A certified MaxSAT solver

Dieter Vandesande Wolf De Wulf

January 4, 2022

Abstract

This article contains the documentation for the first project of the Discrete modelling, Optimisation and Search course. The project consists of creating a MaxSAT solver that outputs verifiable proofs. The iterative partial MaxSAT solver QMaxSAT was chosen to extend to do so. QMaxSAT implements a totalizer encoding to encode cardinality constraints.

1 Introduction

In SAT, proof logging is well established. When a SAT-solver returns a model, and thus states that the theory is satisfiable, verifying if the found model is indeed a model of the theory proves satisfiability. On the other hand, when a SAT-solver states that a given theory is unsatisfiable, there is no model that can be used as proof and thus it could be that bugs in the solver result in incorrect solutions. In the SAT-competition of 2013 (Balint et al., 2013), a SAT-solver named BreakIDGlucose found a number of instances to be unsatisfiable. Unfortunately, the solver did not actually find the theories to be unsatisfiable but only came to this conclusion due to a bug. Once the bug was fixed, BreakIDGlucose could not solve the instances anymore. Therefore, proof logging for SAT-solvers became mandatory in the latest SAT-competitions (Heule et al., 2019). A proof, also called a certificate, is written by a solver and should be written efficiently, as it should not influence the performance of the solver. Moreover, proofs need to be efficiently verifiable. An example of a proof verifier is VeriPB (Gocht et al., 2020). VeriPB expresses problems as pseudo-Boolean (PB) problems and verifies proofs using a combination of Reverse Unit Propagation (RUP) and the Cutting Planes proof system (Cook et al., 1987). The Maximum Satisfiability problem (MaxSAT) (Biere et al., 2009; C. M. Li & Manya, 2009) consists of finding a model for a theory that ensures the least possible unsatisfied clauses. Variants of the MaxSAT problem exist. Partial MaxSAT allows for soft and hard clauses such that hard clauses can not be unsatisfied. Weighted MaxSAT allows for assigning penalties for unsatisfied clauses. Various approaches to solving the MaxSAT problem exist. Core-guided MaxSAT-solvers and MaxSAT-solvers using Minimum Hitting Sets (MHS) start with unsatisfiable cores and try to manipulate these cores until a satisfiable subset of the clauses is found. Iterative (Koshimura et al., 2012) and Branch and Bound (BnB) (Abramé & Habet, 2014; C.-M. Li et al., 2021) solvers look for a model and try to improve it until no better model can be found. Except for the BnB-solvers, most MaxSAT-solvers translate their input into SAT and repeatedly call a SAT-solver as an oracle.

As part of the assignment, we received an unpublished extension to VeriPB that allows for verifying proofs for optimisation problems (Bogaerts et al., 2022). Aside from adding some new proof rules, the extension keeps track of a lower bound for the optimisation. When the optimisation value is improved upon, VeriPB updates its bound and automatically learns a constraint that constrains the optimisation value to be one less than what was just found. When learning this constraint results in a contradiction, the last found best solution is the optimal solution to the problem. In what follows, VeriPB denotes a version of the proof verifier that includes the mentioned extension.

Because of their simplicity and the resemblance of their strategies to VeriPB's optimisation extension, we decided to try to extend an existing iterative MaxSAT solver to output VeriPB verifiable proofs. As solver to extend we chose the partial MaxSAT solver QMaxSAT[1] (Koshimura et al., 2012). QMaxSAT was the best performing MaxSAT-solver in the partial MaxSAT industrial competition of 2010 ("Max-SAT 2010 Fifth Max-SAT Evaluation Results", n.d.). QMaxSAT implements a simple iterative MaxSAT solver approach. To be able to constrain the number of falsified soft clauses, QMaxSAT encodes cardinality constraints using a totalizer encoding (Bailleux & Boufkhad, 2003).

The rest of this article is organised as follows. Section 2 gives preliminaries. Section 3 discusses QMaxSAT and the totalizer encodings it uses. Section 4 explains which adaptions were made to QMaxSAT to make it output proofs. Section 5 explains how totalizer cardinality encodings can be verified by VeriPB. Finally, Section 6 concludes the article with a number of conclusions.

[1]The source code that contains the extended version of QMaxSAT and a patch file that applies this extension to the QMaxSAT source code can be found on GitHub: https://github.com/wulfdewolf/CertifiedMaxSAT

2 Preliminaries

The following preliminaries contain only what is needed to understand what is discussed in this article. For a more complete version we refer the reader to Buss and Nordström (2021).

