$, this value could be achieved by an assignment α' setting $T = T^* > V^* + 1$. For such an α' we would have $O - T = -T^* + V^* + 1 + k \cdot 2^P \leq k \cdot 2^P$, which would violate $z_k = 0$. However, since the tare variables are unconstrained, in this case there would also exist another assignment α'' achieving objective value $V^* + k \cdot 2^P$ for which $T = 0$, and so it is safe to require that solutions improving on α should satisfy $z_k = 0$.

In the fine convergence phase the z_k-variables are all fixed, and assumptions on the tare variables are made in the SAT solver calls to determine the exact value of the optimal solution. This again relies on reasoning without loss of generality, claiming that one can always choose $T \geq s$ for any value $0 \leq s < 2^P$. But now we are treading on dangerous ground: clearly, we cannot assume both $T = 0$ and $T \geq s > 0$ simultaneously! How can we convince ourselves, and more importantly, how can we convince a proof checker, that our derivations are consistent? At a meta-level, we can argue that since the tare variables are completely unconstrained in the original encoding, we should be able to fix them to any value we like at any given point in time. But how do we produce a simple, machine-verifiable proof that this is sound? And are we even sure this is sound?

1.3 Discussion of Our Contribution

In this work, we show how pseudo-Boolean proof logging with VeriPB [12, 34] can certify correctness of the complex CNF encodings used in state-of-the-art solution-improving MaxSAT solvers, as well as of the subtle without loss of generality reasoning applied on these encodings. To give a sense of how this can be done, we need to give a high-level description how VeriPB proofs work (referring the reader to later sections for the missing technical details).

A VeriPB proof maintains a set of *core constraints* $\mathcal{C}$, initialized to the formula F, together with a set of *derived constraints* $\mathcal{D}$ inferred by the solver. The proof semantics ensures that $\mathcal{C}$ and F have the same optimal value for O and that any solution to $\mathcal{C}$ can be extended to $\mathcal{D}$. A new constraint C can be derived "without loss of generality" by the *redundance-based strengthening rule*, which requires the explicit specification of a substitution ω (mapping variables to truth values or literals) together with explicit proofs

$$\mathcal{C} \cup \mathcal{D} \cup \{\neg C\} \vdash (\mathcal{C} \cup \mathcal{D} \cup \{C\})\restriction_\omega \cup \{O\restriction_\omega \geq O\} \tag{1}$$

that all consequences on the right (with the substitution ω applied to the constraints) follow from previously derived constraints $\mathcal{C} \cup \mathcal{D}$ together with the negation $\neg C$ of the constraint to be inferred. This guarantees that if some assignment α satisfies everything so far but violates C, the "patched" assignment $\alpha \circ \omega$ satisfies also C and does not worsen the objective.

To make our informal discussion simple and concrete, suppose that we have a CNF encoding of a circuit $\mathsf{BC}(O-T \geq lo)$ evaluating $O-T \geq lo$, and that the solver has derived no constraints but only has the input formula F. If we want to fix $T = T^*$ using the redundance rule (1), we would have to find a substitution ω such that $F \cup \{\mathsf{BC}(O-T \geq lo)\} \cup \{T \neq T^*\}$ implies $\big(F \cup \{\mathsf{BC}(O-T \geq lo)\} \cup \{T = T^*\}\big)\restriction_\omega$. But it seems like this would force us to prove that if we take any assignment satisfying the Boolean circuit and modify the value of some of its inputs (the tares), the circuit would remain satisfied, and this is just not true. So although the redundance-based strengthening rule is very strong, it is not clear how it can be used to argue that the tare variables are unconstrained.

We get around this problem by first deriving a copy *shadow circuit* BC' of the original circuit, but substituting fixed values t_i^* for the tare variables, so that $\mathsf{BC}'(O - T^* \geq lo)$ evaluates $O - T^* \geq lo$. We then let ω be the substitution setting $t_i = t_i^*$ for all i and mapping all other variables x in BC to the corresponding shadow variables x' in BC', so that, effectively, the shadow circuit computes the substitution needed. This turns our application of the redundance rule (1) into

$$F \cup \{\mathsf{BC}(O-T \geq lo)\} \cup \{\mathsf{BC}'(O-T^* \geq lo)\} \cup \{T \neq T^*\} \tag{2a}$$
$$\vdash \big(F \cup \{\mathsf{BC}(O-T \geq lo)\} \cup \{\mathsf{BC}'(O-T^* \geq lo)\} \cup \{T = T^*\}\big)\restriction_\omega \cup \{O\restriction_\omega \geq O\} \tag{2b}$$
$$= F \cup \{\mathsf{BC}'(O-T^* \geq lo)\} \cup \{\mathsf{BC}'(O-T^* \geq lo)\} \cup \{T^* = T^*\} \cup \{O \geq O\} \tag{2c}$$

(where the final line (2c) is simply the result of applying the substitution ω to (2b)). If we study (2c) carefully, we see that all we need to prove about the circuit now is that the two copies of the shadow circuit in the consequences are implied by the same shadow circuit in the premises, and so (2c) follows trivially from the premises (2a).

This idea of using shadow circuits is crucial for certifying the correctness of assigning tare variables without loss of generality. However, we need to get rid of the completely unrealistic assumption that the solver would not have learned any constraints in $\mathcal{D}$. This is a problem in that the above argument fails when such learned constraints $D \in \mathcal{D}$ contain variables in the BC-circuit, since then there is no way to prove $D\restriction_\omega$ as required in (1).

Here a second idea discovered in recent VeriPB development turns out to be very helpful. Very briefly, it can be shown that if in the proof we enforce the requirement that all new constraints D derived by strengthening are immediately moved to the core set $\mathcal{C}$, referred to as *strengthening-to-core*, then the redundance rule (1) can be simplified to

$$\mathcal{C} \cup \mathcal{D} \cup \{\neg C\} \vdash (\mathcal{C} \cup \{C\})\restriction_\omega \cup \{O\restriction_\omega \geq O\}, \tag{3}$$

omitting the proof obligations for the derived set $\mathcal{D}$. This means that we can ignore the problems arising from derived constraints when using shadow circuit reasoning.

We stress that this is only a brief and informal discussion that sweeps many technical challenges under the rug. Perhaps one of the most annoying such challenges is that the tare variables are sometimes fixed one at a time, and then a new shadow circuit is required for every new fixing. It would be desirable to find better ways of dealing with this problem.

We have implemented our methods in the state-of-the-art solution-improving MaxSAT solver Pacose [52] to make it output VeriPB proofs, and have performed an extensive evaluation of how such proof logging works in practice. While there is certainly room for performance improvements in both proof generation and proof checking, the significance of our contribution is that we present practical methods to certify correctness for a solving paradigm that has previously been beyond the reach of proof logging. We hope that our work can serve as an impetus towards general adoption of proof logging for MaxSAT, and can stimulate further research on how to make these proof logging techniques more efficient.

As a final remark, we note that an interesting aspect of recent progress in proof logging is that it brings together all three software quality assurance methods discussed in the opening paragraphs above. While proof logging does seem like the most viable approach to certify correctness in combinatorial solving, extensive use of fuzzing techniques has been instrumental in our work to debug both proof logging routines and the VeriPB proof checker. This fuzzing, in turn, relies on the use of proof logging and on feedback from the proof checker. Finally, although we do not address this aspect in the current paper, formally verified proof checking backends as in [33, 37] are crucially needed to ensure that the verdict of proof checkers for increasingly powerful proof logging systems can be trusted.

1.4 Outline of This Paper

After reviewing some preliminaries in Section 2, we discuss the dynamic polynomial watchdog (DPW) encoding in Section 3. In Section 4 we describe how to design proof logging for solution-improving solvers using the DPW encoding, including a discussion of possible variations of our method (and of why simply using SAT proof logging for the final unsatisfiability call does not work). We report results from an empirical evaluation in Section 5 and end with some conclusions and a discussion of future research directions in Section 6.

2 Preliminaries

In this section, we review some pseudo-Boolean basics and then discuss MaxSAT in general and solution-improving search in particular, referring the reader to [3, 17, 44] for more details.

2.1 Pseudo-Boolean Constraints and Proofs

We write x to denote a $\{0,1\}$-valued Boolean variable, and write $\overline{x}$ as a shorthand for $1-x$, using ℓ to denote such *positive* and *negative literals*, respectively. A (linear) *pseudo-Boolean (PB) constraint* C is a 0–1 integer linear inequality $\sum_i w_i\ell_i \geq A$. Without loss of generality, we will often assume our constraints to be *normalized*, meaning that all literal are over distinct variables and the coefficients w_i and the *degree* A are non-negative. A *PB formula* is a conjunction of PB constraints.

A (disjunctive) *clause* is a PB constraint $\sum_i \ell_i \geq 1$ with all coefficients and degree equal to 1. We sometimes refer to constraints $\ell \geq 1$ with a single literal as *unit clauses* ℓ. We say that a formula is in *conjunctive normal form (CNF)* if it is a conjunction of clauses. A *(linear) pseudo-Boolean term* is a weighted sum $\sum_i w_i\ell_i$ of literals with integer coefficients. A *(partial) assignment* α is a (partial) function from variables to $\{0,1\}$; it is extended to literals by respecting the meaning of negation. We write $C{\restriction_\alpha}$ for the constraint obtained from C by substituting all assigned variables x by $\alpha(x)$ (and simplifying). A constraint C is *satisfied* under α if $\sum_{\alpha(\ell_i)=1} w_i \geq A$, and a formula F is satisfied if all its constraints are. We say that F *implies* C, denoted $F \models C$, if all assignments that satisfy F also satisfy C.

A *pseudo-Boolean optimization* (PBO) instance consists of a formula F and a linear term $O = \sum_i w_i\ell_i$ (called the *objective*). An assignment α to the variables in F and O that satisfies F is a *solution* to the instance, which is optimal if it *maximizes* the value $O{\restriction_\alpha} = \sum_i w_i\alpha(\ell_i)$.[1] For a PBO instance (F, O) the VeriPB proof system maintains a

[1] Note that most of the PBO literature is formulated in terms of *minimization*, and this is also the perspective of VeriPB, but reasoning in terms of maximization is in line with the papers on solution-improving MaxSAT relevant for this work. We therefore adopt this perspective here, although the actual VeriPB proofs will argue in terms of minimizing the negation of the objective as described here.

proof configuration of *core* and *derived constraints* $(\mathcal{C}, \mathcal{D})$, initialized to F and $\emptyset$, respectively. The VeriPB proofs we consider are in the so-called *strengthening-to-core* mode, which maintains the invariant that all constraints in the derived set $\mathcal{D}$ are implied by the core set $\mathcal{C}$. Constraints can be moved from $\mathcal{D}$ to $\mathcal{C}$ but not vice versa. New constraints can be derived from $\mathcal{C} \cup \mathcal{D}$ and added to $\mathcal{D}$ using the *cutting planes* proof system [18] as follows:

Literal Axioms. For any literal ℓ_i, $\ell_i \geq 0$ is an axiom.

