QMaxSATpb: A Certified MaxSAT Solver

Dieter Vandesande[0000−0002−8150−3202], Wolf De Wulf[0000−0002−0219−3120], and Bart Bogaerts[0000−0003−3460−4251]

Vrije Universiteit Brussel, Artificial Intelligence Laboratory
Pleinlaan 9, 1050 Brussel, Belgium
`dieter.vandesande@vub.be, wolf.de.wulf@vub.be, bart.bogaerts@vub.be`

Abstract. While certification has been successful in the context of satisfiablity solving, with most state-of-the-art solvers now able to provide proofs of unsatisfiability, in maximum satisfiability, such techniques are not yet widespread. In this paper, we present QMaxSATpb, an extension of QMaxSAT that can produce proofs of optimality in the VeriPB proof format, which itself builds on the well-known cutting planes proof system. Our experiments demonstrate that proof logging is possible without much overhead.

Keywords: Boolean satisfiability, maximum satisfiability, optimization, certification, proofs

1 Introduction

As the area of combinatorial search and optimization matures, we observe a strong increase in applications, as well as in highly optimized solving technology. Since some of these applications involve high-value and life-affecting decision-making processes (e.g., verifying software that drives our transportation infrastructure [44], or matching donors and recipients for Kidney transplants [34]), it is of utmost importance that the answers produced by the solvers be completely reliable. Unfortunately, the reality is different: the constant need for more efficient and advanced algorithms forms an excellent breeding ground for bugs, resulting in numerous reports of solvers outputting faulty answers [9, 11, 1, 20].

There are multiple ways to deal with this issue. One possibility is to formally verify correctness of the solvers. While there have lately been promising advances in this direction (e.g., [18]), formally verifying advanced reasoning methods used in modern-day solvers turns out to be challenging. Another approach is *certification* or *proof logging*. The idea here is that a solver should not just produce an answer, but also an efficiently verifiable *certificate* or *proof* showing that the answer is correct [2, 38]. This certificate can then subsequently be verified by an independent tool (often referred to as a *verifier* or *proof checker*) of much lesser complexity. In Boolean satisfiability (SAT) [36], this approach has been successfully applied, with numerous proof formats and verifiers, including even some formally verified verifiers [26, 6, 27, 28, 43, 14, 13]. Moreover, for several years, it has been a requirement of the (main track of the) SAT competition that solvers provide certificates of their answers.

In the domain of Maximum Satisfiabilty (MaxSAT), the optimization variant of SAT, certification has not yet had its breakthrough moment. A couple of proof systems and tools have been developed to certify MaxSAT solutions [8, 40, 33, 39]. However, none of them can truly be called a general-purpose proof system for MaxSAT, either because they are built to certify only specific MaxSAT algorithms, or because of a limited expressivity, such as for instance a lack of support for rules introducing new variables, which is common in several pre- and inprocessing techniques. For this reason, we expect that richer proof systems will be needed to handle the full range of MaxSAT solving techniques.

One promising proof format to fill this gap is VeriPB [17, 25, 24, 22], a proof format for pseudo-Boolean satisfiability, that was recently extended to *pseudo-Boolean optimization* [7], which generalizes MaxSAT. VeriPB builds on top of the *cutting planes* proof system [12], and extends it with a generalization of the *Resolution Asymmetric Tautology* (RAT) rule that lies at the basis of the most successful proof formats for SAT. VeriPB naturally facilitates proof logging for advanced techniques such as XOR and cardinality reasoning [25] and symmetry breaking [7] that have been largely out of reach of other proof formats for SAT.

In this paper, we present QMaxSATpb, an extension of the MaxSAT solver QMaxSAT with capabilities to output proofs in the VeriPB format. In brief, QMaxSAT works as follows. First, the input CNF is augmented by a *totalizer circuit* [5]. The purpose of this circuit is to count the number of falsified clauses. Next, it iteratively calls MiniSAT [16], asking for a solution in which strictly more clauses are satisfied, until no better solution can be found. The biggest challenge to obtain VeriPB-compatible proofs, was to deduce a native pseudo-Boolean encoding of the variables involved in the totalizer circuit. Once that is in place, the rest of the algorithm can easily be forced to output VeriPB-compatible proofs. In fact, since the VeriPB format generalizes the DRAT proof system, which is the most common proof system for SAT, the MiniSAT oracle could be replaced by any SAT solver that supports DRAT proofs (which in practice means, most state-of-the-art SAT solvers).

We experimentally validate our solver on benchmarks from MaxSAT challenges. Our experiments show that proof logging is generally possible with minimal overhead. While most proofs can be verified by VeriPB, we do notice that verification still takes more time than solving, and we did find some cases where VeriPB can not verify a proof within reasonable limits, thereby suggesting that the proofs generated by our new tool form an interesting new testing ground for efficiency improvements to the proof verifier.

The rest of this paper is structured as follows. In Section 2 we introduce the necessary preliminaries. Section 3 describes how we extended QMaxSAT with proof logging and Section 4 contains our experiments. We conclude in Section 5.