Propositional logic A Boolean variable x ranges over values *true* and *false*. We represent *true* by 1 and *false* by 0. A literal over a Boolean variable x is x itself or its negation $\overline{x}$. A clause $C = a_1 \vee \cdots \vee a_k$ is a disjunction of literals over distinct variables. The empty clause is denoted $\bot$. A truth assignment is a mapping from variables to 0 or 1, a substitution can also map variables to literals. We write $p \circ \omega$ to denote the composed substitution, resulting from first applying ω and then p. Assignments can be *partial*, i.e., some variables are unassigned. An assignment is *total* if it assigns values to all variables under consideration. A (partial) assignment p is represented by a set of literals set to *true* by p. We write $p(l_i) = 1$ if $l_i \in p$ and $p(l_i) = 0$ if $\overline{l_i} \in p$. The assignment p satisfies a clause C if it assigns at least one of C's literals to true, it falsifies C if it assigns all its literals to false. An assignment p satisfies a conjunctive normal form (CNF) formula, i.e. a conjunction of clauses, if it satisfies all the clauses of F. A formula F is *satisfiable* if there exists some assignment, i.e. a model, that satisfies it, otherwise it is *unsatisfiable*. Given a formula F then the Boolean Satisfiability problem (SAT) consists in finding an assignment that satisfies F. Given a formula F then the Maximum Boolean Satisfiability problem (MaxSAT) consists in finding an assignment that satisfies as many clauses of F as possible.

Pseudo-Boolean theories A pseudo-Boolean (PB) constraint is a linear inequality

$$C = \sum_i a_i l_i \geq A \tag{1}$$

where a_i and A are integers, l_i are literals, and $=$ denotes syntactic equality. Without loss of generality it can be assumed that PB constraints are *normalized*, hence, all literals l_i are distinct variables and the coefficients a_i and A are non-negative. The negation of (1) is:

$$\overline{C} = \sum_i -a_i l_i \geq -A + 1 \tag{2}$$

A PB formula or theory is a conjunction of PB constraints. It is important to note that a clause $l_1 \vee \cdots \vee l_k$ is equivalent to the PB constraint $1 \cdot l_1 + \cdots + 1 \cdot l_k \geq 1$. Hence, SAT formulas in CNF are special cases of PB formulas. A PB constraint C is satisfied by an assignment p if $\sum_{p(l_i)=1} a_i \geq A$. A PB formula is satisfied by an assignment p if all constraints in it are. If no satisfying assignment exists, the formula is unsatisfiable. We also consider optimisation problems, where in addition to a formula F, an integer linear objective function $f = \sum_i w_i l_i$ is given and the task is to find an assignment that satisfies F and minimises f. To deal with maximisation problems the objective function can just be negated.

Proof systems The main task of a SAT solver is to determine whether or not a given formula is satisfiable. If a formula is satisfiable there is always a short proof of its satisfiability, namely, a satisfying truth assignment. On the other hand, proving that a formula is unsatisfiable is usually done using some kind of proof system. Proof systems are methods for iteratively deriving clauses/constraints C from a formula F. Informally, the strength of a proof system is measured by how small proofs it can produce. The size of a formula or proof is equal to the number of symbols it contains. There exist a variety of proof systems that can be used to prove the unsatisfiability of SAT formulas: resolution, polynomial calculus, cutting planes, etc. In the context of this article, resolution and cutting planes are the ones that are discussed. The resolution proof system works for SAT formulas and consists of a single derivation rule:

$$\frac{B \vee x \qquad C \vee \overline{x}}{B \vee C} \tag{3}$$

with B and C clauses and x a Boolean variable. On the other hand, the cutting planes proof system (Cook et al., 1987) is designed to derive constraints from a PB formula. It consists of the following rules:

- Literal axioms:

$$\frac{}{l_i \geq 0} \tag{4}$$

- Linear combination:

$$\frac{\sum_i a_i l_i \geq A \qquad \sum_i b_i l_i \geq B}{\sum_i (c_A a_i + c_B b_i) l_i \geq c_A A + c_B B} \quad [c_A, c_B > 0] \tag{5}$$

- Division:

$$\frac{\sum_i c a_i l_i \geq A}{\sum_i a_i l_i \geq \lceil \frac{A}{c} \rceil} \quad [c > 0] \tag{6}$$

- Saturation:

$$\frac{\sum_i a_i l_i \geq A}{\sum_i min\{a_i, A\} l_i \geq A} \tag{7}$$

In what follows, we write $F \vdash C$ if C can be derived from F through a number of applications of the above cutting planes rules. Aside from these cutting planes rules, the extension of VeriPB (Bogaerts et al., 2022) that is used here has a number of other rules that can be used to verify a proof. The ones that are used in this article are:

- Reverse Unit Propagation (RUP): A formula F implies a constraint C by reverse unit propagation if $F \wedge \overline{C}$ propagates to a conflict under the empty assignment. It can be shown that $F \vdash C$ if and only if $F \wedge \overline{C} \vdash \bot$.
- Substitution Redundancy (SR): the substitution redundancy rule in VeriPB allows for deriving constraints that are not implied by the current formula, i.e., the input formula extended with previously learned constraints. Bogaerts et al. (2022) state that it can be shown that any satisfying assignment p of the current formula can be transformed into another assignment p' that also satisfies C with the optimisation value for p' smaller than or equal to that for p. To do so, a substitution ω needs to be provided such that if p satisfies the current formula but falsifies C, then $p \circ \omega$ still satisfies the current formula and also satisfies C.