Linear Combination. Given two previously derived PB constraints C_1 and C_2, any positive integer linear combination of these constraints can be inferred.

Division. Given the normalized PB constraint $\sum_i w_i \ell_i \geq A$ and a positive integer c, the constraint $\sum_i \lceil w_i/c \rceil \ell_i \geq \lceil A/c \rceil$ can be inferred.

Some additional VeriPB proof rules extending cutting planes are as listed below – we refer to [12, 34, 36] for more details. For optimization problems we have rules for improvements of or rewriting of the objective function:

Objective Improvement. Given a total assignment α that satisfies $\mathcal{C} \cup \mathcal{D}$, one can add the constraint $O \geq 1 + O\restriction_\alpha$ to $\mathcal{C}$, which forces the search for strictly better solutions.

Objective Reformulation. The current objective O can be replaced by a new objective O_{new} given explicit proofs from the core set $\mathcal{C}$ (using the VeriPB proof rules above) of the constraints $O - O_{\text{new}} \geq 0$ and $O_{\text{new}} - O \geq 0$ (i.e., a proof that $O = O_{\text{new}}$ holds).

Importantly, there are also rules for deriving non-implied constraints as long as the optimal value of the objective is preserved. VeriPB has a generalization of the RAT rule [39] that makes use of *substitutions* ω, mapping variables to truth values or literals (where we extend the meaning of $C\restriction_\omega$ to denote C with each x replaced by $\omega(x)$):

Redundance-Based Strengthening. The constraint C can be inferred and added to $\mathcal{C}$ by explicitly specifying a substitution ω and proofs $\mathcal{C} \cup \mathcal{D} \cup \{\neg C\} \vdash (\mathcal{C} \cup \{C\})\restriction_\omega \cup \{O\restriction_\omega \geq O\}$. This assumes strengthening-to-core mode – otherwise derivations for all constraints in $\mathcal{D}\restriction_\omega$ are also needed (but then C can be placed in $\mathcal{D}$ instead of $\mathcal{C}$).

Intuitively, this rule shows that ω remaps any solution of $\mathcal{C}$ that does not satisfy C to a solution of $\mathcal{C}$ that satisfies also C without worsening the objective value. A typical use case of redundance-based strengthening is *reification*, which is the derivation of two pseudo-Boolean constraints that encode $\ell \Leftrightarrow D$ for some PB constraint D and for some fresh literal ℓ.

Finally, VeriPB has rules for deleting constraint in a way that guarantees that no spurious better-than-optimal solutions are introduced:

Deletion. A constraint $D \in \mathcal{D}$ in the derived set can be deleted at any time. If strengthening-to-core mode is used, then deleting a constraint $C \in \mathcal{C}$ in the core set requires an explicit proof that C is implied by $\mathcal{C} \setminus \{C\}$. Otherwise, it is sufficient to show the weaker property that C can be derived from $\mathcal{C} \setminus \{C\}$ by redundance-based strengthening.

2.2 MaxSAT, Incremental SAT Solving, and Solution-Improving Search

An instance of (weighted partial) Maximum Satisfiability (MaxSAT) consists of a CNF formula F and a pseudo-Boolean objective $O = \sum_i w_i \ell_i$ to be maximized under satisfying assignments to F, where we can assume without loss of generality that all literals in O are over distinct variables and that the constants are positive. Viewing MaxSAT in terms of an objective function and a CNF formula is equivalent to the more classical definition in terms of hard and soft clauses, in the sense that maximizing the objective corresponds to maximizing the total weight of satisfied soft clauses (see, e.g., [43] for more details).

The solution-improving search (SIS) algorithm we focus on in this work makes extensive use of incremental SAT solving with assumptions [22]. Invoking a SAT solver on a CNF formula F with a set of *assumptions* $\mathcal{A}$, i.e., a partial assignment, returns either (i) SAT and an extension of $\mathcal{A}$ that satisfies F or (ii) UNSAT if no such assignment exists.

Given a MaxSAT instance (F, O), solution-improving search (SIS) computes an optimal solution by issuing a sequence of queries to a SAT solver asking for solutions of improving quality until an optimal one is found. More precisely, during search SIS maintains the best known solution α^*. In each iteration, the algorithm queries a SAT solver on the working formula $F \wedge \mathsf{AsCNF}(O > O|_{\alpha^*})$, where $\mathsf{AsCNF}(O > O|_{\alpha^*})$ is a CNF formula that is satisfied by an assignment α if and only if it is a better solution than α^*, i.e., if $O|_\alpha > O|_{\alpha^*}$. If the SAT solver returns SAT, a better solution has been obtained and the working formula updated accordingly. Otherwise, if the SAT solver reports UNSAT, the best known solution α^* is determined to be optimal and the search is terminated.

The existing practical instantiations of SIS differ mainly in how the encoding of the formula $\mathsf{AsCNF}(O > O|_{\alpha^*})$ is realized. Numerous CNF encodings of pseudo-Boolean constraints have been proposed for this task [23, 38, 40, 45, 58]. For many instantiations of SIS the main challenge for proof logging is to certify the clauses added when encoding the objective constraint [59, 60], but as we will explain in the rest of this paper the so-called Dynamic Polynomial Watchdog encoding requires much more subtle arguments.

3 The Dynamic Polynomial Watchdog Encoding for SIS

The polynomial watchdog (PW) encoding [6] is currently one of the best approaches for encoding pseudo-Boolean constraints in CNF, in terms of being compact while still propagating well. Using it for solution-improving search requires some non-trivial alternations, however, such as the addition of a dynamic constant. In this section we review this dynamic polynomial watchdog (DPW) encoding to the extent required for MaxSAT solution-improving search (SIS), referring the reader to [52] for more details.

3.1 Initialization

Given a linear pseudo-Boolean term $L = \sum_i w_i \ell_i$, we define $w_{\max}$ to be the largest constant appearing in L. Additionally, we let $P := \lfloor \log_2(w_{\max}) \rfloor$ be one smaller than the number of bits in the binary representation of $w_{\max}$ and $W := \sum_i w_i$ be the maximum value for L. The polynomial watchdog encoding for L is a CNF formula $\mathsf{PW}(L)$ with $c := \lceil \frac{W}{2^P} \rceil$ output variables z_k for $k \in [0, c-1]$ enforcing the implications $\overline{z}_k \Rightarrow L \geq 1 + k \cdot 2^P$. In words, a satisfying assignment α of $\mathsf{PW}(L)$ that sets $\alpha(z_k) = 0$ will also satisfy $\sum_i w_i \alpha(\ell_i) \geq 1 + k \cdot 2^P$. We describe the formula $\mathsf{PW}(L)$ in more detail in Section 4.1.

▶ **Example 1.** Consider a MaxSAT instance (F, O) and a working formula $F^w = F \wedge \mathsf{PW}(O)$. Assume we first invoke a SAT solver on F^w under the assumption $z_{k-1} = 0$ and then a second time under the assumption $z_k = 0$, and that the solver reports SAT for the first call and UNSAT for the second. At this point, we know that an optimal solution α^{opt} has value $O|_{\alpha^{\text{opt}}}$ in the range $\left[1 + (k-1) \cdot 2^P, k \cdot 2^P\right]$.

The PW encoding was proposed as a way of enforcing a fixed bound B on the term L by considering a (static) constant $T = B - (1 + k \cdot 2^P)$, where k is the largest integer for which $B \geq 1 + k \cdot 2^P$, and encoding $\mathsf{PW}(L - T)$ [6]. Then a solution that sets the k^{th} output z_k of $\mathsf{PW}(L - T)$ to 0 will also satisfy $\sum_i w_i \alpha(\ell_i) - T \geq 1 + k \cdot 2^P$, which is equivalent to

$\sum_i w_i\alpha(\ell_i) \geq B$. The dynamic polynomial watchdog (DPW) encoding [52] is an extension of the PW encoding that allows dynamically changing the value of T, and therefore also of B, so that the optimal value can be determined precisely with a single CNF encoding.

Consider a MaxSAT instance (F, O) and let $P = \lfloor \log_2(w_{\max}) \rfloor$ as described above. Instantiations of SIS with DPW introduce a "dynamic constant" in the form of a *tare* term $T := \sum_{i=0}^{P-1} 2^i \cdot t_i$, for fresh variables t_i not appearing anywhere else in the instance. The SAT solver is instantiated with the working formula $F \wedge \mathsf{PW}(O - T)$. Now we can use the output variables z_k to determine the optimal value within an additive constant 2^P, and then assign the tare T to values in $[0, 2^P - 1]$ to determine the precise value in that range. These are the *coarse convergence* and *fine convergence* phases mentioned in Section 1.2, which we describe in more detail next.

3.2 Coarse Convergence Phase

During the initial coarse convergence phase, only assumptions over the output variables z_k are made. Whenever a solution α is found, a call to the SAT solver is made with the assumption $z_k = 0$ where k is the largest natural number such that $O\restriction_\alpha \geq 1 + (k-1) \cdot 2^P$. The coarse convergence phase ends when the solver reports UNSAT. The following observation summarizes the relevant conclusions of coarse convergence.

▶ **Observation 2.** *Assume F is satisfiable and the SAT solver returns UNSAT under an assumption $z_{k^*} = 0$ in the coarse convergence phase. Then (i) there is a solution α^* to $F \wedge \mathsf{PW}(O-T)$ that assigns the tare variables so that $(O-T)\restriction_{\alpha^*} \geq 1+(k^*-1)\cdot 2^P$ holds, and (ii) no solution β to F assigning also the tare variables can satisfy $(O-T)\restriction_\beta \geq 1 + k^* \cdot 2^P$.*

In words, coarse convergence provides bounds on the maximum value of $O - T$ obtainable by any solution of F. Importantly, as the tare term T is unconstrained by the formula F, its value can without loss of generality be assumed to be 0 at this stage, resulting in bounds on the objective value of optimal solutions as well. From now on, the algorithm commits to only searching for solutions that have $O - T$ in the specified interval, adding the unit clauses $\overline{z}_{k^*-1}$ and z_{k^*} to the working formula before proceeding to the fine convergence phase. In practice, whenever the SAT solver returns SAT after being called with assumption $\overline{z}_k$, the unit clause $\overline{z}_k$ is added immediately, allowing the SAT solver to simplify its clause database.

3.3 Fine Convergence Phase

During the fine convergence phase, assumptions for the tare variables are used to pinpoint the precise optimal value. Let k^* be the value for which the assumption $z_{k^*} = 0$ returned UNSAT in coarse convergence, and $o^* = O\restriction_{\alpha^*}$ the objective value of the currently best known solution α^*. Then we define $s := o^* - (k^* - 1) \cdot 2^P$ to be the smallest value of the tare that would force an improved solution. The next call to the SAT solver assumes $t_i = 1$ for all tare variables for which the i^{th} bit in the binary representation of s is 1. These assumptions enforce $T \geq s$, so any solution α to the working formula (which now includes the unit clause $\overline{z}_{k^*-1} \geq 1$) that extends the assumptions will satisfy $O\restriction_\alpha \geq o^* + 1$.