2 Preliminaries

Propositional logic As usual, a *Boolean variable* x ranges over the Boolean values 0 (*false*) and 1 (*true*). A *literal* is a variable x or its negation $\overline{x}$. A *clause* $C =$

$a_1 \vee \cdots \vee a_k$ is a disjunction of literals. A *formula* (in Conjunctive Normal Form – CNF) is a conjunction of clauses. The *empty clause* is denoted $\bot$. An *assignment* α is a partial function from the set of variables to 0 or 1; it is extended to literals by $\alpha(\overline{x}) = 1 - \alpha(x)$. The assignment α *satisfies* a clause C if it assigns at least one of C's literals to 1; it satisfies a formula F, if it satisfies all the clauses of F. In that case, we also call α a *model* of F. A formula F is *satisfiable* if it has a model, and *unsatisfiable* otherwise. The *Boolean Satisfiability problem* (SAT) consists of deciding whether a formula F is satisfiable. A *partial* MaxSAT-instance[1] is a tuple (F, S) with F the hard clauses and S the soft clauses. The *Maximum Boolean Satisfiability problem* (MaxSAT) consists of finding an assignment that satisfies all clauses of F and as many clauses of S as possible. Without loss of generality, we can assume that $S = \{\overline{x_1}, \ldots, \overline{x_n}\}$; this can always be enforced by introducing so-called *relaxation variables.*

QMaxSAT QMaxSAT [32] is an *iterative partial MaxSAT solver*. Such solvers work by repeated calls to a SAT oracle, each time requesting a better solution than the best one found so far. To express the constraint that the next solution should be better than the previous one (i.e., that fewer literals in S should be falsified), QMaxSAT uses an encoding of *cardinality constraints* in propositional logic, namely the *totalizer circuit encoding* [5]. We will review the full encoding in Section 3, but in order to understand the algorithm what matters is that, with $S = \{\overline{x_1}, \ldots, \overline{x_n}\}$, it introduces new variables (among those new variables are the "counting" variables $v_1, \ldots, v_n$) and introduces a formula G over the original and the new variables such that: each model α of F can uniquely be extended to a model α' of $F \cup G$; moreover, α' is such that $\alpha'(v_j)$ is true if and only if at least j of the relaxation variables x_i are true.

After adding this encoding, QMaxSAT searches for a model of $F \cup G$ (using an oracle call to a SAT solver). If a model α is found, the clauses $\overline{v_j}$ are added for all $j \geq UB$, with UB (the "upper bound") equal to the number of x_i that is true in α. These added clauses express that strictly fewer of the relaxation variables x_i are true (and hence, strictly more soft constraints are satisfied), and a new model is sought. This process repeats until no more solutions can be found. Pseudocode of the QMaxSAT algorithm can be found in Algorithm 1.

The VeriPB proof system We now review the rules of the VeriPB proof system we use for certification of QMaxSAT; we refer the reader to Bogaerts et al. [7] for an exposition of the full proof system. A pseudo-Boolean (PB) constraint C is a linear inequality of the form $\sum_i a_i l_i \geq A$ where a_i and A are integers, l_i are literals. We call $\sum_i a_i l_i$ a *linear term*; the value of this term in a total assignment α is $\sum_i a_i \alpha(l_i)$. The constraint C is true in α if $\sum_i a_i \alpha(l_i) \geq A$. Without loss of generality it can be assumed that PB constraints are *normalized*: that all literals l_i are over distinct variables and the coefficients a_i and A are non-negative.

[1]Since QMaxSATpb is based on QMaxSAT version 0.1 [31], which does not support weights, we do not discuss weighted (partial) MaxSAT here. Later versions of QMaxSAT do support weights, using other encodings of PB constraints in CNF.

Algorithm 1: QMaxSAT

```
1  input: a set of hard clauses $F$, a set of soft clauses $S = \{\overline{x_1}, \dots, \overline{x_n}\}$, a
     SAT-solver Solver
2  $\alpha \leftarrow$ Solver.solve($F$)
3  if $\alpha =$ 'UNSAT' then
4      return 'UNSAT'
5  $G, Y \leftarrow$ generateTotalizerClauses($R$) /* with $G$ the totalizer clauses
       and $v_1, \dots v_n \in Y$ the output variables meaning that at least $i$
       variables $x_i$ are satisfied.                                          */
6  $F \leftarrow F \cup G$
7  while true
8      $UB \leftarrow \#\{i \mid \alpha(x_i) = 1\}$
9      $F \leftarrow F \cup \{\overline{v_j} \mid j \geqslant UB\}$
10     $\alpha \leftarrow$ Solver.solve($F$)
11     if $\alpha =$ 'UNSAT' then
12         return $UB$
```

The negation of C is a PB constraint as well, namely $\overline{C} = \sum_i -a_i l_i \geq -A + 1$. A PB formula or theory is a conjunction of PB constraints. Clearly, a clause $l_1 \vee \dots \vee l_k$ is equivalent to the PB constraint $l_1 + \dots + l_k \geq 1$. Hence, SAT formulas (in CNF) are special cases of PB formulas. An instance of a *pseudo-Boolean optimization problem* is a tuple (F, f) with F a PB formula and f a linear term to be minimized, referred to as the *objective function*. A solution for (F, f) is a model of F, for which there does not exist another model of F with a smaller objective function value. A partial MaxSAT problem (F, S) can also be seen as a pseudo-Boolean optimization problem $(F, \sum_{s \in S} s)$.[2]