For more information about these rules we refer the reader to Bogaerts et al. (2022).

3 QMaxSAT

QMaxSAT is a partial MaxSAT solver built on top of MiniSat (Eén & Sörensson, 2004). The QMaxSAT version that is used here is the first ever QMaxSAT version (0.1; Koshimura et al., 2012). While newer versions exist, the first version was chosen because of its simplicity. In what follows, QMaxSAT's algorithm is discussed as well as the totalizer encoding it uses to encode cardinality constraints.

Algorithm Algorithm 1 gives pseudo-code for the QMaxSAT algorithm. The MaxSAT solver starts by adding relaxation variables to every soft clause. When a relaxation variable of a soft clause is falsified, this means that the corresponding clause needs to be satisfied. Hence, optimisation occurs by looking for models that falsify the most relaxation variables. QMaxSAT iteratively calls MiniSat with the same theory. Therefore, MiniSat can keep its learned clauses, which improves the performance of later MiniSat calls. At first, no additional constraints on the number of satisfied relaxation variables are added. Hence, when MiniSat is unable to find a model in the first call, this means that it is not possible to satisfy all the hard clauses. Otherwise, the first lower bound is set to the number of satisfied relaxation variables n. Every time a model is found, a cardinality constraint that encodes that the number of satisfied relaxation variables is less than or equal to $n-1$ is added. This means that if a new model is found afterwards, it must have satisfied less relaxation variables and thus more soft clauses. The lower bound can then be updated to the number of satisfied relaxation variables in the new model and consequently, the cardinality constraint on the number of satisfied relaxation variables can be updated. If no new model can be found, it is not possible to find a model that satisfies more soft clauses and thus the last found model is the optimal solution.

Totalizer cardinality encoding QMaxSAT implements a totalizer encoding to encode the cardinality constraints on the number of satisfied relaxation variables (Koshimura et al., 2012). Said encoding is presented here as was originally done by Bailleux and Boufkhad (2003).

Definition 3.1 The totalizer is a CNF formula defined on three sets of variables:

1. $E = \{e_1, ..., e_n\}$: the set of input variables.
2. $Y = \{v_1^E, ..., v_n^E\}$: the set of output/counting variables.
3. L: a set of variables called linking variables.

In QMaxSAT, the input variables correspond to the relaxation variables. The counting variables $v_i^E \in Y$ have as semantics that if v_i^E is assigned to true, at least i of the input variables E are satisfied. If v_i^E is assigned false, there are less than i input variables true. For example, when there are four relaxation variables $R = \{r_1, r_2, r_3, r_4\}$, then there are four counting variables $Y = \{v_1^R, v_2^R, v_3^R, v_4^R\}$. Every model that assigns three relaxation variables to true, will also assign v_1^R to v_3^R to true and v_4^R to false. The totalizer encoding is generated through a complex tree-recursive function. Figure 1 depicts an example of a tree that said function

Algorithm 1: QMaxSAT

input: a set of hard clauses H, a set of soft clauses S, a SAT-solver Solver
output: 'UNSAT', optimalSolution
foreach $s_i \in S$ **do**
    $r_i \leftarrow$ new relaxation variable
    $R \leftarrow R \cup r_i$
    $h_i \leftarrow s_i \vee r_i$
    $H \leftarrow H \cup h_i$
model $\leftarrow$ Solver.solve(H)
if *model = 'UNSAT'* **then**
    return 'UNSAT'
bestSoFar $\leftarrow$ number of satisfied variables in R
$C, Y \leftarrow$ generateTotalizerClauses(R) `/* with` C `the totalizer clauses and` $v_i^R \in Y$ `the counting variable meaning that at least` i `variables in` R `are satisfied. */`
$H \leftarrow H \cup C$
$H \leftarrow H \cup \{\overline{v_i^R} | i \geqslant \text{bestSoFar}\}$
repeat
    model $\leftarrow$ Solver.solve(H)
    if *model* $\neq$ *'UNSAT'* **then**
        bestSoFar $\leftarrow$ number of satisfied variables in R
        $H \leftarrow H \cup \{\overline{v_i^R} | i \geqslant \text{bestSoFar}\}$
until *model = 'UNSAT'*;
return bestSoFar

generates for an instance with four clauses and thus four relaxation variables. Every node that counts for a set of variables of size $m \geq 2$ divides this set into two disjoint sets of variables, one of size $m_1 = \lfloor m/2 \rfloor$ and one of size $m_2 = m - \lfloor m/2 \rfloor$. This procedure is recursively repeated until the leaves are reached, where the variable sets only contain a single variable. Said variables will always be one of the input variables.