The fine convergence phase continues in this manner until the SAT solver reports UNSAT, at which point an optimal solution has been found. As the value of s is monotonically increasing, we add unit clauses t_i to the working formula whenever we have deduced that the i^{th} bit t_i in the tare T can safely be set to 1 in any solution (and hence in any future SAT call), which is the case when $s - 1 \geq 1 + \sum_{j=i}^{P-1} 2^i \cdot t_j$ holds. The fact that we have $s - 1$ rather than s in this last inequality is related to *stratification*, which we discuss next.

3.4 Stratification

Stratification is a technique for partitioning the indices of an objective $O = \sum_{i=1}^{m} w_i \ell_i$ into two sets $\{H, L\}$ in a way that allows computing the maximum values first of $O_H = \sum_{i \in H} w_i \ell_i$ and then of $O_L = \sum_{i \in L} w_i \ell_i$, and finally combining them to get the maximum value of O.

Specifically, stratification is applied when $\gcd\{w_i \mid i \in H\} \geq \sum_{i \in L} w_i$, i.e., when the greatest common divisor of the coefficients in O_H is at least the sum of all coefficients in O_L. SIS with the DPW encoding and stratification will first run coarse and fine convergence only on O_H as described above. At the end of the fine convergence, the SAT solver returns UNSAT after being invoked with assumptions that enforce $T_H \geq s$ for the tare term T_H added to the DPW encoding of O_H and some constant s. At this stage, the value of T_H will be fixed to $s - 1$ with unit clauses, effectively fixing O_H to its maximum value. This fixing of O_H is consistent with the unit clauses learned in the previous section. After this O_L is optimized via coarse and fine convergence under the fixed value of O_H. The solution obtained at the end of the final fine convergence phase will be optimal with respect to the original instance. For more details on stratification, we refer the reader to [2, 51].

▶ **Example 3.** Consider the objective $O = 10x_1 + 5x_2 + 5x_3 + 3x_4 + 2x_5$ and the partition $H = \{1, 2, 3\}$ and $L = \{4, 5\}$. Since $\gcd\{10, 5, 5\} = 5 \geq 3 + 2$, changes of the objective restricted to $\{x_1, x_2, x_3\}$ will dominate any contributions from $3x_4 + 2x_5$. If a solution α with $O_H{\restriction}_\alpha = 15$ is found, we can without loss of generality assume $O_H \geq 15$, since for any solution β with $O_H{\restriction}_\beta < 15$ we have $O{\restriction}_\beta \leq O{\restriction}_\alpha$. Notice that maximizing first O_H and then O_L can remove some optimal solutions from the search space, but never all of them.

4 Certifying Solution-Improving MaxSAT with the DPW Encoding

We are now ready to describe how to do proof logging for solution-improving MaxSAT with the dynamic polynomial watchdog encoding. In addition to certifying the correctness of CNF encodings, as done in previous work on proof logging SIS for MaxSAT [59, 60], we need to certify the without loss of generality reasoning discussed in Section 3. This turns out to require quite intricate proof logging methods.

We start with a brief discussion how to certify the DPW encoding. We then turn to proof logging for the without loss of generality reasoning during the coarse and fine convergence phases. Afterwards, we deal with proof logging for stratification. We defer a discussion of minor additional heuristics used in state-of-the-art solvers to Appendix B. We note that for all clauses learned by the SAT solver we can use standard VeriPB proof logging, and since all such learned clauses are logically implied by the working formula it is safe to add them to the derived set $\mathcal{D}$. This means that we can ignore all constraints added to the database by the SAT solver when we perform redundance-based strengthening steps.

4.1 Proof Logging for Clauses of the DPW Encoding

Figure 1 depicts the structure of the DPW encoding of the term $2x_1 + 3x_2 + 5x_3 + 7x_4$. For a term L in which the largest coefficient has P bits, the encoding introduces P totalizers [5] (which are circuits that sort their inputs), and $P - 1$ mergers. The i^{th} totalizer takes as input all variables in L for which the corresponding coefficient has its i^{th} bit equal to 1.

Proof logging for the DPW encoding boils down to taking care of the totalizer encodings as described in [60]. At a high level, the proof for $\mathsf{PW}(O - T)$ derives a number of constraints encoding implications $y \Rightarrow C_y$ and $y \Leftarrow C_y$, where y are variables in the auxiliary variable set Y and C_y are suitably chosen PB constraints over the variables in $O - T$. A concrete

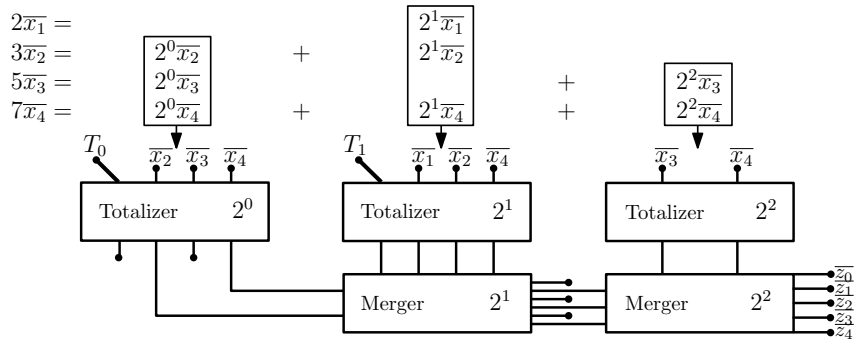

Figure 1 Illustration of the polynomial watchdog encoding.

example is the output variable z_k for which the constraint C_{z_k} is chosen as $O - T \leq k \cdot 2^P$. From these pseudo-Boolean definitions all clauses in the CNF encoding added to the solver database can be derived with explicit VeriPB derivations. A technical point that is crucial for the proof logging is that in this way we only need to add the PB definitions of new variables to the core set $\mathcal{C}$. The clauses actually used for the SAT solver calls are implied from these definitions, and can therefore be placed in the derived set $\mathcal{D}$.

4.2 Proofs Without Loss of Generality Using Shadow Circuits

The MaxSAT solving algorithm uses without loss of generality (wlog) reasoning when (i) introducing fresh variables for encoding $\mathsf{PW}(O - T)$; (ii) adding unit clauses $\overline{z}_k$ during coarse convergence; (iii) learning unit clause over the tare variables t_i during fine convergence; and (iv) concluding that the optimal value has been found.

To see why unit clauses $\overline{z}_k \geq 1$ require wlog reasoning, suppose in the coarse convergence phase that the SAT solver returns a solution α when invoked with the assumption $z_k = 0$, indicating that $(O - T)\restriction_\alpha \geq 1 + k \cdot 2^P$. The constraint $\overline{z}_k \geq 1$ is *not* entailed by the solution-improving constraint $O \geq O\restriction_\alpha$, since some other (possibly optimal) solution β might have $O\restriction_\beta \geq O\restriction_\alpha$ but assign the tare variables so that $(O - T)\restriction_\beta < 1 + k \cdot 2^P \leq (O - T)\restriction_\alpha$ holds. However, since the tare variables are not constrained by the original formula F, any solution to F could be extended to any fixed value for the tare T. Hence, in particular, we can assume without loss of generality that $T = 0$, which in turn implies that $\overline{z}_k \geq 1$.

The fine convergence phase makes use of the fact that the DPW encoding does not constrain T, which takes values in the range $[0, 2^P - 1]$. The unit clauses $t_i \geq 1$ learned are not entailed, but can be deduced since the tare variables are unconstrained in the DPW encoding. This requires a VeriPB proof that wlog $T \geq s - 1$. When the SAT solver reports UNSAT during fine convergence, it does so under the assumption that a specific set of tare variables take value 1. If this yields UNSAT, then we can conclude that the current solution is optimal (since we can wlog assume T to be equal to the value that led to UNSAT).

It is worth noticing that the without loss of generality arguments above are quite intricate even at a human meta-level. The coarse convergence phase repeatedly claims to be able to assume $T = 0$, after which the fine convergence phase picks an increasing sequence $0 < s_1 < s_2 < \ldots$ and assumes $T \geq s_i - 1$ wlog. Finally, a specific value $T = s_{i^*}$ is used to argue about optimality. The meta-level argument for why this works is that no conclusions are drawn from the assumptions made during coarse and fine convergence that invalidate subsequent assumptions. The challenge is how to convince a mechanical proof checker of this.

Consider first proof logging for the coarse convergence phase, and suppose the solver returns SAT when invoked with assumption $\overline{z}_k$. The only rule that would allow us to derive $\overline{z}_k \geq 1$ without loss of generality (from the argument that we can set $T = 0$ wlog) is *redundance-based strengthening*, which requires specification of a witness substitution ω that can be used to "patch" any assignment α in which $\overline{z}_k \geq 1$ is violated. More formally, our witness should guarantee that $\mathcal{C} \cup \mathcal{D} \cup \{\neg(\overline{z}_k \geq 1)\} \models \big(\mathcal{C} \cup \{\overline{z}_k \geq 1\}\big)\restriction_\omega \cup \{O \leq O\restriction_\omega\}$. A natural approach would be to choose a witness ω that maps (i) z_k to 0, (ii) all original variables to themselves, and (iii) T to 0. Such a witness would make $(\overline{z}_k \geq 1)\restriction_\omega$ trivially true and would incur no proof obligations for the formula F or the objective O. However, setting $T = 0$ will not work for the constraints $C \in \mathcal{C}$ defining variables in the DPW encoding. If we fix $T = 0$, then we also need to update all auxiliary variables Y in the circuit evaluating $\mathsf{PW}(O - T)$. But how this should be done depends on which assignment α we need to patch, and the redundance rule has no mechanism for defining "conditional witnesses" $\omega = \omega(\alpha)$.

To determine how the witness should assign the auxiliary variables in $\mathsf{PW}(O - T)$, we devise a new proof logging technique that we call *shadow circuits*. Corresponding to each auxiliary variable y defined as the reification of a PB constraint C_y in the original circuit, a *shadow circuit for a fixed value* v has a fresh variable $y^{T=v}$ defined by $y^{T=v} \Leftrightarrow C_y\restriction_{T\mapsto v}$. In words, the defining constraints of $y^{T=v}$ and y are the same except that we fix the tare variables t_i so that $T = v$. The definitions of such shadow circuits are stored in the core set $\mathcal{C}$ since they are derived using the redundance rule. Note that the shadow circuit only "copies" the pseudo-Boolean definitions of the variables and not their clausal encodings.

Shadow circuits provide us with a mechanism to compute witnesses for the redundance rule that allow us to assume the value of T and certify the without loss of generality reasoning. During coarse convergence, each addition of a constraint $\overline{z}_i \geq 1$ is logged with a witness that maps all tare variables t_i to 0 and other auxiliary variables y in $\mathsf{PW}(O - T)$ to their counterparts $y^{T=0}$ in the shadow circuit for $T = 0$. During fine convergence, the constraints $T \geq s - 1$ are derived using shadow circuits for $s - 1$, which allows adding unit constraints over individual tare variables to the proof. Finally, for proving optimality a shadow circuit for the final value s^* for which the SAT solver returned UNSAT will be used to derive contradiction.