For an instance (F, f), the VeriPB proof system keeps track of a *proof configuration* $(\mathcal{C}, v^*)$ with $\mathcal{C}$ a set of constraints (initialized as F) and v^* an integer or ∞ representing the best value for f found so far (initialized as ∞). It allows updating this configuration $(\mathcal{C}, v^*)$ using the cutting planes proof system [12]:

Literal Axioms: For any literal, we can add $l_i \geq 0$ to $\mathcal{C}$.
Linear Combination: Given two PB constraints C_1 and C_2 in $\mathcal{C}$, we can add a linear combination of C_1 and C_2 to $\mathcal{C}$.
Division: Given the normalized PB constraint $\sum_i a_i l_i \geq A$ in $\mathcal{C}$ and an integer c, we can add the constraint $\sum_i \lceil a_i/c \rceil l_i \geq \lceil A/c \rceil$ to $\mathcal{C}$.
Saturation: Given the normalized PB constraint $\sum_i a_i l_i \geq A$ in $\mathcal{C}$, we can add $\sum_i b_i l_i \geq A$ with $b_i = \min(a_i, A)$ for each i, to $\mathcal{C}$.

Additionally, VeriPB allows for rules for dealing with optimization statements:

Objective Bound Update: Given a model α of F, we can update v^* to $\alpha(f)$.
Objective Improvement: We can always add the constraint $f < v^*$ to $\mathcal{C}$.

[2] Here, we use the fact that we can assume that $S = \{\overline{x_1}, \dots, \overline{x_n}\}$, but our proof logging and verification also work with an arbitrary set of soft clauses.

The first of these rules updates the objective value, given a solution; the second represents that once a solution is found, we search for strictly better solutions. VeriPB also has a rule for **deleting** constraints (in a way that guarantees that no better-than-optimal values can be found). Finally, VeriPB allows deriving non-implied constraints with a generalization of the RAT rule (which is common in proof systems for SAT). This rule makes use of a *substitution*, which maps every variable to 0, 1, or a literal. Applying a substitution ρ on a constraint C results in the constraint $C\upharpoonright_\rho$ obtained from C by replacing each x by $\rho(x)$.

Redundance-based strengthening: In case $\mathcal{C} \wedge \overline{C} \vdash f\upharpoonright_\rho \leq f \wedge (\mathcal{C} \cup C)\upharpoonright_\rho$, we can add C to $\mathcal{C}$.

Intuitively, this rule can be used to show that ρ, when applied to assignments instead of formulas, maps any solution of $\mathcal{C}$ that does not satisfy C to a solution of $\mathcal{C}$ that does satisfy C and that has an objective value that is at least as good. Importantly, in many cases, it is not required to show precisely why a constraint can be derived, but the verifier can figure it out itself (by means of so-called *reverse unit propagation [26]*).

In case the proof system ends in a state $(\mathcal{C}, v^*)$, with $\bot \in \mathcal{C}$, we know that v^* is the value of the optimal solution of (F, f). If $v^* = \infty$, then F is unsatisfiable.

3 QMaxSATpb

We now explain how we extended QMaxSAT with proof logging capabilities, resulting in QMaxSATpb. The overall idea is that whenever the upper bound is updated (Line 8 in Algorithm 1), we have a model of F and can use the bound update rule to update v^* accordingly. In the rest of the algorithm, we make sure that every clause that is added, can be derived in the VeriPB proof system. There are three different places where new clauses are derived:

- Clauses derived by the SAT solver (Lines 2 and 10).
- Clauses representing the totalizer encoding (Line 5).
- Clauses for strengthening the theory to match the new upper bound (Line 9).

We will discuss them in this order. All the **clauses derived by the SAT solver** used by QMaxSAT (which is MiniSAT), are in fact reverse unit propagation (RUP) clauses, meaning that they are implied by the cutting planes proof system. VeriPB can check itself that they are indeed implied; simply claiming they are RUP suffices to yield a valid proof. It is important to note here that our approach does not hinge on all clauses being RUP. If the solver is replaced by any other modern SAT solver with proof logging capabilities, only minor syntactic modifications are needed to make it VeriPB-compatible. Indeed, as mentioned before, redundance-based strengthening generalizes the well-known RAT rule, and moreover, VeriPB can additionally handle symmetry breaking, cardinality reasoning and XOR reasoning [25, 7].

Let us now turn our attention to proofs for the **totalizer encoding**. First, we observe that cardinality definitions can easily be derived by redundance-based strengthening.

Proposition 1. *Let $(\mathcal{C}, v^*)$ be a proof configuration, X a set of variables, $j \leq |X| + 1$ an integer and v_j^X a variable not occurring in $\mathcal{C}$. The following two constraints can be derived by redundance-based-strengthening:*

$$P_{1,j}^X := \sum_{x \in X} \overline{x} + (|X| - j + 1) \cdot v_j^X \geq |X| - j + 1$$

$$P_{2,j}^X := \sum_{x \in X} x + j \cdot \overline{v_j^X} \geq j$$

The first of these constraints expresses that v_j^X is true if at least j of the variables in X are true. The second expresses that v_j^X is false if it is not the case that at least j of the variables in X are true.