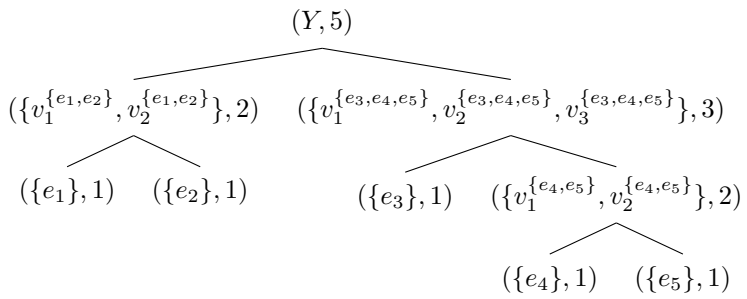

Figure 1: Example of totalizer generated tree. Example after (Bailleux & Boufkhad, 2003)

Take, for example, an internal node with linking variables $L = \{v_1^{E_j}, ..., v_m^{E_j}\}$ of size m. Every linking variable $v_i^{E_j}$ has as meaning that in the subset $E_j \subset E$, at least i variables are assigned to true. To that end, two disjoint subsets $E_k \subset E_j$ of size m_1 and $E_l \subset E_j$ of size m_2 are created and two corresponding children of the current node are built. The linking variables for the subset E_k are $\{v_1^{E_k}, ..., v_{m_1}^{E_k}\}$ and the linking variables for the subset E_l are $\{v_1^{E_l}, ..., v_{m_2}^{E_l}\}$. For all combinations of $0 \leq \alpha \leq \lfloor m/2 \rfloor$, $0 \leq \beta \leq m - \lfloor m/2 \rfloor$, $0 \leq \sigma \leq m$, with $\alpha + \beta = \sigma$, the following clauses are added:

$$C_1(\alpha, \beta, \sigma) = \overline{v_\alpha^{E_l}} \vee \overline{v_\beta^{E_k}} \vee v_\sigma^{E_j} \tag{8}$$

$$C_2(\alpha, \beta, \sigma) = v_{\alpha+1}^{E_l} \vee v_{\beta+1}^{E_k} \vee \overline{v_{\sigma+1}^{E_j}} \tag{9}$$

with $v_0^{E_j} = v_0^{E_k} = v_0^{E_l} = 1$ and $v_{m+1}^{E_j} = v_{m_1+1}^{E_k} = v_{m_2+1}^{E_l} = 0$. The C_1 clauses ensure that when the sum of the variables that are assigned to true in E_k and E_l is larger than or equal to σ, the linking variable $v_\sigma^{E_j}$ is assigned to true. On the other hand, the C_2 clauses ensure that when the sum of the variables that are assigned to true in E_k and E_l is smaller than $\sigma + 1$, the variable $v_{\sigma+1}^{E_j}$ is assigned to false. The totalizer encoding consists of the conjunction of all the clauses of all the internal nodes. The counting variables are the linking variables of the root.

Comparator Finally, Bailleux and Boufkhad (2003) also define a comparator.

> **Definition 3.2** The comparator is a set of unary clauses that are satisfied if and only if the instantiation of the input variables of the totalizer represents an interval that matches with the cardinality constraint that the comparator specifies. The constraint $\mu \leq N(E) \leq \rho$ is specified as follows:
>
> $$\bigwedge_{1 \leq i \leq \mu} (v_i^E) \quad \bigwedge_{\rho+1 \leq j \leq m} (\overline{v_j^E})$$

QMaxSAT only uses clauses to constrain the maximum number of relaxation variables to be assigned to true. Therefore, if the cardinality constraint on the relaxation variables is $\sum_{r_i \in R} 1r_i \leq n$ then the clauses $C_3 = \overline{v_j^E}$ for all $j > n$ are added.

4 Proof logging

To make QMaxSAT output VeriPB verifiable proofs, it was extended with a Prooflogger class that can be called from the MiniSat core. The Prooflogger class has two types of methods, the first kind write to a proof PBP file while the second kind write to a pseudo-Boolean OPB file. In what follows, five important aspects of the implementation of the extension to QMaxSAT are discussed.

Input file transformation An adapted version of the input problem is needed because QMaxSAT takes WDIMACS files as input and VeriPB requires OPB input. Therefore, QMaxSAT was extended to call its Prooflogger object to write the clauses of the input problem to an OPB file, immediately after parsing them. It is important to note that the OPB file contains the clauses after adding relaxation variables as well as a minimise statement that states that the number of relaxation variables that are assigned to *true* should be minimised.

Learned clauses When a call to the SAT oracle results in learning a reason clause, this clause needs to be added to the proof. This is done by calling a specific Prooflogger method for writing learned clauses. For example, the following clause:

$$x_1 \vee x_2 \vee \overline{x_3}$$

would be written to the proof file as:

```
u 1 x1 1 x2 1 ~x3 >= 1;
```

where the 'u' indicates that VeriPB should try to prove this clause through RUP.