The next proposition gives a more formal summary of the wlog proof logging performed during the coarse convergence phase. The proof for this proposition, together with precise descriptions of the other wlog proof logging steps, are given in Appendix A.

▶ **Proposition 4.** *Suppose the VeriPB proof log contains derivations of reification constraints* $\overline{z}_k \Leftrightarrow O - T \geq 1 + k \cdot 2^P$ *and a shadow circuit for* $T = 0$ *as well as the constraint* $O \geq 1 + k \cdot 2^P$. *Then the constraint* $\overline{z}_k \geq 1$ *can be derived using redundance-based strengthening with witness* $\omega = \{t_i \mapsto 0 \mid 0 \leq i \leq P - 1\} \cup \big\{y \mapsto y^{T=0} \mid y \in Y\big\}$.

The constraint $O \geq 1 + k \cdot 2^P$ in Proposition 4 can be obtained by weakening the solution-improving constraint $O \geq O\restriction_\alpha + 1$ for the previously found solution α. If stratification is used, deriving $O_H \geq 1 + k \cdot 2^P$ requires more work (see Section 3.4 for details).

Our technique with shadow circuits and repeated without loss of generality arguments selecting (different) values for the same variables in T heavily relies on that VeriPB proofs in the *strengthening-to-core* mode maintain the guarantee that all constraints in the derived set $\mathcal{D}$ are entailed by the core set $\mathcal{C}$. In particular, what this means is that whenever we want to apply redundance-based strengthening, fixing tare variables and using the corresponding shadow circuit, we do not need to worry about reproving any clauses learned by the SAT solver under the witness ω. It turns out that for all non-trivial proof obligations, the solution-improving constraint $O \geq O\restriction_\alpha$ for the latest solution α obtained is helpful. This also makes it easier to see why the entire pipeline is consistent. During coarse convergence, we never derive

$T = 0$, but instead derive $z_k = 0$ for certain values of k using the fact that we could set $T = 0$ wlog. This constraint $z_k = 0$ will be used by the solver for deriving several consequences. Later, when we make the wlog argument that $T \geq s - 1$ for some value s, this incurs the obligation to reprove that $z_k = 0$ holds! That is, the proof checker realizes that $z_k = 0$ was also derived wlog, and we need to prove that this is still consistent with the current wlog assumption to justify that we can "change our mind" about the value of T.

The use of *strengthening-to-core* requires some extra care when dealing with constraint deletions. SAT solvers use heuristics to aggressively erase clauses that are believed to no longer be useful, and this is crucial for performance. Also, clauses in the input are removed whenever some literal in the clause is deduced to be true. In strengthening-to-core mode, we can still do unrestricted deletions of constraints in the derived set $\mathcal{D}$, but a core constraint $C \in \mathcal{C}$ can only be erased if the implication $\mathcal{C} \setminus \{C\} \models C$ can be shown to hold. For this reason we did not implement deletion from the core set in our proof logging routines.

4.3 Stratification

For proof logging of stratification steps as in Section 3.4, we need to be able to convert known facts about the whole objective O to statements about the split objectives O_H and O_L. To certify a unit constraint added during coarse convergence or to derive the constraints $T \geq s - 1$ during fine convergence when maximizing O_H, we need to derive $O_H \geq O_H\restriction_\alpha$ from $O \geq O\restriction_\alpha + 1$. We do this by weakening away all terms in O_L – meaning that for every term $w_i\ell_i$ in O_L we add $w_i\bar{\ell}_i \geq 0$ to cancel the term – to get $O_H \geq O\restriction_\alpha + 1 - g$, where g is the greatest common divisor of the coefficients in O_H. This clearly also entails $O_H \geq O_H\restriction_\alpha - g + 1$. Dividing by g and rounding up yields $\frac{1}{g}O_H \geq \frac{O_H\restriction_\alpha}{g} - 1 + 1$, and multiplying this again by g yields $O_H \geq O_H\restriction_\alpha$.

By applying this reasoning, we can derive the constraint $O_H \geq o_H^*$ right after finding the optimal value o_H^* for O_H. Moreover, after introducing a shadow circuit for $T = s$, we can derive (local) optimality in the form of the constraint $O_H \leq o_H^*$. Hence, we can reformulate the objective by replacing O_H with the constant o_H^*, from which we can now derive the constraint $O_L + o_H^* \geq O\restriction_\alpha + 1$. Observe that this constraint coincides with the solution-improving constraint for O_L. Once the constraints $O_L \geq o_L^*$ and $O_L \leq o_L^*$ have been derived in a similar way, the objective will be rewritten to a constant, for which proving optimality boils down to logging a solution that has objective value $o^* = o_H^* + o_L^*$.

4.4 Limiting the Use of Shadow Circuits

Our proof logging method makes repeated use of shadow circuits, which are copies of the original circuit, and repeatedly deriving all constraints defining such circuits could potentially incur serious overhead for proof generation in the solver. Let us discuss ways of limiting or completely eliminating the use of shadow circuits and the downside of such approaches.

First, the shadow circuits are introduced each time the solver deduces a unit clause over an output variable z_k or tare variable t_i. Instead of learning these unit clauses, we could do all subsequent solver calls with those literals as assumptions. At the very end of the fine convergence phase, we could then introduce a single shadow circuit to prove optimality (or, in case of stratification, two shadow circuits: one to prove optimality and one to fix the value of the tare variables). The disadvantage is that when variables used as assumptions, the solver cannot use them to simplify its clause database; so while this would have a positive effect on the time required to do the actual proof logging, it could have negative effects on solving time. Appendix C.2 reports on an experimental evaluation of this approach.

Second, there is a way to completely eliminate shadow circuits. By the end of the execution, the solver knows which value $T = s$ resulted in the final *UNSAT* call in the fine convergence. What we could do at this point is insert at the *beginning* of the proof constraints saying that $T = s$ holds (which at this point can easily be derived by redundance-based strengthening). The rest of the proof will then be checked for a fixed value of T that happens to be the value needed at the end. There are two important reasons why we prefer the shadow circuit approach. The first reason is that it is not clear if and how this would work together with stratification, where after a stratification level we want to fix $T = s - 1$. The second reason is that fixing T in advance adds substantial new information that the solver did not have available when constructing the proof. This means that we would not be verifying that the reasoning the solver actually performed was correct, but only that its reasoning checks out given advance information about the optimal solution. While this could still be used to certify the correctness of the final answer, it would not provide any guarantees about the process leading there. It has been shown repeatedly that proof logging can catch subtle bugs in solvers that only report correct results but for the wrong reasons [9, 24, 32, 41], but in order for this to be possible the correctness of solver-generated proofs should only depend on what the solver actually knows when the proof is being produced.

4.5 Discussion of an Even Simpler Approach and Why It Does Not Work

The proof logging techniques in this paper certify every single reasoning step in the solver. An alternative, and seemingly much simpler, way to get proofs of correctness for *any* MaxSAT solver would be to (i) compute an optimal solution by running the MaxSAT solver without proof logging, (ii) check that this solution is feasible, (iii) encode a solution-improving constraint into CNF, and (iv) call a SAT solver to generate a proof of unsatisfiability (and hence of optimality of the solution) with standard SAT proof logging. However, there are several serious issues with this approach that we would like to point out.

First, proofs of correctness are needed for the CNF encodings used in step (iii), and such proofs cannot be done with SAT proof logging since it cannot reason about values of objective functions. Second, it is not possible to just repeat the "final UNSAT call" of the MaxSAT solver in step (iv). Even if the same SAT solver is used, in the original UNSAT call this solver had access to all constraints learned in previous calls, and there is no guarantee that the solver will learn these constraints again, or other equally good constraints, when it is now run in a different way and with a different input. It is therefore impossible to know for sure whether the final SAT solver invocation with the solution-improving constraint would be faster or, more likely, slower, than the original solving process, and by how much. This defeats the whole idea of generating proofs with a small and predictable overhead, since there would be no way of knowing in advance whether "proof logging" for a previously claimed result would succeed or not. Moreover, when a solution-improving MaxSAT solver makes use of stratification (as discussed in Section 3.4), then optimality is not derived by a single UNSAT call but by a combination of UNSAT calls at different levels. It is hard to see how such a combination of calls could be replicated with the simple approach described above.

Third, an increasingly popular usage scenario for MaxSAT solvers is so-called anytime solving, where the solver can be terminated at any point and then returns the best upper and lower bounds on the objective computed so far. Proofs constructed as described in this paper (as well as in other MaxSAT papers using VeriPB proof logging) will at all times contain formal proofs of everything the solver knows about upper and lower bounds on the objective. Whenever the solver is terminated, it can therefore just end the generated proof at that point by printing a concluding line stating what upper and lower bounds have been proven. This functionality would be lost in the alternative approach.

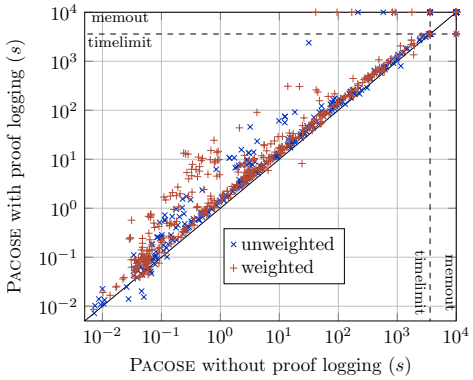

Figure 2 Proof logging overhead for PACOSE using the DPW encoding.

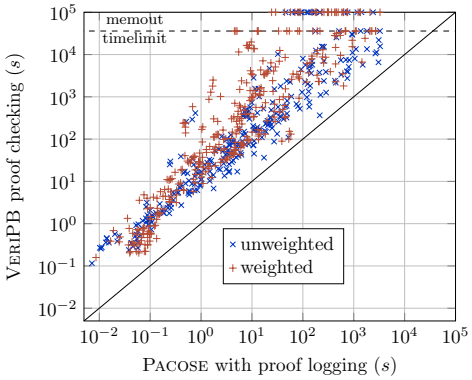

Figure 3 PACOSE vs. VERIPB running time using DPW encoding.

Finally, even if this approach could be made to work efficiently – which, as explained above, is not really the case, for several reasons – we would have the same problem as in Section 4.4 that we would only certify the final result and not the solver reasoning process.

5 Experimental Evaluation

To evaluate our proof logging approach in practice, we implemented it in the state-of-the-art solution-improving MaxSAT solver PACOSE [52]. The source code for all software tools used, as well as all experimental data, are available in [8]. During development, we extensively checked the correctness of our implementation with a fuzzer [50] and minimized failed instances with a delta debugger. This process accelerated the development, as we did not need to create instances for special cases, and helped us fix unexpected and sporadic bugs. The proofs emitted by PACOSE were verified by the pseudo-Boolean proof checker VERIPB [61], and our fuzzing also helped to debug the proof checker.