Proof. The variable v_j^X is a fresh variable; the constraints enforce it to be true if and only if $\sum_{x \in X} \overline{x} \geq j$. Gocht and Nordström [25] have shown how these defining constraints can be derived using redundance-based strengthening.[3] We include a full proof here to make the paper self-contained. To prove that a constraint $P_{i,j}^X$ can be derived by redundance-based strengthening, we need to show that a substitution ρ exists such that $\mathcal{C} \wedge \overline{P_{i,j}^X} \vdash f\!\upharpoonright_\rho \leq f \wedge (\mathcal{C} \cup P_{i,j}^X)\!\upharpoonright_\rho$.

We start by proving this for $P_{1,j}^X$. Let ρ_1 be the substitution that maps v_j^X to 1 (and every other variable to itself). Since v_j^X is not used in $\mathcal{C}$ nor f, $\mathcal{C}$ clearly entails $f\!\upharpoonright_\rho \leq f \wedge \mathcal{C}\!\upharpoonright_{\rho_1}$. Since ρ_1 maps v_j^X to 1, $(\mathcal{C} \cup P_{i,j}^X)\!\upharpoonright_\rho$ is trivially true.

We now have to prove that we can derive $P_{2,j}^X$ from $\mathcal{C} \cup P_{1,j}^X$, since $P_{1,j}^X$ was derived before $P_{2,j}^X$. Let ρ_2 be the substitution that maps v_j^X to 0. Since ρ_2 does not change $\mathcal{C}$ nor f and since $P_{2,j}^X\!\upharpoonright_{\rho_2}$ is trivially satisfied, it suffices to show that $P_{1,j}^X \wedge \overline{P_{2,j}^X} \vdash P_{1,j}^X\!\upharpoonright_\rho$. Now, whenever $\overline{P_{2,j}^X}$ holds, it must be that v_j^X is true and $\sum_{x \in X} x < j$ and therefore $\sum_{x \in X} \overline{x} \geq |X| - j + 1$. I.e., $P_{1,j}^X$ must then be satisfied independently of v_j^X and hence also $P_{1,j}^X\!\upharpoonright_\rho$ is entailed. □

Now, the totalizer encoding does not consist of the constraints $P_{i,j}^X$ above but of clauses encoding the same information. We will show that these clauses can be derived by cutting planes derivations from the PB constraints $P_{i,j}^X$. As such, the way we implemented proof logging in QMaxSATpb consists of first deriving the constraints in PB form, and subsequently extracting the clauses used in the totalizer encoding. Also when adding the strengthening constraints below, will we make use of the PB encoding of these variables.

Definition 1 (Totalizer encoding [5]). *Let X be a set of variables and T a binary tree of which the leaves are the variables in X. For every node η of T, let $vars(\eta)$ denote the leaves of T that are descendants of η. For each internal node η, the totalizer encoding introduces variables $v_j^{vars(\eta)}$ with $0 \leq j \leq |vars(n)| + 1$ with intended meaning that $v_j^{vars(\eta)}$ holds if at least j variables of $vars(\eta)$ are*

[3] There, this rule was called *subsitution redundancy*.

true. For each leaf node x, we write $v_0^x = 1$, $v_1^x = x$, and $v_2^x = 1$. For each internal node η with children η_1 and η_2, the encoding consists of the clauses

$$C_1^\eta(\alpha,\beta,\sigma) = \overline{v_\alpha^{E_l}} \vee \overline{v_\beta^{E_k}} \vee v_\sigma^{E_j} \tag{1}$$

$$C_2^\eta(\alpha,\beta,\sigma) = v_{\alpha+1}^{E_l} \vee v_{\beta+1}^{E_k} \vee \overline{v_{\sigma+1}^{E_j}} \tag{2}$$

for all combinations of $0 \leq \alpha \leq |vars(\eta_1)|$, $0 \leq \beta \leq |vars(\eta_2)|$, $0 \leq \sigma \leq |vars(\eta)|$, with $\alpha + \beta = \sigma$.

This encoding can be simplified by replacing v_0^η by 1 and $v_{|\eta|+1}^\eta$ by 0 for each η. QMaxSATpb adds such a totalizer encoding for the set of relaxation variables $R := \{x \mid \overline{x} \in S\}$. The next theorem shows that it can indeed be derived in the cutting planes proof system.

Theorem 1. *Let X be a set of variables and T a binary tree of which the leaves are the variables in X. Let η be an internal node of T with children η_1 and η_2. The totalizer encoding clauses $C_1^\eta(\alpha,\beta,\sigma)$ and $C_2^\eta(\alpha,\beta,\sigma)$ can be derived by a cutting planes derivation from the PB constraints*

$$P_{i,j}^\eta \cup P_{i,j}^{\eta_1} \cup P_{i,j}^{\eta_2}$$

Proof. Let η be an internal node with children η_1 and η_2 and define $X_1 = vars(\eta_1)$ and $X_2 = vars(\eta_2)$. Assume $\alpha+\beta = \sigma$ with $0 \leq \alpha \leq |X_1|$, $0 \leq \beta \leq |X_2|$, $0 \leq \sigma \leq |X|$.