Totalizer encoding Before writing the totalizer encoding clauses to the proof file, their underlying meanings are added to the proof through substitution redundancy. Take, for example, the linking variable $v_2^{\{x3,x4,x5\}}$ and the totalizer C_1 and C_2 clauses that define this variable. Before QMaxSAT writes the C1 and C2 clauses to the proof file, the PB cardinality definitions of the linking variables are written. They are proven by the application of the substitution redundancy rule:

```
red 1 ~x3 1 ~x4 1 ~x5 2 v2_x3_x5 >= 2; v2_x3_x5 -> 1
red 1 x3 1 x4 1 x5 2 ~v2_x3_x5 >= 2; v2_x3_x5 -> 0
```

Because $v_2^{\{e_3,e_4,e_5\}}$ is an auxiliairy variable and the above constraints are the first that it occurs in, VeriPB can prove the substitution redundancy by itself. Once VeriPB has these constraints, a number of cutting planes derivations that derive the corresponding C_1 and C_2 clauses from them can be written to the proof file. The actual derivations are discussed in the next section. It is important to note that, to keep the proof file readable, above cardinality definitions and the cutting planes derivations are split up in the proof file. Firstly, all the substitution redundancy rules are written in a tree-like manner. Afterwards, in a separate proof section and in the same tree-like manner, the cutting planes derivations are written. Once the C_1 and C_2 clauses have been derived, we show that they were proven by adding them to the proof file using a RUP rule similar to the one in the previous paragraph. Lastly, note that the meaningful names of the auxiliairy variables can be toggled through a command line option.

Objective bound updates Every time a call to the SAT oracle results in a model, this model needs to be written to the proof file such that VeriPB can update its lower bound. Models are written to the proof file as follows, for example, the model $\{x_1, x_2, \overline{x_3}\}$ is written using a VeriPB objective bound update rule:

```
o x1 x2 ~x3
```

Such a rule results in VeriPB verifying whether or not the encoded model is actually a model of the input theory and if so, adding a cardinality constraint that constrains the optimisation value to be at least one less than the newly found optimisation value.

Constraining relaxation variables After having written the objective bound update to the proof file, QMaxSAT learns a number of C_3 clauses that force a number of counting variables to be false, thereby constraining the relaxation variables. To prove these clauses, once more, a number of cutting planes derivations are written that derive them from the cardinality definitions. However, in this case, the cardinality definitions are not enough, the actual cardinality constraint that is added by VeriPB after updating its lower bound will be needed as well. The cutting planes derivations are clarified in the next section. Once the C_3 clauses have been derived, we again show that they were by adding them using a RUP rule.

5 Proving the cardinality constraint encoding

In what follows, a generalised process for deriving the totalizer C_1 and C_2 clauses from the cardinality definitions is given. The process is proven by induction. Afterwards, a generalised process for deriving the comparator C_3 clauses from the cardinality definitions and the VeriPB objective bound constraint is given.

Cardinality definitions As mentioned, through substitution redundancy, the underlying semantics of the totalizer's linking variables can be defined using the following constraints:

> **Definition 5.1** Let S be a set of variables with $n \leq |S| + 1$. Let the variable v_n^S denote the linking variable that is satisfied if and only if there are at least n satisfied variables in S. The PB cardinality constraints that define the semantics of v_n^S are defined as follows:
>
> - $P_{1,n}^S := \sum_{x_i \in S} 1\overline{x_i} + (|S| - n + 1)v_n^S \geq |S| - n + 1$
> - $P_{2,n}^S := \sum_{x_i \in S} 1x_i + n\overline{v_n^S} \geq n$

These PB constraints are added to the proof file before the corresponding C_1 and C_2 constraints are. Therefore, it is guaranteed that VeriPB can automatically prove them as the linking variables that they contain will not have occurred in the proof file yet.

Deriving C_1 constraints The following theorem states that the totalizer C_1 constraints can be derived from the substitution redundancy cardinality definitions of the linking variables that it consists of:

> **Theorem 5.1** Let S be the set of variables of an internal node of the totalizer encoding tree for which we there are two subsets, S_a and S_b such that $S_a \cap S_b = \emptyset$ and $S_a \cup S_b = S$. Let α and β be two numbers such that $\alpha + \beta = \sigma$ with $\sigma \leq |S|$. Using the PB cardinality constraints for the linking variables of S, S_a, and S_b defined by Definition 5.1, it is possible to derive the totalizer C_1 clauses that encode the meaning of the linking variables of S.