The experiments were performed on identical machines with an 11th Gen Intel(R) Core(TM) i5-1145G7 @ 2.60 GHz CPU and 16 GB of memory. Each benchmark ran exclusively on a machine and the memory limit was set to 14 GB. The time limits were set to 3 600 seconds for solving a MaxSAT instance with PACOSE and to 36 000 seconds for checking the proof with VERIPB. As our benchmark set we used the 558 weighted and 572 unweighted MaxSAT instances from the MaxSAT Evaluation 2023 [47].

Our implementation supports all techniques PACOSE employed in the MaxSAT Evaluation 2023. This means that in addition to the dynamic polynomial watchdog encoding we also implemented proof logging for the binary adder encoding [62] following the approach in [31,59] as well as support for stratification as described in Section 3.4 and for the preprocessing techniques in TRIMMAXSAT [51]. Appendix B discusses TRIMMAXSAT in detail and Appendix C contains detailed experimental results for the default setup in which PACOSE employs heuristics to choose between different encodings. In this section, we focus on the main novelty of this paper, namely proof logging for SIS with the DPW encoding.

To show the viability of enabling proof logging while solving, we analyse the overhead of generating proofs. In Figure 2 we compare the running time of PACOSE with and without proof logging. With proof logging enabled 674 instances were solved within the resource limits, which is 11 fewer instances than without proof logging. Out of the 11 instances that were not solved with proof logging enabled, 9 instances failed due to the memory limit and 2 instances due to the time limit. For the solved instances, PACOSE with proof logging was

on average 1.93× slower than without proof logging. About 90% of the solved instances were solved at most 5.26× more slowly with proof logging enabled. This overhead for solving is to some extent caused by our shadow circuits approach. While we demonstrate that shadow circuits can be used to justify the without loss of generality reasoning in Pacose, it remains to investigate whether there is a better approach. It is important to note, though, that the average overhead of 1.93× is heavily biased by small instances: the cumulative solving time of all 674 instances, with proof logging is only 1.32× the cumulative solving time without proof logging. This suggests that proof logging overhead decreases for harder instances.

For proof logging to be maximally useful in practice, it is also desirable that it should be possible to check generated proofs within a time limit that is some small constant factor of the solving time for the instance. To evaluate the efficiency of proof checking, we compared the running time of Pacose with proof logging enabled with the running time of VeriPB, with results plotted in Figure 3. Out of the 674 instances solved by Pacose with proof logging, 592 were successfully checked by VeriPB, but 53 instances failed due to the memory limit and 29 instances due to the time limit. On average, checking the proof with VeriPB was 22.5× slower than solving and generating the proof with Pacose. 90% of the proofs were checked within 100× the running time of Pacose. These results for checking are in line with what has been reported in other works on proof logging for MaxSAT [9, 59]. While there is certainly room for further improvements, this shows that proof logging and checking is viable. It should also be emphasized that the only sources of problems for VeriPB were the time and memory limits – other than that all proofs were successfully checked.

6 Conclusion

In this paper, we demonstrate how to design proof logging for solution-improving MaxSAT solving using the dynamic polynomial watchdog encoding. This turns out to be surprisingly challenging, mainly due to the heavy use of reasoning without loss of generality. To understand the correctness of this reasoning at a human level is one thing, but convincing a proof checker by producing machine-verifiable proofs is quite another. What we show is that by combining the redundance-based strengthening rule and the strengthening-to-core mode in VeriPB, together with a technique we call shadow circuits for having more expressive witnessing capabilities, we are able to devise efficient pseudo-Boolean proof logging techniques.

We have implemented our approach in the state-of-the-art MaxSAT solver Pacose. Our experimental evaluation shows that while enabling proof logging is feasible, it does incur a non-negligible overhead in solving time. Moreover, the time needed to check the generated proofs is several times larger than the time needed to generate them, suggesting that more efficient algorithms and more optimized engineering are needed in VeriPB. This is not so surprising, since the focus of VeriPB development so far has been on providing support for certifying algorithms in combinatorial optimization paradigms previously beyond the reach of proof logging, rather than on optimizing the proof checker code base.

The addition of Pacose to the collection of certifying MaxSAT solvers using VeriPB proofs provides further support to the hypothesis that pseudo-Boolean proof logging hits a sweet spot for MaxSAT solving, being rich enough to support a wide variety of solving algorithms and complex reasoning tricks, but still being simple enough to support even formally verified proof checking as in [13, 33, 37].

We believe that in the longer term VeriPB can have a strong positive impact on the reliability and robustness of MaxSAT solvers. In the other direction, MaxSAT solving is likely to provide excellent benchmarks and performance challenges to further improve pseudo-Boolean proof logging and checking. Our suggestion for speeding up these developments is to introduce a certifying track in the yearly MaxSAT Evaluation [46].

References

1 Eyad Alkassar, Sascha Böhme, Kurt Mehlhorn, Christine Rizkallah, and Pascal Schweitzer. An introduction to certifying algorithms. *it - Information Technology Methoden und innovative Anwendungen der Informatik und Informationstechnik*, 53(6):287–293, December 2011.

2 Josep Argelich, Inês Lynce, and João P. Marques-Silva. On solving Boolean multilevel optimization problems. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI '09)*, pages 393–398, July 2009.

3 Fahiem Bacchus, Matti Järvisalo, and Ruben Martins. Maximum satisfiabiliy. In 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*, chapter 24, pages 929–991. IOS Press, 2nd edition, February 2021.

4 Seulkee Baek, Mario Carneiro, and Marijn J. H. Heule. A flexible proof format for SAT solver-elaborator communication. In *Proceedings of the 27th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS '21)*, volume 12651 of *Lecture Notes in Computer Science*, pages 59–75. Springer, March-April 2021.

5 Olivier Bailleux and Yacine Boufkhad. Efficient CNF encoding of Boolean cardinality constraints. In *Proceedings of the 9th International Conference on Principles and Practice of Constraint Programming (CP '03)*, volume 2833 of *Lecture Notes in Computer Science*, pages 108–122. Springer, September 2003.

6 Olivier Bailleux, Yacine Boufkhad, and Olivier Roussel. New encodings of pseudo-Boolean constraints into CNF. In *Proceedings of the 12th International Conference on Theory and Applications of Satisfiability Testing (SAT '09)*, volume 5584 of *Lecture Notes in Computer Science*, pages 181–194. Springer, June 2009.

7 Haniel Barbosa, Andrew Reynolds, Gereon Kremer, Hanna Lachnitt, Aina Niemetz, Andres Nötzli, Alex Ozdemir, Mathias Preiner, Arjun Viswanathan, Scott Viteri, Yoni Zohar, Cesare Tinelli, and Clark Barrett. Flexible proof production in an industrial-strength SMT solver. In *Proceedings of the 11th International Joint Conference on Automated Reasoning (IJCAR '22)*, volume 13385 of *Lecture Notes in Computer Science*, pages 15–35. Springer, August 2022.

8 Jeremias Berg, Bart Bogaerts, Jakob Nordström, Andy Oertel, Tobias Paxian, and Dieter Vandesande. Experimental Repository for "Certifying Without Loss of Generality Reasoning in Solution-Improving Maximum Satisfiability", June 2024. `doi:10.5281/zenodo.10826301`.

9 Jeremias Berg, Bart Bogaerts, Jakob Nordström, Andy Oertel, and Dieter Vandesande. Certified core-guided MaxSAT solving. In *Proceedings of the 29th International Conference on Automated Deduction (CADE-29)*, volume 14132 of *Lecture Notes in Computer Science*, pages 1–22. Springer, July 2023.

10 Armin Biere. Tracecheck. `http://fmv.jku.at/tracecheck/`, 2006.

11 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, February 2021.

12 Bart Bogaerts, Stephan Gocht, Ciaran McCreesh, and Jakob Nordström. Certified dominance and symmetry breaking for combinatorial optimisation. *Journal of Artificial Intelligence Research*, 77:1539–1589, August 2023. Preliminary version in *AAAI '22*.

13 Bart Bogaerts, Ciaran McCreesh, Magnus O. Myreen, Jakob Nordström, Andy Oertel, and Yong Kiam Tan. Documentation of VeriPB and CakePB for the SAT competition 2023. Available at `https://satcompetition.github.io/2023/checkers.html`, March 2023.

14 Maria Luisa Bonet, Jordi Levy, and Felip Manyà. Resolution for Max-SAT. *Artificial Intelligence*, 171(8-9):606–618, June 2007. Extended version of paper in *SAT '06*.

15 Robert Brummayer and Armin Biere. Fuzzing and delta-debugging SMT solvers. In *Proceedings of the 7th International Workshop on Satisfiability Modulo Theories (SMT '09)*, pages 1–5, August 2009.

16 Robert Brummayer, Florian Lonsing, and Armin Biere. Automated testing and debugging of SAT and QBF solvers. In *Proceedings of the 13th International Conference on Theory and Applications of Satisfiability Testing (SAT '10)*, volume 6175 of *Lecture Notes in Computer Science*, pages 44–57. Springer, July 2010.

17 Samuel R. Buss and Jakob Nordström. Proof complexity and SAT solving. In 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*, chapter 7, pages 233–350. IOS Press, 2nd edition, February 2021.

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

19 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, September 2013.

20 Jessica Davies and Fahiem Bacchus. Exploiting the power of MIP solvers in MAXSAT. In *Proceedings of the 16th International Conference on Theory and Applications of Satisfiability Testing (SAT '13)*, volume 7962 of *Lecture Notes in Computer Science*, pages 166–181. Springer, July 2013.

21 Jasper van Doornmalen, Leon Eifler, Ambros Gleixner, and Christopher Hojny. A proof system for certifying symmetry and optimality reasoning in integer programming. Technical Report 2311.03877, arXiv.org, November 2023.

22 Niklas Eén and Niklas Sörensson. Temporal induction by incremental SAT solving. In *Proceedings of the 1st International Workshop on Bounded Model Checking (BMC '03)*, volume 89 of *Electronic Notes in Theoretical Computer Science*, pages 543–560, July 2003.

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

24 Leon Eifler and Ambros Gleixner. A computational status update for exact rational mixed integer programming. *Mathematical Programming*, 197(2):793–812, February 2023.

25 Salomé Eriksson and Malte Helmert. Certified unsolvability for SAT planning with property directed reachability. In *Proceedings of the 30th International Conference on Automated Planning and Scheduling*, pages 90–100, October 2020.

26 Salomé Eriksson, Gabriele Röger, and Malte Helmert. Unsolvability certificates for classical planning. In *Proceedings of the 27th International Conference on Automated Planning and Scheduling (ICAPS '17)*, pages 88–97, June 2017.

27 Salomé Eriksson, Gabriele Röger, and Malte Helmert. A proof system for unsolvable planning tasks. In *Proceedings of the 28th International Conference on Automated Planning and Scheduling (ICAPS '18)*, pages 65–73, June 2018.

28 Mathias Fleury. *Formalization of Logical Calculi in Isabelle/HOL*. PhD thesis, Universität des Saarlandes, 2020. Available at `https://publikationen.sulb.uni-saarland.de/handle/20.500.11880/28722`.