Summation of $P_{2,\alpha}^{X_1}$, $P_{2,\beta}^{X_2}$ and $P_{1,\sigma}^{X}$ results in:

$$\left(\sum_{x_i \in X_1} x_i\right) + \alpha\overline{v_\alpha^{X_1}} + \left(\sum_{x_j \in X_2} x_j\right) + \beta\overline{v_\beta^{X_2}} + \left(\sum_{x \in X} \overline{x}\right) + (|X| - \sigma + 1)v_\sigma^X$$
$$\geq \alpha + \beta + |X| - \sigma + 1$$

We can rewrite the left hand side because $X_1 \cap X_2 = \emptyset$ and $X_1 \cup X_2 = X$. The right hand side can be rewritten because $\alpha + \beta - \sigma = 0$. This results in:

$$\left(\sum_{x \in X} \overline{x}\right) + \left(\sum_{x \in X} x\right) + \alpha\overline{v_\alpha^{X_1}} + \beta\overline{v_\beta^{X_2}} + (|X| - \sigma + 1)v_\sigma^X \geq |X| + 1$$

Because of the opposite signs, the first two terms simplify to $|X|$; after a saturation step, this results in:

$$\overline{v_\alpha^{X_1}} + \overline{v_\beta^{X_2}} + v_\sigma^X \geq 1$$

This is exactly $C_1^\eta(\alpha,\beta,\sigma)$ from the totalizer encoding.

A similar pattern is followed in the proof for the $C_2^\eta(\alpha,\beta,\sigma)$ constraints. First, summation of $P^X_{2,\sigma+1}$, $P^{X_1}_{1,\alpha+1}$ and $P^{X_2}_{1,\beta+1}$ gives:

$$\left(\sum_{x\in X} x\right) + (\sigma+1)\overline{v^X_{\sigma+1}} + \left(\sum_{x_i\in X_1} \overline{x_i}\right) + (|X_1|-\alpha)v^{X_1}_{\alpha+1}$$
$$+ \left(\sum_{x_j\in X_2} \overline{x_j}\right) + (|X_2|-\beta)v^{X_2}_{\beta+1} \geq \sigma+1+|X_1|-\alpha+|X_2|-\beta$$

Because $X_1\cap X_2=\emptyset$ and $X_1\cup X_2=X$, the left hand side can be rewritten. The right hand side can be rewritten because $|X_1|+|X_2|=|X|$ and $\sigma-\alpha-\beta=0$. This results in:

$$\sum_{x\in X} x + \sum_{x\in X}\overline{x} + (\sigma+1)\overline{v^X_{\sigma+1}} + (|X_1|-\alpha)v^{X_1}_{\alpha+1} + (|X_2|-\beta)v^{X_2}_{\beta+1} \geq 1+|X|$$

Because of the opposite signs in the first two terms, this is equivalent to:

$$(\sigma+1)\overline{v^X_{\sigma+1}} + (|X_1|-\alpha)v^{X_1}_{\alpha+1} + (|X_2|-\beta)v^{X_2}_{\beta+1} \geq 1$$

After a saturation step, this becomes equivalent to the $C_2^\eta(\alpha,\beta,\sigma)$ constraints from the totalizer encoding. □

The next theorem shows that the unit clauses to constrain the next solutions to better ones can be derived in the VeriPB proof system as well.

Theorem 2. *Let $(\mathcal{C}, v^*)$ be a proof configuration and $Y=\{v^R_1,...v^R_{|R|}\}$ be the set of counting variables on the set $R:=\{x \mid \overline{x}\in S\}$. For any $j\geq v^*$, the unit clause $\overline{v^R_j}$ can be derived if $\mathcal{C}$ contains the pseudo-Boolean constraint $P^R_{2,j}$.*

Proof. Let $n\geq v^*$. Because of the Objective Improvement rule, VeriPB can derive the constraint:

$$\sum_{x_i\in R}\overline{x_i} \geq |R|-v^*+1$$

Summation with $P^R_{2,j}$ results in:

$$\left(\sum_{x_i\in R}\overline{x_i}\right) + \left(\sum_{x_i\in R} x_i\right) + n\overline{v^S_j} \geq |R|-v^*+1+n$$

Because of the opposite signs of the first two terms, they can be dropped, resulting in:

$$n\overline{v^S_j} \geq n-v^*+1$$

Division by $n-v^*+1$ followed by a saturation step results in:

$$\overline{v^S_j} \geq 1$$

These are exactly the unit clauses added by QMaxSAT. □

Therefore, given an upper bound UB, QMaxSAT can derive the constraints $\overline{v^Y_j}$ for all $j\geq UB$ and call MiniSAT to find a better solution.