Proof: The proof goes per induction on $|S|$. If $S = \{x_0\}$, and thus $|S| = 1$, this means that the node is a leaf. For the leafs there are no C1 and C2 clauses to derive, and therefore, the theorem is trivially true. It is important to note that the totalizer also uses a number of trivial C_1 and C_2 clauses. Say v_0^S is the variable that denotes that at least 0 variables in S are satisfied, which is trivially true and let v_2^S be the variable that denotes that at least 2 variables in S are satisfied, which is trivially false. Also, consider x_0 the only variable in S such that $v_1^S = x_0$. Then, the totalizer uses C_1 clauses that correspond to the following trivial PB cardinality definitions:

$$P_{1,0}^S = 1\overline{x_0} + 2v_0^S \geq 2$$
$$P_{1,1}^S = 1\overline{x_0} + 1x_0 \geq 1$$
$$P_{1,2}^S = 1\overline{x_0} + 0v_2^S \geq 0$$

Now for the induction step, let S be a set of variables with $|S| > 1$. Then, the totalizer clauses for this set are created by tree recursion. This means that we have two disjoint subsets S_a and S_b of S. We will now show that for every α, β and σ for which $\alpha + \beta = \sigma$, we can reconstruct the corresponding C_1 clause from the corresponding PB cardinality definitions. Since we are in an internal node, the following PB cardinality definitions are defined:

$$\begin{aligned}
P^{S_a}_{2,\alpha} &:= \left(\sum_{x_i \in S_a} 1x_i\right) + \alpha\overline{v^{S_a}_{\alpha}} \geq \alpha \\
P^{S_b}_{2,\beta} &:= \left(\sum_{x_j \in S_b} 1x_j\right) + \beta\overline{v^{S_b}_{\beta}} \geq \beta \\
P^{S}_{1,\sigma} &:= \left(\sum_{x_k \in S} 1\overline{x_k}\right) + (|S| - \sigma + 1)v^S_\sigma \geq |S| - \sigma + 1 \\
&:= \left(\sum_{x_i \in S_a} 1\overline{x_i}\right) + \left(\sum_{x_j \in S_b} 1\overline{x_j}\right) + (|S| - \sigma + 1)v^S_\sigma \geq |S| - \sigma + 1
\end{aligned}$$

The last step is allowed because $S_a \cap S_b = \emptyset$ and $S_a \cup S_b = S$. Summing these results in:

$$\sum_{x_i \in S_a} 1x_i + \sum_{x_j \in S_b} 1x_j + \sum_{x_i \in S_a} 1\overline{x_i} + \sum_{x_j \in S_b} 1\overline{x_j} + \alpha\overline{v^{S_a}_{\alpha}} + \beta\overline{v^{S_b}_{\alpha}} + (|S| - \sigma + 1)v^S_\sigma \geq \alpha + \beta + |S| - \sigma + 1$$

Which can be simplified to:

$$\alpha\overline{v^{S_a}_{\alpha}} + \beta\overline{v^{S_b}_{\beta}} + (|S| - \sigma + 1)v^S_\sigma \geq 1$$

because $\alpha + \beta - \sigma = 0$ and thus we can subtract $|S|$ from the right hand side because of the pairwise opposite signs of the first four terms. Lastly, a saturation step results in:

$$\overline{v^{S_a}_{\alpha}} + \overline{v^{S_b}_{\beta}} + v^S_\sigma \geq 1$$

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

Deriving C_2 constraints The following theorem states that the totalizer C_2 constraints can be derived from the cardinality definitions of the linking variables that it consists of:

> **Theorem 5.2** Let S be the set of variables of an internal node of the totalizer encoding tree for which we there are two subsets, S_a and S_b such that $S_a \cap S_b = \emptyset$ and $S_a \cup S_b = S$. Let α and β be two numbers such that $\alpha + \beta = \sigma$ with $\sigma \leq |S|$. Using the PB cardinality constraints for the linking variables of S, S_a, and S_b defined by Definition 5.1, it is possible to derive the totalizer C_2 clauses that encode the meaning of the linking variables of S.

Proof: Just as for the C_1 constraints, the base case is trivial. Also, it is again important to note that the totalizer uses a number of trivial C_2 clauses. Say v^S_0 is the variable that denotes that at least 0 variables in S are satisfied, which is trivially true and let v^S_2 be the variable that denotes that at least 2 variables in S are satisfied, which is trivially false. Also, consider x_0 the only variable in S such that $v^S_1 = x_0$. Then, the totalizer uses C_2 clauses that correspond to the following trivial PB cardinality definitions:

$$\begin{aligned}
P^S_{2,0} &= 1x_0 + 0\overline{v^S_0} \geq 0 \\
P^S_{2,1} &= 1x_0 + 1\overline{x_0} \geq 1 \\
P^S_{2,2} &= 1x_0 + 2\overline{v^S_2} \geq 2
\end{aligned}$$