29 Zhaohui Fu and Sharad Malik. On solving the partial MAX-SAT problem. In *Proceedings of the 9th International Conference on Theory and Applications of Satisfiability Testing (SAT '06)*, volume 4121 of *Lecture Notes in Computer Science*, pages 252–265. Springer, August 2006.

30 Xavier Gillard, Pierre Schaus, and Yves Deville. SolverCheck: Declarative testing of constraints. In *Proceedings of the 25th International Conference on Principles and Practice of Constraint Programming (CP '19)*, volume 11802 of *Lecture Notes in Computer Science*, pages 565–582. Springer, October 2019.

31 Stephan Gocht, Ruben Martins, Jakob Nordström, and Andy Oertel. Certified CNF translations for pseudo-Boolean solving. In *Proceedings of the 25th International Conference on Theory and Applications of Satisfiability Testing (SAT '22)*, volume 236 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 16:1–16:25, August 2022.

32 Stephan Gocht, Ross McBride, Ciaran McCreesh, Jakob Nordström, Patrick Prosser, and James Trimble. Certifying solvers for clique and maximum common (connected) subgraph problems. In *Proceedings of the 26th International Conference on Principles and Practice of Constraint Programming (CP '20)*, volume 12333 of *Lecture Notes in Computer Science*, pages 338–357. Springer, September 2020.

33 Stephan Gocht, Ciaran McCreesh, Magnus O. Myreen, Jakob Nordström, Andy Oertel, and Yong Kiam Tan. End-to-end verification for subgraph solving. In *Proceedings of the 368h AAAI Conference on Artificial Intelligence (AAAI '24)*, pages 8038–8047, February 2024.

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

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

36 Alexander Hoen, Andy Oertel, Ambros Gleixner, and Jakob Nordström. Certifying MIP-based presolve reductions for 0–1 integer linear programs. In *Proceedings of the 21st International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR '24)*, volume 14742 of *Lecture Notes in Computer Science*, pages 310–328. Springer, May 2024.

37 Hannes Ihalainen, Andy Oertel, Yong Kiam Tan, Jeremias Berg, Matti Järvisalo, Magnus O. Myreen, and Jakob Nordström. Certified MaxSAT preprocessing. In *Proceedings of the 12th International Joint Conference on Automated Reasoning (IJCAR '24)*, volume 14739 of *Lecture Notes in Computer Science*, pages 396–418. Springer, July 2024.

38 Saurabh Joshi, Ruben Martins, and Vasco M. Manquinho. Generalized totalizer encoding for pseudo-Boolean constraints. In *Proceedings of the 21st International Conference on Principles and Practice of Constraint Programming (CP '15)*, volume 9255 of *Lecture Notes in Computer Science*, pages 200–209. Springer, august-september 2015.

39 Matti Järvisalo, Marijn J. H. Heule, and Armin Biere. Inprocessing rules. In *Proceedings of the 6th International Joint Conference on Automated Reasoning (IJCAR '12)*, volume 7364 of *Lecture Notes in Computer Science*, pages 355–370. Springer, June 2012.

40 Michal Karpinski and Marek Piotrów. Encoding cardinality constraints using multiway merge selection networks. *Constraints*, 24(3–4):234–251, October 2019.

41 Sonja Kraiczy and Ciaran McCreesh. Solving graph homomorphism and subgraph isomorphism problems faster through clique neighbourhood constraints. In *Proceedings of the 30th International Joint Conference on Artificial Intelligence (IJCAI '21)*, pages 1396–1402, August 2021.

42 Javier Larrosa, Robert Nieuwenhuis, Albert Oliveras, and Enric Rodríguez-Carbonell. A framework for certified Boolean branch-and-bound optimization. *Journal of Automated Reasoning*, 46(1):81–102, January 2011.

43 Marcus Leivo, Jeremias Berg, and Matti Järvisalo. Preprocessing in incomplete MaxSAT solving. In *Proceedings of the 24th European Conference on Artificial Intelligence (ECAI '20)*, volume 325 of *Frontiers in Artificial Intelligence and Applications*, pages 347–354, August-September 2020.

44 Chu Min Li and Felip Manyà. MaxSAT, hard and soft constraints. In 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*, chapter 23, pages 903–927. IOS Press, 2nd edition, February 2021.

45 Norbert Manthey, Tobias Philipp, and Peter Steinke. A more compact translation of pseudo-Boolean constraints into CNF such that generalized arc consistency is maintained. In *Proceedings of the 37th Annual German Conference on Artificial Intelligence (KI '14)*, volume 8736 of *Lecture Notes in Computer Science*, pages 123–134. Springer, September 2014.

46 MaxSAT evaluations: Evaluating the state of the art in maximum satisfiability solver technology. `https://maxsat-evaluations.github.io/`.

47 MaxSAT evaluation 2023. `https://maxsat-evaluations.github.io/2023`, July 2023.

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

49 António Morgado and João P. Marques-Silva. On validating Boolean optimizers. In *Proceedings of the 23rd IEEE International Conference on Tools with Artificial Intelligence (ICTAI '12)*, pages 924–926, November 2011.

50 Tobias Paxian and Armin Biere. Uncovering and classifying bugs in MaxSAT solvers through fuzzing and delta debugging. In *Proceedings of the 14th International Workshop on Pragmatics of SAT*, volume 3545 of *CEUR Workshop Proceedings*, pages 59–71. CEUR-WS.org, July 2023.

51 Tobias Paxian, Pascal Raiola, and Bernd Becker. On preprocessing for weighted MaxSAT. In *Proceedings of the 22nd International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI '21)*, volume 12597 of *Lecture Notes in Computer Science*, pages 556–577. Springer, January 2021.

52 Tobias Paxian, Sven Reimer, and Bernd Becker. Dynamic polynomial watchdog encoding for solving weighted MaxSAT. In *Proceedings of the 21st International Conference on Theory and Applications of Satisfiability Testing (SAT '18)*, volume 10929 of *Lecture Notes in Computer Science*, pages 37–53. Springer, July 2018.

53 Matthieu Py, Mohamed Sami Cherif, and Djamal Habet. Towards bridging the gap between SAT and Max-SAT refutations. In *Proceedings of the 32nd IEEE International Conference on Tools with Artificial Intelligence (ICTAI '20)*, pages 137–144, November 2020.

54 Matthieu Py, Mohamed Sami Cherif, and Djamal Habet. A proof builder for Max-SAT. In *Proceedings of the 24th International Conference on Theory and Applications of Satisfiability Testing (SAT '21)*, volume 12831 of *Lecture Notes in Computer Science*, pages 488–498. Springer, July 2021.

55 Matthieu Py, Mohamed Sami Cherif, and Djamal Habet. Proofs and certificates for Max-SAT. *Journal of Artificial Intelligence Research*, 75:1373–1400, December 2022.

56 Gabriele Röger. Towards certified unsolvability in classical planning. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence (IJCAI '17)*, pages 5141–5145, August 2017.

57 Hans-Jörg Schurr, Mathias Fleury, Haniel Barbosa, and Pascal Fontaine. Alethe: Towards a generic SMT proof format (extended abstract). In *Proceedings of the 7th Workshop on Proof eXchange for Theorem Proving (PxTP '21*, volume 336 of *Electronic Proceedings in Theoretical Computer Science*, pages 49–54, July 2021.

58 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)*, volume 3709 of *Lecture Notes in Computer Science*, pages 827–831. Springer, October 2005.

59 Dieter Vandesande. Towards certified MaxSAT solving: Certified MaxSAT solving with SAT oracles and encodings of pseudo-Boolean constraints. Master's thesis, Vrije Universiteit Brussel (VUB), 2023. URL: `https://researchportal.vub.be/nl/studentTheses/towards-certified-maxsat-solving`.

60 Dieter Vandesande, Wolf De Wulf, and Bart Bogaerts. QMaxSATpb: A certified MaxSAT solver. In *Proceedings of the 16th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR '22)*, volume 13416 of *Lecture Notes in Computer Science*, pages 429–442. Springer, September 2022.

61 VeriPB: Verifier for pseudo-Boolean proofs. `https://gitlab.com/MIAOresearch/software/VeriPB`.

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

63 Nathan Wetzler, Marijn J. H. Heule, and Warren A. Hunt Jr. DRAT-trim: Efficient checking and trimming using expressive clausal proofs. In *Proceedings of the 17th International Conference on Theory and Applications of Satisfiability Testing (SAT '14)*, volume 8561 of *Lecture Notes in Computer Science*, pages 422–429. Springer, July 2014.

A Formalization of the Proof Logging of SIS with the DPW

In this appendix, we provide formal details on the claims made in the main body of the paper. In the proofs, we follow the same notation. The formalization of the reasoning in the coarse convergence is discussed in Section 4.2, here we discuss the other phases.

A.1 Coarse Convergence

Our first proposition formalizes the wlog performed during the coarse convergence phase.

▶ **Proposition 5** (Proposition 4, restated). *Assume the definition of z_k has been derived and a complete shadow circuit for $T = 0$ has been introduced. Furthermore assume the constraint*

$$O \geq 1 + k \cdot 2^P \tag{4}$$

has been derived. The constraint $\overline{z}_k \geq 1$ can be derived using redundance-based strengthening with witness

$$\omega = T \mapsto 0, Y \mapsto Y^{T=0}.$$

The notation for the witness in this proposition is a shorthand for the mapping that sends each variable t_i to 0 and every introduced circuit variable y to the corresponding shadow circuit variable $y^{T=0}$.

Proof. To verify this is indeed possible, we need to show that from

$$\mathcal{C} \cup \mathcal{D} \cup \{z_k \geq 1\}$$

we can derive the following constraints:

- $\overline{z}_k\restriction_\omega \geq 1$; in other words we need to show that $\overline{z}_k^{T=0} \geq 1$ holds. Recall that $z_k^{T=0}$ is defined by the reification

  $$\overline{z}_k^{T=0} \Leftrightarrow O - 0 \geq 1 + k \cdot 2^P.$$

  Adding up one direction of this definition to (4), immediately yields that $\overline{z}_k^{T=0} \geq 1$, as desired.
- $C\restriction_\omega$ for each $C \in \mathcal{C}$.
  - If C is a clause in the original input, $C\restriction_\omega = C$ and this is trivial.
  - If C is a previously derived solution-improving constraint, also $C\restriction_\omega = C$ (since ω does not touch any variable in O.
  - If C is a previously derived constraint of the form $\overline{z}_{k'} \geq 1$ with $k' < k$, this can either be derived analogously to $\overline{z}_k\restriction_\omega \geq 1$ or directly from the fact that the definitions of z_k and z'_k immediately imply that $z_k = 0$ implies that $z_{k'} = 0$.
- $O\restriction_\omega \geq O$; this is obvious since the variables in O are unaltered by ω. ◀

▶ Remark 6. Proposition 5 assumes the existence of a constraint (4). It can be seen that this constraint is actually a (potentially weakened version of a) non-strict solution improving constraint $O \geq O\restriction_\alpha$ where α is a previously found solution. During the coarse convergence phase, this constraint can be obtained by weakening the solution-improving constraint.