4 Implementation and Experiments

In order to test our work, we extend QMaxSAT (version 0.1 [31]) with proof logging as described above. The extensions amount to the following:

– All learned clauses are written to the proof file using VeriPB's RUP notation.
– All performed clause deletions (internally in the SAT solver) are written to the proof file.
– Each time a new objective value is found, it is written to the proof file using VeriPB's objective bound update notation.
– Before QMaxSAT builds its totalizer circuit, redundance-based strengthening is used to derive a pseudo-Boolean encoding of the cardinality constraints.
– The original procedure for generating the totalizer encoding now also writes the cutting planes derivations for these clauses to the proof file.
– The unit clauses that constrain the objective function and their cutting planes derivations are written to the proof file as well.

The QMaxSAT patch as well as the source code and all necessary scripts to run the experiments, are available on GitHub [41, 42].

We experimentally validate QMaxSATpb using benchmarks from the 2021 MaxSAT evaluation [10]. All benchmarks were ran on the VUB Hydra cluster. Each solver call was assigned a single core on a 20-core INTEL Xeon Gold 6148 (skylake) processor with a time limit of 60 minutes and a memory limit of 32GB, matching the resource limits used in the evaluation. Preliminary tests suggested that verification using VeriPB (commit **0e61617**) requires substantially more memory and time than solving. For this reason, we assigned veripb 600 minutes of computation time and 64GB of memory.

For all instances that QMaxSAT could solve within the set limits, we plot the time taken to solve the instance using QMaxSAT and using QMaxSATpb in Figure 1. We observe that for the majority of the instances, the overhead induced by proof logging is negligible. In fact, there are only four instances that QMaxSATpb could not solve within the limits when QMaxSAT could.

When comparing the time needed to solve instances and the time needed to verify the produced proofs (see Figure 2), we notice that VeriPB typically needs more time to validate a proof than QMaxSATpb needs to solve the corresponding instance. Moreover, VeriPB could not verify 10.2% of the solved instances within the time limits and 2.4% within the memory limits. This suggests that VeriPB might benefit from performance and memory optimisation.

5 Conclusion

In this paper, we presented QMaxSATpb, an extension to QMaxSAT with proofs of optimality of the computed solutions. To verify solutions we used the VeriPB proof system, which is based on the cutting planes proof system. We experimentally validated our approach and found that proof logging itself requires

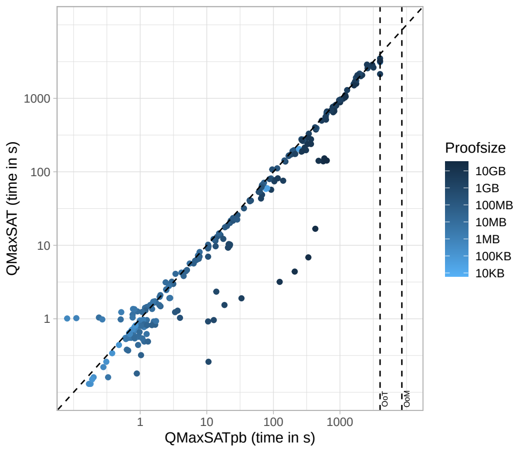

Fig. 1: Performance overhead induced by proof logging: for each instance, this plot contains a comparison between the time needed to solve it with and without proof logging.

minimal overhead. Furthermore, we find that while verification can generally happen in reasonable time, for some instances with relatively large proofs, the verifier struggles to finish in time.

Together with ongoing research on certified translations of pseudo-Boolean constraints into CNF clauses [21], this work takes an important step towards certification of state-of-the-art MaxSAT solvers. Indeed, one important contribution of our paper is to show very precisely how totalizer encodings can be added in a certified way such that the semantics of each newly introduced variable is explicitly expressed as a pseudo-Boolean formula. Such totalizer encodings lie at the heart of so-called *core-guided* MaxSAT algorithms (e.g., [4, 29, 30, 37]). In future work we want to extend this class of solvers with proof-logging capabilities as well.

While our paper focusses on SAT and MaxSAT, the VeriPB proof system has also been used to provide proof logging for certifying solvers in richer domains [23]. We believe it can also play an important role for the field of answer set programming [35], where all modern solvers [19, 15, 3] support pseudo-Boolean constraints, and are based on search and optimization algorithms that are similar to those from SAT and MaxSAT.

Acknowledgement. The authors are grateful to Jeremias Berg, Stephan Gocht, Ciaran McCreesh, and Jakob Nordström for fruitful discussions on proof logging and feedback on previous versions of this manuscript. This work was partially supported by Fonds Wetenschappelijk Onderzoek – Vlaanderen (FWO) (project G070521N); computational resources and services were provided by the VSC (Flemish Supercomputer Center), funded by FWO and the Flemish Government.