Now for the induction step, let $\alpha_1 = \alpha + 1$, $\beta_1 = \beta + 1$, $\sigma_1 = \sigma + 1$. The following PB cardinality definitions are defined:

$$\begin{aligned}
P^{S_a}_{1,\alpha_1} &:= \sum_{x_i \in S_a} 1\overline{x_i} + (|S_a| - \alpha_1 + 1)v^{S_a}_{\alpha_1} \geq |S_a| - \alpha_1 + 1 \\
P^{S_b}_{1,\beta_1} &:= \sum_{x_j \in S_b} 1\overline{x_j} + (|S_b| - \beta_1 + 1)v^{S_b}_{\beta_1} \geq |S_b| - \beta_1 + 1 \\
P^{S}_{2,\sigma_1} &:= \sum_{x_k \in S} 1x_k + \sigma_1\overline{v^S_{\sigma_1}} \geq \sigma_1 \\
&:= \sum_{x_i \in S_a} 1x_i + \sum_{x_j \in S_b} 1x_j + (\sigma + 1)\overline{v^S_{\sigma_1}} \geq \sigma_1
\end{aligned}$$

Summing these results in:

$$\sum_{x_i \in S_a} 1\overline{x_i} + \sum_{x_i \in S_a} 1x_i + \sum_{x_j \in S_b} 1\overline{x_j} + \sum_{x_j \in S_b} 1x_j + (|S_a| - \alpha_1 + 1)v_{\alpha_1}^{S_a} + (|S_b| - \beta_1 + 1)v_{\beta_1}^{S_b} + \sigma_1\overline{v_{\sigma_1}^{S}}$$
$$\geq |S_a| - \alpha_1 + 1 + |S_b| - \beta_1 + 1 + \sigma_1$$

Which can be simplified to:

$$(|S_a| - \alpha_1 + 1)v_{\alpha_1}^{S_a} + (|S_b| - \beta_1 + 1)v_{\beta_1}^{S_b} + \sigma_1\overline{v_{\sigma_1}^{S}} \geq |S_a| - \alpha_1 + 1 + |S_b| - \beta_1 + 1 + \sigma_1 - |S|$$
$$\Leftrightarrow (|S_a| - \alpha + 1)v_{\alpha_1}^{S_a} + (|S_b| - \beta_1 + 1)v_{\beta_1}^{S_b} + \sigma_1\overline{v_{\sigma_1}^{S}} \geq 2$$

Division by 2, a saturation step, and substitution of α_1, β_1 and σ_1 results in:

$$v_{\alpha+1}^{S_a} + v_{\beta+1}^{S_b} + \overline{v_{\sigma+1}^{S}} \geq 1$$

This is exactly $C_2(\alpha, \beta, \sigma)$ from the totalizer encoding.

Deriving C_3 constraints The following theorem states that the comparator C_3 clauses can be derived from the PB cardinality definitions $P_{2,n}^R$, which define the counting variables $\{v_1^R, ..., v_{|R|}^R\}$ on the relaxation variables R. Just as for the C_1 and C_2 clauses, the C_3 clauses are only added after their derivations. Therefore, VeriPB can verify the addition of the C_3 clauses using RUP.

Theorem 5.3 Let $Y = \{v_1^R, ...v_{|R|}^R\}$ be the set of counting variables on the relaxation variables R, such that for each n, v_n^R means that there are at least n variables assigned to *true* in the relaxation variables. The PB-constraints $P_{1,n}^R$ and $P_{2,n}^R$ are the constraints that ensure the semantics of the variables v_n^R. When a new optimal solution has been found with o relaxation variables assigned to *true*, the new $C_3 := \overline{v_n^R}(\forall n \geq o)$ constraints added by QMaxSAT can be derived from the PB-constraints $P_{2,n}^R$.

Proof: When a new model that assigns o relaxation variables to true is found, because of the objective bound update rule that is then written to the proof file, VeriPB adds the following PB cardinality constraint:

$$\sum_{x_i \in R} 1\overline{x_i} \geq |R| - o + 1$$

Additionally, we know that for all $n \geq o$, the following PB cardinality definitions have already been defined:

$$P_{2,n}^S := \sum_{x_i \in S} 1x_i + n\overline{v_n^S} \geq n$$

Summing these results in:

$$\left(\sum_{x_i \in R} 1\overline{x_i}\right) + \left(\sum_{x_i \in S} 1x_i\right) + n\overline{v_n^S} \geq |R| - o + 1 + n$$

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

$$n\overline{v_n^S} \geq n - o + 1$$

Division by $n - o + 1$ followed by a saturation step results in:

$$1\overline{v_n^S} \geq 1$$

These are exactly the comparator's C_3 constraints.

6 Conclusions

An extension to the partial MaxSAT solver QMaxSAT that makes it output VeriPB verifiable proofs is presented. While QMaxSAT is not the state-of-the-art for MaxSAT solving, it is a good start for making MaxSAT solvers certified. While the totalizer encoding that it uses to constrain the number of falsified soft clauses has been improved throughout the years, the extension that is presented here should not require too much change to also work for the newer totalizer encodings (Asín et al., 2009).