At the end of the coarse convergence phase, also the unit clause $z_{k^*} \geq 1$ is derived. This requires no additional proof logging: this clause is obtained by running the SAT solver with the assumption that $z_{k^*} = 0$ and failing. Whenever this is the case; we know that $z_{k^*} \geq 1$ is internally derived by standard conflict analysis; hence this constraint is added to $\mathcal{D}$ without any additional effort.

A.2 Fine Convergence

As with the coarse convergence, the constraints derived during fine convergence that require a justification in the proof are the unit clauses added to the solver. Proving this relies again on redundance-based strengthening and a shadow circuit.

▶ **Proposition 7.** *Assume $\overline{z}_{k^*-1} \geq 1$ has been derived. Let s be any number and assume a complete shadow circuit for $T = s-1$ has been introduced. Furthermore assume the constraint*

$$O \geq s + (k^* - 1) \cdot 2^P \tag{5}$$

has been derived. The constraint $T \geq s-1$ can be derived using redundance-based strengthening with witness

$$\omega = T \mapsto s, Y \mapsto Y^{T=s-1}.$$

Proof. As in the proof of Proposition 4, this yields several proof obligations. The only non-trivial ones are

- Previously derived constraints of this form $T \geq s' - 1$, but they are trivially satisfied under ω since $s \geq s'$.
- The unit clause $\overline{z}_{k^*-1} \geq 1\restriction_\omega$. In other words we need to show that $\overline{z}_{k^*-1}^{T=s-1}$ holds. Recall that $z_{k^*-1}^{T=s-1}$ is defined by the reification

  $$\overline{z}_{k^*-1}^{T=s-1} \Leftrightarrow O - (s-1) \geq 1 + (k^* - 1) \cdot 2^P$$

  which simplifies to

  $$\overline{z}_{k^*-1}^{T=s-1} \Leftrightarrow O - s \geq (k^* - 1) \cdot 2^P.$$

  Now (5) tells us precisely that the right-hand side of this equivalence is satisfied, hence a straightforward cutting planes derivation indeed allows us to conclude that $\overline{z}_{k^*-1}^{T=s} \geq 1$. ◀

▶ Remark 8. Just like Proposition 4, also Proposition 7 does not make use of the model-improving constraint, but rather makes the assumption on O it uses explicit in (5). As before, this turns out to be useful when applying Proposition 7 in the context of stratification.

Proposition 7 will be applied when a solution α is found taking

$$s := O\restriction_\alpha - (k^* - 1) \cdot 2^P.$$

In this case, the solution-improving tells us that

$$O \geq O\restriction_\alpha + 1 = s + (k^* - 1) \cdot 2^P + 1,$$

and (5) is indeed satisfied. Unit clauses are derived if for a certain j, $s \geq 2^P - 2^j + 1$. In this case, the derived constraint $T \geq s - 1$ guarantees that $T \geq 2^P - 2^j$, i.e., that all dominant bits of T up to j must be equal to one. This follows using reverse unit propagation or a straightforward cutting planes derivation.

A.3 Conclusion of Optimality

When the very last call to the SAT solver is unsatisfiable, we need to derive a contradiction in the proof, to complete the proof that the previously best found solution is optimal. We proceed as follows. First, we introduce a fresh variable, let us call it p using the reification

$$p \Leftrightarrow O \geq o^* + 1. \tag{6}$$

Our goal will be to show that p is false, which then allows us to conclude that the objective can no longer be improved, meaning we have indeed proven optimality. Recall that at this point, we have s defined as $s := o^* - (k^* - 1) \cdot 2^P$. The crucial step in our proof is showing that without loss of generality T can be set equal to s. We proceed as follows.

▶ **Proposition 9.** *Assume $\overline{z}_{k^*-1} \geq 1$ and the definition of p have been derived. Furthermore suppose that a shadow circuit for $T = s$ has been introduced. Using redundance-based strengthening with witness*

$$\omega = T \mapsto s, Y \mapsto Y^{T=s}$$

we can derive the PB constraints representing

$$p \Rightarrow T = s, \tag{7}$$

i.e., in normalised form, the constraints

$$s \cdot \overline{p} + T \geq s, \text{ and} \tag{8}$$

$$(2^P - s - 2) \cdot \overline{p} + \sum_{j=0}^{P-1} 2^j \cdot \overline{T}_j \geq (2^P - 1) - s - 1. \tag{9}$$

Proof. The proof for the two constraints is similar. The only proof goal where they differ is showing that the constraint to-be-derived is satisfied under ω, but this is trivial since the witness sets T equal to s by construction.

For all the other proof goals, we can make use the negation of the constraint to be derived (the negation of (8) or of (9)). From this negation, we can directly derive $p \geq 1$. Adding this up to (one direction of (6) yields $O \geq o^* + 1$, i.e., that

$$O \geq s + (k^* - 1) \cdot 2^P + 1. \tag{10}$$

In other words, the conditions of 7 are satisfied. All the other proof obligations are the same as the ones in the proof of that proposition and hence, making use of (10), the proof proceeds identically to the proof of Proposition 7. ◀

In words, Proposition 9 tells us is that *if* the objective is strictly improving on the previously found best value, *then* we can set T equal to s without loss of generality. The SAT solver, however, has in its last call that yielded UNSAT already derived a clause telling us that at least one of the bits of T does not correspond to s. So we can now straightforwardly derive that $\overline{p} \geq 1$ and hence that $O \leq o^*$, which is what we needed for concluding optimality.

B Proof Logging of Additional Techniques Implemented in Pacose

We detail some of the additional search techniques implemented in and how we proof log them. As a minor point, we note for completeness that in addition to the gcd-based criterion described in Section 3.4, Pacose attempts to find more partitions of the objective during stratification via exhaustive search, as illustrated by the following example:

▶ **Example 10.** Consider the objective $O := 14x_1 + 9x_2 + 5x_3 + 2x_4 + 1x_5 + 1x_6$ and the partition $H = \{1, 2, 3\}$ and $L = \{4, 5, 6\}$. According to the gcd-based criterion from Section 3.4, this partition is not viable due to the gcd not aligning with any single divisor that groups the weights cohesively. However, this partition still validly separates the weights of x_1 to x_6 through an alternative method: Define L_C as the set containing all possible summed combinations of weights from L: $L_C := 5, 9, 14, 5+9, 5+14, 9+14, 5+9+14$. To validate this partitioning, ensure that the total weight W_L from L is at most the difference between any two sums in L_C. This ensures that L forms a consistent grouping, as there is no weight combination of L invalidating a prior result of solving H.

A more in-depth explanation together with a proof can be found in [51]. While certifying the exhaustive search remains interesting future work, we note that it did not result in additional partitions on any of the benchmarks in our evaluation, nor on the weighted instances of the 2019 and 2020 MaxSAT Evaluation.

We would like to mention that a naive approach to certify the exhaustive search would be to derive the desired constraint $O_H \geq O_H{\restriction_\alpha}$ from the weakened constraint $O_H \geq O{\restriction_\alpha} - W_L + 1$ using redundance-based strengthening with an empty witness. As $O_H{\restriction_\alpha}$ is the sum of a subset of the coefficients in O_H and the distance between any two sums is at least W_L, the negation $O_H < O_H{\restriction_\alpha}$ of the desired constraint can only be satisfied if the sum of true literals in O_H is at most $O_H{\restriction_\alpha} - W_L$. As $O{\restriction_\alpha} \geq O_H{\restriction_\alpha}$, the weakened constraint can only be satisfied if the sum of true literals in O_H is at least $O_H{\restriction_\alpha} - W_L + 1$. Hence, there exists no assignment to the variables in O_H for which both constraints are satisfied. To show this we can iterate through every possible assignment α of the variables in O_H and derive the clause excluding this assignment by reverse unit propagation. This step works, as reverse unit propagation for this clause assigns all variables in O_H, which will falsify either the negated constraint or the weakened constraint by the arguments above. Resolving all the clauses will result in a contradiction that proves that $O_H \geq O_H{\restriction_\alpha}$ is implied.

B.1 TrimMaxSAT

TrimMaxSAT [51] is a preprocessing technique applied before the main SIS algorithm in order to decrease the number of literals in the objective that need to be encoded by the DPW and to get a good initial value of the objective. TrimMaxSAT heuristically splits the variables in the objective into partitions and queries the SAT solver for a solution that assigns at least one of the literals in each partition to 1. If such an assignment is found, the objective variables set to 1 are removed from consideration and the number of partitions are decreased. If the partition size is 1 and the SAT solver reports UNSAT, all remaining literals are fixed to 0 for the rest of the search. In other words TrimMaxSAT aims to find objective literals whose negation is implied by the constraints in the formula and fix their value, thus conceptually decreasing the size of the objective under consideration and–as a consequence–also the size of the DPW encoding built over it.

In more detail, assume $\mathcal{L}$ contains the set of objective variables that have not been set to 1 in any solutions found so far during TrimMaxSAT. During an iteration of TrimMaxSAT, $\mathcal{L}$ is partitioned into m subsets $\mathcal{L}^i$ for $i = 1, \ldots, m$. A new variable r is introduced and the clauses $r \Rightarrow (\sum_{\ell \in \mathcal{L}^i} \ell \geq 1)$ for every $i = 1, \ldots, m$ are added to the SAT solver and the proof via redundance-based strengthening to the core set. The SAT solver is then queried under the assumption that r is true. If the result is SAT, the literals in $\mathcal{L}$ assigned to 1 in the obtained solution are removed from the set under consideration and the unit clause $\overline{r} \geq 1$ is added to the solver such that the SAT solver can remove the clauses of the form

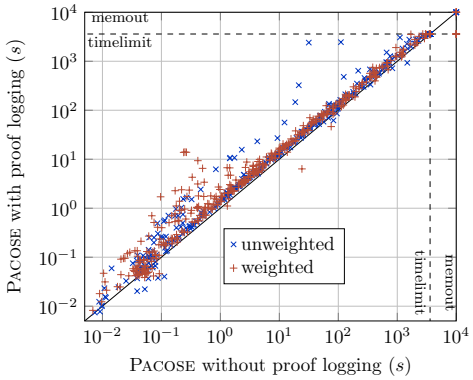

Figure 4 Proof logging overhead for PACOSE using the binary adder encoding.

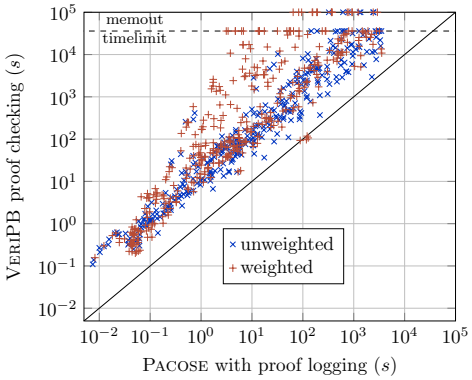

Figure 5 PACOSE vs. VERIPB running time using binary adder encoding.