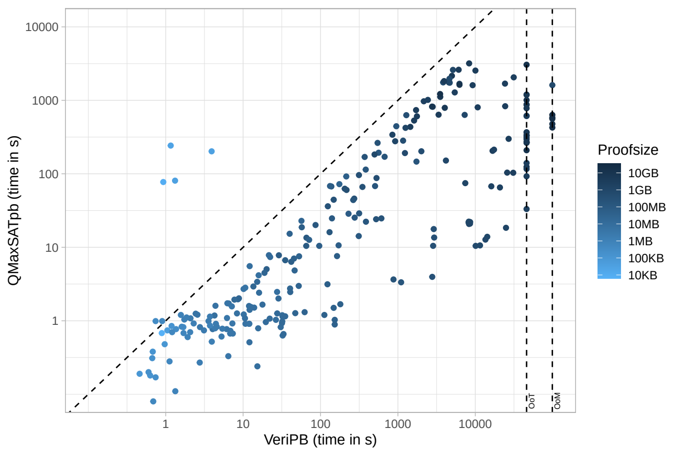

Fig. 2: Performance of proof verification: for each instance the time needed to solve the instance with proof logging enabled is compared to the time needed to verify the produced proof.

References

1. Akgün, Ö., Gent, I.P., Jefferson, C., Miguel, I., Nightingale, P.: Metamorphic testing of constraint solvers. In: Proceedings of the 24th International Conference on Principles and Practice of Constraint Programming (CP '18). pp. 727–736 (2018)
2. Alkassar, E., Böhme, S., Mehlhorn, K., Rizkallah, C., Schweitzer, P.: An introduction to certifying algorithms. it - Information Technology Methoden und innovative Anwendungen der Informatik und Informationstechnik **53**(6), 287–293 (2011)
3. Alviano, M., Dodaro, C., Faber, W., Leone, N., Ricca, F.: WASP: A native ASP solver based on constraint learning. In: Logic Programming and Nonmonotonic Reasoning, 12th International Conference, LPNMR '13. pp. 54–66 (2013)
4. Avellaneda, F.: A short description of the solver EvalMaxSAT. MaxSAT Evaluation Solver and Benchmark Descriptions pp. 10–11 (2021)
5. Bailleux, O., Boufkhad, Y.: Efficient CNF encoding of Boolean cardinality constraints. In: Rossi, F. (ed.) Principles and Practice of Constraint Programming (CP '03). pp. 108–122 (2003)
6. Biere, A.: Tracecheck. http://fmv.jku.at/tracecheck/ (2006), accessed on 2021–03–19
7. Bogaerts, B., Gocht, S., McCreesh, C., Nordström, J.: Certified symmetry and dominance breaking for combinatorial optimisation. In: Proceedings of the Thirty-Sixth AAAI Conference on Artificial Intelligence (AAAI '22) (2022), accepted
8. Bonet, M.L., Levy, J., Manyà, F.: Resolution for Max-SAT. Artif. Intell. **171**(8-9), 606–618 (2007)
9. Brummayer, R., Lonsing, F., Biere, A.: 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). pp. 44–57 (2010)
10. Carnegie Mellon University, University of Helsinki, University of Toronto: The 2021 MaxSAT Evaluation (MSE 2021). Website: https://maxsat-evaluations.github.io/2021/, last accessed June 2022

11. Cook, W., Koch, T., Steffy, D.E., Wolter, K.: A hybrid branch-and-bound approach for exact rational mixed-integer programming. Mathematical Programming Computation **5**(3), 305–344 (2013)
12. Cook, W.J., Coullard, C.R., Turán, G.: On the complexity of cutting-plane proofs. Discrete Applied Mathematics **18**(1), 25–38 (1987)
13. Cruz-Filipe, L., Heule, M.J.H., Hunt, W.A., Kaufmann, M., Schneider-Kamp, P.: Efficient certified RAT verification. In: Proceedings of the 26th International Conference on Automated Deduction (CADE-26). pp. 220–236 (2017)
14. Cruz-Filipe, L., Marques-Silva, J., Schneider-Kamp, P.: Efficient certified resolution proof checking. In: Proceedings of the 23rd International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS '17). pp. 118–135 (2017)
15. De Cat, B., Bogaerts, B., Devriendt, J., Denecker, M.: Model expansion in the presence of function symbols using constraint programming. In: 2013 IEEE 25th International Conference on Tools with Artificial Intelligence (ICTAI '13). pp. 1068–1075 (2013)
16. Eén, N., Sörensson, N.: An extensible SAT-solver. In: Giunchiglia, E., Tacchella, A. (eds.) SAT. LNCS, vol. 2919, pp. 502–518. Springer (2003)
17. Elffers, J., Gocht, S., McCreesh, C., Nordström, J.: Justifying all differences using pseudo-Boolean reasoning. In: Proceedings of the 34th AAAI Conference on Artificial Intelligence (AAAI '20). pp. 1486–1494 (2020)
18. Fleury, M.: Formalization of logical calculi in Isabelle/HOL. Ph.D. thesis, Saarland University, Saarbrücken, Germany (2020)
19. Gebser, M., Kaufmann, B., Schaub, T.: Conflict-driven answer set solving: From theory to practice. Artif. Intell. **187**, 52–89 (2012)
20. Gillard, X., Schaus, P., Deville, Y.: SolverCheck: Declarative testing of constraints. In: Proceedings of the 25th International Conference on Principles and Practice of Constraint Programming (CP '19). pp. 565–582 (2019)
21. Gocht, S., Martins, R., Nordström, J., Oertel, A.: Certified CNF translations for pseudo-boolean solving. In: The 25nd International Conference on Theory and Applications of Satisfiability Testing (SAT '22) (2022), accepted
22. Gocht, S., McBride, R., McCreesh, C., Nordström, J., Prosser, P., Trimble, J.: 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). pp. 338–357 (2020)
23. Gocht, S., McCreesh, C., Nordström, J.: An auditable constraint programming solver. In: The 28th International Conference on Principles and Practice of Constraint Programming (CP '22) (2022), accepted
24. Gocht, S., McCreesh, C., Nordström, J.: Subgraph isomorphism meets cutting planes: Solving with certified solutions. In: Proceedings of the 29th International Joint Conference on Artificial Intelligence (IJCAI '20). pp. 1134–1140 (2020)
25. Gocht, S., Nordström, J.: Certifying parity reasoning efficiently using pseudo-Boolean proofs. In: Proceedings of the 35th AAAI Conference on Artificial Intelligence (AAAI '21). pp. 3768–3777 (2021)
26. Goldberg, E., Novikov, Y.: Verification of proofs of unsatisfiability for CNF formulas. In: Proceedings of the Conference on Design, Automation and Test in Europe (DATE '03). pp. 886–891 (2003)
27. Heule, M.J.H., Hunt Jr., W.A., Wetzler, N.: Trimming while checking clausal proofs. In: Proceedings of the 13th International Conference on Formal Methods in Computer-Aided Design (FMCAD '13). pp. 181–188 (2013)