Important to mention is that a lot of time was spent trying to prove the totalizer encoding in the more difficult direction. The totalizer encoding can be proven in two ways. The more difficult approach starts from the totalizer clauses and tries to derive the pseudo-Boolean cardinality constraints from them. While we believe that we came very close to succeeding in this approach, finding a generalisation for the process that could easily be implemented proved to be too difficult. Luckily, the other appraoch was more manageable. Starting from the pseudo-Boolean cardinality constraints, which can easily be proven through substitution redundancy, and trying to derive the totalizer clauses from them is a process that is easily generalised and a proof for the correctness of the generalised process was presented.

Regarding future work for this project, three things come to mind. Firstly, the extension should be thoroughly tested. While the implemented process was worked out theoretically, due to time constraints, the implementation of the actual extension had to be rushed and we can not guarantee that no bugs slipped in. Secondly, it would be nice to implement VeriPB prooflogging for newer QMaxSAT versions. Doing so would for verifying weighted partial MaxSAT instances (Zha, 2020). Finally, implementing VeriPB prooflogging for other approaches to MaxSAT solving can be an interesting direction as well.

References

Abramé, A., & Habet, D. (2014). Ahmaxsat: Description and evaluation of a branch and bound max-sat solver. *Journal on Satisfiability, Boolean Modeling and Computation*, *9*(1), 89–128.

Asín, R., Nieuwenhuis, R., Oliveras, A., & Rodríguez-Carbonell, E. (2009). Cardinality networks and their applications. In O. Kullmann (Ed.), *Theory and applications of satisfiability testing - sat 2009* (pp. 167–180). Springer Berlin Heidelberg.

Bailleux, O., & Boufkhad, Y. (2003). Efficient CNF Encoding of Boolean Cardinality Constraints. In F. Rossi (Ed.), *Principles and Practice of Constraint Programming – CP 2003* (pp. 108–122). Springer. https://doi.org/10.1007/978-3-540-45193-8_8

Balint, A., Belov, A., Heule, M. J., Järvisalo, M., et al. (2013). Proceedings of sat competition 2013: Solver and benchmark descriptions.

Biere, A., Heule, M., & van Maaren, H. (2009). *Handbook of satisfiability* (Vol. 185). IOS press.

Bogaerts, B., Gocht, S., McCreesh, C., & Nordström, J. (2022). Certified Symmetry and Dominance Breaking for Combinatorial Optimisation. *Thirty-Sixth AAAI Conference on Artificial Intelligence* (Accepted).

Buss, S., & Nordström, J. (2021). Chapter 7. Proof Complexity and SAT Solving. *Handbook of Satisfiability*, 233–350. https://doi.org/10.3233/FAIA200990

Cook, W., Coullard, C. R., & Turán, G. (1987). On the complexity of cutting-plane proofs. *Discrete Applied Mathematics*, *18*(1), 25–38. https://doi.org/10.1016/0166-218X(87)90039-4

Eén, N., & Sörensson, N. (2004). An Extensible SAT-solver. In E. Giunchiglia & A. Tacchella (Eds.), *Theory and Applications of Satisfiability Testing* (pp. 502–518). Springer. https://doi.org/10.1007/978-3-540-24605-3_37

Gocht, S., McCreesh, C., & Nordström, J. (2020). *Veripb: The easy way to make your combinatorial search algorithm trustworthy* [Presented at the workshop *From Constraint Programming to Trustworthy AI* at the *26th International Conference on Principals and Practice of Constraint Programming (CP2020)*]. https://www.csc.kth.se/~jakobn/research/VeriPB_CPTAI2020.pdf

Heule, M. J., Järvisalo, M., & Suda, M. (2019). Sat competition 2018. *Journal on Satisfiability, Boolean Modeling and Computation*, *11*(1), 133–154.

Koshimura, M., Zhang, T., Fujita, H., & Hasegawa, R. (2012). QMaxSAT: A Partial Max-SAT Solver. *Journal on Satisfiability, Boolean Modeling and Computation*, *8*(1-2), 95–100. https://doi.org/10.3233/SAT190091

Li, C. M., & Manya, F. (2009). Maxsat, hard and soft constraints. *Handbook of satisfiability* (pp. 613–631). IOS Press.

Li, C.-M., Xu, Z., Coll, J., Manyà, F., Habet, D., & He, K. (2021). Combining clause learning and branch and bound for maxsat. *27th International Conference on Principles and Practice of Constraint Programming (CP 2021)*.

Max-SAT 2010 Fifth Max-SAT Evaluation Results [Accessed: 2022-01-02]. (n.d.).

Zha, A. (2020). Qmaxsat in maxsat evaluation 2018. *MaxSAT Evaluation 2018*, 21.