$r \Rightarrow (\sum_{\ell \in \mathcal{L}^i} \ell \geq 1)$. This unit clause can be derived by redundance-based strengthening with witness $\omega = r \mapsto 0$. If, on the other hand, the result is UNSAT, the unit clause $\overline{r} \geq 1$ is added to the SAT solver and the SAT solver can simplify its clause database. This clause is derived by standard cutting planes reasoning in the conflict analysis by the SAT solver and is therefore added to the derived set in the proof. If in this case $m = 1$, we can also conclude that all literals $\ell \in \mathcal{L}$ are implied to be false. Hence, the solver learns the unit clauses $\overline{\ell} \geq 1$. In order to derive $\overline{\ell} \geq 1$ for each $\ell \in \mathcal{L}^i$, we first introduce the second part of the reification $r \Leftarrow (\sum_{\ell \in \mathcal{L}^i} \ell \geq 1)$ using the redundance rule with witness $r \mapsto 1$ and then use cutting planes reasoning to derive that since r is false, all literals in $\mathcal{L}^i$ must be false. Interestingly, thanks to the use of strengthening-to-core, the unit clause $\overline{r} \geq 1$ derived earlier does not interfere with the derivation of the second direction of the reification.

B.2 Hardening

Hardening refers to the addition of the unit clause l_i for an objective literal l_i if the currently best known solution o^* is larger than the sum of all weights in O excluding w_i. In the proof, the unit clause l_i can be derived easily from the solution-improving constraint and the objective reformulation rule can be used to replace l_i by the constant w_i in the objective.

C Additional Experimental Evaluation

In this appendix, we present some additional experimental analysis with data and plots to give some further insights into proof logging for PACOSE. In Section C.1, we present results for the binary adder encoding that is also used in PACOSE and how detail how well proof logging performs for PACOSE when it heuristically selects the encoding. We present data for an additional approach that uses assumptions instead of unit clauses for fixing variables in the coarse convergence in Section C.2. To better understand the proof logging overhead in PACOSE, we have a deeper look at some additional data for the proof logging process in Section C.3.

C.1 Binary Adder Encoding and Encoding Selection Heuristic

PACOSE also uses the binary adder encoding [62] instead of the DPW encoding. A comparison between these two encodings is beyond the scope of this paper, but as we implemented proof logging for both encodings, we can also have a look at the data for the binary adder

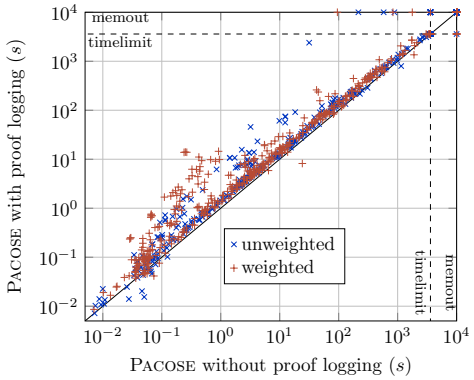

Figure 6 Proof logging overhead for PACOSE using heuristic encoding selection.

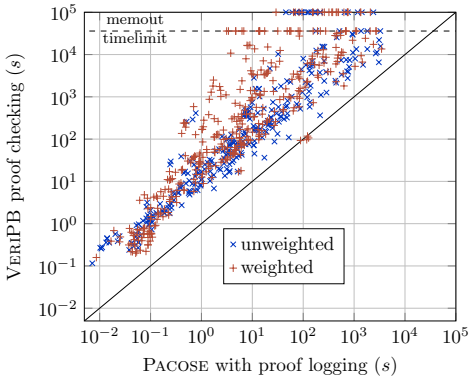

Figure 7 PACOSE vs. VERIPB running time using heuristic encoding selection.

encoding. A comparison of solving with and without proof logging for this encoding can be found in Figure 4. With proof logging for the binary adder encoding 722 instances could be solved within the resource limits, which are 6 fewer instances than without proof logging. This also demonstrates that the heuristic for selecting the encoding works, as the number of solved instances for the heuristic is bigger than for any of the two encodings on their own. In the mean, PACOSE with proof logging is 1.63× slower than without proof logging. This overhead is smaller than for the DPW encoding, which lead to the conclusion that more work is required to certify the DPW encoding compared to the binary adder encoding.

Out of the 722 instances that were solved with the binary adder encoding, 658 instances were successfully checked by VERIPB within the resource limits. In Figure 5, the running time of PACOSE is compared to that of VERIPB. In the mean, VERIPB is 21.1× slower than PACOSE for solving the instance with proof logging, which is similar to the DPW encoding. This could mean that the bottleneck for checking the proofs is the implementation of the checker.

Using the default settings, PACOSE heuristically selects between the DPW and binary adder encoding. A plot comparing PACOSE with and without proof logging in the default settings in Figure 6 and a plot comparing PACOSE with proof logging with VERIPB for checking the proof in Figure 7. With this heuristic activated, 698 instances are solved within the resource limits with proof logging enabled and 707 instances without. PACOSE with proof logging is 1.83× slower in the mean than PACOSE without proof logging. Checking the proof with VERIPB is 21.8× slower than running PACOSE with proof logging in the mean.

C.2 Coarse Convergence with Assumptions Instead of Unit Clauses

An alternative approach for representing the information that output variables of the DPW encoding are fixed to a value in the coarse convergence is to use additional assumptions for the SAT solver instead of unit clauses. As we need a shadow circuit to derive each unit clause, we could reduce the number of shadow circuits by using assumptions. The idea is that we add the variable fixing to the assumptions for all future calls to the SAT solver. This approach is supported in PACOSE, and we ran additional experiments using this approach.

The following data always use assumptions instead of unit clauses for fixing variables. In Figure 8, PACOSE with proof logging is compared to PACOSE without proof logging. Using assumptions PACOSE with proof logging could solve 666 instances, which is 10 fewer instances than without proof logging. PACOSE with proof logging is 1.81× slower than without proof

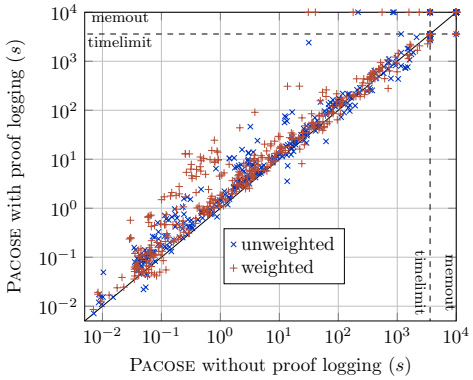

Figure 8 Proof logging overhead for PACOSE using DPW encoding and assumptions.

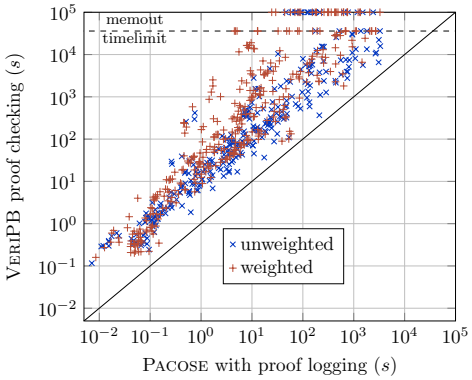

Figure 9 PACOSE vs. VERIPB running time using DPW encoding and assumptions.

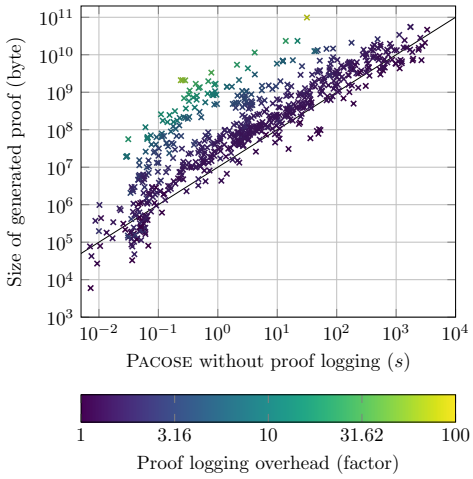

Figure 10 Solving time vs. proof size vs. solving overhead for proof logging for the DPW encoding.

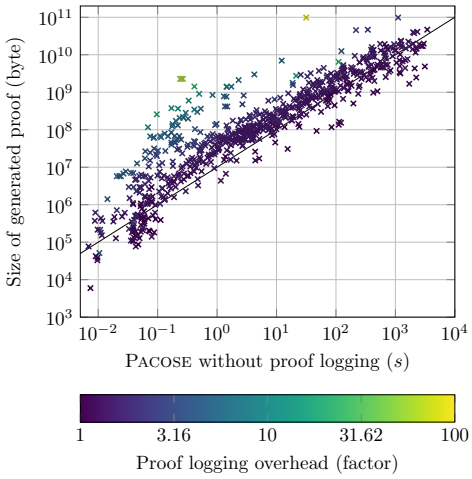

Figure 11 Solving time vs. proof size vs. solving overhead for proof logging for the binary adder encoding.

logging in the mean. This is very similar to PACOSE with the DPW encoding where the variables are fixed by unit clauses and introducing shadow circuits. In the mean, the proof checking is 22.2× slower than solving the instance with proof logging.

It can be concluded that this alternative approach of fixing variables by adding assumptions is about as good as doing the fixing by unit clauses. Hence, it could be that introducing additional shadow circuits for deriving the unit clauses does not slow down the solving a lot, or it is a coincidence that the performance gains are countered by the additional work required for keeping track of the assumptions.

C.3 Proof Logging Overhead Analysis

To get a better understanding of the 1.93× slowdown of PACOSE with proof logging compared to without proof logging, we investigate different causes for the extra running time with proof logging. The idea for doing so is to get insights into how to improve the running time of the solvers.

The expectation is that the proof size scales linearly with the running time of the solver. It would be interesting to look into the instances where this is not the case and if there is a correlation with the solving overhead. We can illustrate this by plotting the solving time against the proof size and colour the marks depending on the overhead as it is done in Figure 10 for the DPW encoding and in Figure 11 for the binary adder encoding. We added a diagonal line representing linear scaling of proof size with running time for better orientation, which is not related to the data at all. It can be seen that for the instances that have a proof size that is significantly bigger than expected, the overhead also seems to increase similarly. To confirm this observation, we compute the correlation of the proof logging overhead and the proof size divided by the solving time. For the DPW encoding we have a correlation of 0.92 and for the binary adder encoding we have a correlation of 0.88, which shows that the two parameters are highly correlated. This mean that the slowdown is due to proof being larger than expected for some instances.

We can conclude with some ideas to improve the performance of proof logging in Pacose. First, the performance can be improved by engineering better data structures to handle the proof logging to increase the speed for writing the proof. This idea only works if we have not reached the maximum persistent disk write speed, which is not the case for our experiments. Second, the proof could be done in a smarter way to reduce the size of the proof, where slow parts of the proof logging could be identified by profiling. Considering that we also have a $1.63\times$ slowdown for the binary adder encoding, the slowdown is not purely caused by the shadow circuits, as they are not used for this encoding.