28. Heule, M.J.H., Hunt Jr., W.A., Wetzler, N.: Verifying refutations with extended resolution. In: Proceedings of the 24th International Conference on Automated Deduction (CADE-24). pp. 345–359 (2013)
29. Ignatiev, A., Morgado, A., Marques-Silva, J.: RC2: an Efficient MaxSAT Solver. Journal on Satisfiability, Boolean Modeling and Computation **11**, 53–64 (09 2019)
30. Jahren, E., Achá, R.A.: The MSUSorting MaxSAT solver. MaxSAT Evaluation Solver and Benchmark Descriptions p. 15 (2017)
31. Koshimura, M.: Qmaxsat: Q-dai maxsat solver. Website: https://sites.google.com/site/qmaxsat/, last accessed June 2022
32. Koshimura, M., Zhang, T., Fujita, H., Hasegawa, R.: QMaxSAT: A Partial Max-SAT Solver. Journal on Satisfiability, Boolean Modeling and Computation **8**(1-2), 95–100 (Jan 2012). https://doi.org/10.3233/SAT190091
33. Larrosa, J., Nieuwenhuis, R., Oliveras, A., Rodríguez-Carbonell, E.: A framework for certified Boolean branch-and-bound optimization. J. Autom. Reason. **46**(1), 81–102 (2011)
34. Manlove, D.F., O'Malley, G.: Paired and altruistic kidney donation in the UK: Algorithms and experimentation. In: Proceedings of the 11th International Symposium on Experimental Algorithms (SEA '12). pp. 271–282 (2012)
35. Marek, V., Truszczyński, M.: Stable models and an alternative logic programming paradigm. In: Apt, K.R., Marek, V., Truszczyński, M., Warren, D.S. (eds.) The Logic Programming Paradigm: A 25-Year Perspective, pp. 375–398 (1999)
36. Marques Silva, J.P., Lynce, I., Malik, S.: Conflict-driven clause learning SAT solvers. In: Handbook of Satisfiability, pp. 131–153 (2009)
37. Martins, R., Terra-Neves, M., Joshi, S., Janota, M., Manquinho, V., Lynce, I.: Open-wbo in MaxSAT evaluation 2017. MaxSAT Evaluation Solver and Benchmark Descriptions p. 17 (2017)
38. McConnell, R.M., Mehlhorn, K., Näher, S., Schweitzer, P.: Certifying algorithms. Computer Science Review **5**(2), 119–161 (2011)
39. Morgado, A., Marques-Silva, J.: On validating Boolean optimizers. In: IEEE 23rd International Conference on Tools with Artificial Intelligence, ICTAI 2011. pp. 924–926 (2011)
40. Py, M., Cherif, M.S., Habet, D.: A proof builder for Max-SAT. In: 4th International Conference on Theory and Applications of Satisfiability Testing (SAT 2021). pp. 488–498. Springer (2021)
41. Vandesande, D., De Wulf, W., Bogaerts, B.: QMaxSATpb: A Certified MaxSAT Solver. Website: https://github.com/wulfdewolf/CertifiedMaxSAT, last accessed June 2022
42. Vandesande, D., De Wulf, W., Bogaerts, B.: QMaxSATpb: A Certified MaxSAT Solver: Patches & Benchmarks. Website: https://github.com/wulfdewolf/CertifiedMaxSAT_benchmarks, last accessed June 2022
43. Wetzler, N., Heule, M.J.H., Hunt Jr., W.A.: 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). pp. 422–429 (2014)
44. Zhang, H.: Combinatorial designs by SAT solvers. In: Handbook of Satisfiability, pp. 533–568