TECHNISCHE UNIVERSITÄT DRESDEN

Faculty of Computer Science Institute of Artificial Intelligence Knowledge Representation and Reasoning

A Compact Encoding of Pseudo-Boolean Constraints into SAT

Steffen Hölldobler Norbert Manthey Peter Steinke

KRR Report 12-03

Mail to
Technische Universität Dresden
01062 Dresden

Bulk mail to
Technische Universität Dresden
Helmholtzstr. 10
01069 Dresden

Office
Room 2006
Nöthnitzer Straße 46
01187 Dresden

Internet
http://www.wv.inf.tu-dresden.de

A Compact Encoding of Pseudo-Boolean Constraints into SAT

Steffen Hölldobler, Norbert Manthey and Peter Steinke

Knowledge Representation and Reasoning Group
Technische Universität Dresden, 01062 Dresden, Germany
peter@janeway.inf.tu-dresden.de

Abstract. Many different encodings for pseudo-Boolean constraints to the Boolean satisfiability problem have been proposed in the past. In this work we present a novel small sized and simple to implement encoding. The encoding maintains generalized arc consistency by unit propagation and results in a linear sized formula in conjunctive normal form with respect to the number of input variables. Experimental data confirms the advantages of the encoding over existing ones for most of the relevant PB instances.

1 Introduction

Due to the various improvements in satisfiability testing (see e.g.[14,15,19,8]) SAT solvers are successfully applied to many domains like electronic design automation [12,6], periodic scheduling [11], or cryptography [10]. To encode a problem into SAT it is often necessary to translate cardinality constraints or general pseudo-Boolean (PB) constraints. In this work we concentrate on the latter one.

PB constraints of the form $\sum_{i=1}^{n} w_i x_i \leq k$ are a special case of 0-1 integer linear programming [9], where k and w_i are integers, x_i are Boolean variables, and n is the number of variables. Besides translating PB constraints into SAT, there exists also solver that handle these constraints natively. We show that native domain solvers can be outperformed by encoding PB and using SAT solvers, when using appropriate encodings.

There are different ways of translating a PB constraint into a SAT instance (see e.g. [2,5,9]), which differ in the size of the resulting formula and the properties, which are enjoyed by the formula and the used SAT solvers. Two properties are particularly important, both of which are related to unit propagation, the main inference rule within a modern SAT solver: (i) the ability to detect inconsistencies by unit propagation and (ii) maintaining general arc consistency by unit propagation. The former is achieved by running into a conflict as soon as an inconsistency is observed, whereas the latter is achieved if unit propagation assigns all variables to false which are not part of any solution of the constraint.

In particular, to encode PB constraints into SAT instances the following methods have been applied: *binary decision diagrams* [2,9], *sorting* and *adder*

networks [9] as well as the so-called *local watchdog* encoding [5]. Binary decision diagrams and the local watch dog encoding maintain general arc consistency. The former requires $\mathcal{O}(n^3\, log(k))$ clauses and variables, the latter uses $\mathcal{O}(n^3\, log(n)\, log(k))$ clauses and $\mathcal{O}(n^2\, log(n)\, log(k))$ variables. The value of n is the number of variables in the PB constraints. There also exist encodings that require much less clauses, namely sorting networks and adder networks, but these encodings do not maintain general arc consistency in general. Sorting networks encode a PB constraint with $\mathcal{O}(n\, log^2(n))$ clauses and maintain arc consistency for cardinality constraints, which are a special case of PB constraints. Adder networks require $\mathcal{O}(n\, log(k))$ clauses.

The contributions of this paper are the following: We present a new translation from PB constraints to SAT instances, called the *sequential weight counter* (SWC) encoding. For a PB constraint of the form $\sum_i w_i x_i \leq k$, where $1 \leq i \leq n$, this encoding requires $\mathcal{O}(nk)$ clauses and auxiliary variables while preserving the ability to detect inconsistencies and to maintain general arc consistency by unit propagation. Compared to the other encodings, the SWC encoding depends linearly on n and k. Furthermore, its structure is simple and easy to understand compared to complex BDDs or sorting networks. Analyzing instances of recent PB competitions shows that for more than 99 % of the PB constraints the SWC encoding produces a smaller SAT formula than with BDDs of [2] or the watchdog encoding. Finally, we provide an experimental analysis that empirically verifies the practicability of the new encoding.

The paper is structured as follows: the background of SAT and PB solving is given in Section 2. In Section 3 the sequential weight counter encoding is introduced, followed by an empirical evaluation in Section 4. Some final conclusions are presented in Section 5.

2 Preliminaries

Let V be a finite set of Boolean variables. The set of *literals* $V \cup \{\overline{x} \mid x \in V\}$ consists of positive and negative Boolean variables. A *clause* is a finite disjunction of literals whereas a *formula* (in *conjunctive normal form* (CNF)) is a finite conjunction of clauses. We sometimes consider clauses and formulas as sets of literals and sets of clauses, respectively. A *unit clause* is a clause that contains a single literal.

An *interpretation* is a (partial or total) mapping from the set of variables into the set $\{1, 0\}$ of truth values; it is represented by a set J of literals with the understanding that x is mapped to 1 if $x \in J$ and is mapped to 0 if $\overline{x} \in J$. If J can be determined from the context, then we simply write $x = 1$ or $x = 0$ in case of $J(x) = 1$ and $J(x) = 0$, respectively. If a variable x is neither mapped to 1 nor to 0 by J, we say the variable is *undefined* and write $J(x) = \mathit{undef}$ or, for short, $x = \mathit{undef}$. One should observe that $\{x, \overline{x}\} \not\subseteq J$ for any x and J.

A clause C is *satisfied* by an interpretation J if $J(l) = 1$ for some $l \in C$. An interpretation *satisfies* a formula F if it satisfies every clause in F. If there exists an interpretation that satisfies F, then F is said to be *satisfiable* or *consistent*,

otherwise it is said to be *unsatisfiable* or *inconsistent*. The *reduct* $F|_J$ of a formula F with respect to an interpretation J is the formula obtained from F by replacing each variable x by $J(x)$ if $J(x) \in \{1, 0\}$ and simplifying the formula: all satisfied clauses are removed and from all the remaining clauses all x with $J(x) = 0$ and all $\overline{x}$ with $J(x) = 1$ are removed.

In the sequel we will use gate representations for special propositional formulas. The conjunction $x \leftrightarrow (x_i \wedge x_j)$ is called *AND gate* with the *input bits* x_i and x_j and the *output bit* x. We will refer to AND gates with the symbol &. Similarly we represent the disjunction $x \leftrightarrow (x_i \vee x_j)$ by the *OR gate*. Again, x_i and x_j are are input bits and x is the output bit. The symbol used for OR gates is $\geq$1. Depending on the circuit, sometimes only a single direction of the equivalence needs to be encoded [20,16].

One of the most important inference rules in modern SAT solvers is *unit propagation* (UP) [7]. Let F be a formula and J be an interpretation, then unit propagation extends an interpretation if in the reduct there exists a unit clause: $F, J \vdash_{up} F, J \cup \{l\}$ if $\{l\} \in F|_J$. Let $\vdash^*_{up}$ be the transitive and reflexive closure of $\vdash_{up}$. UP *detects a conflict* in a formula F and an interpretation J, if the resulting reduct $F|_{J'}$ contains the empty clause: $F, J \vdash^*_{up} F, J'$ and $\{\} \in F|_{J'}$.

A *pseudo-Boolean (PB)* constraint is defined over a finite set of Boolean variables x_i and has the form $\sum_i w_i x_i \triangleright k$, where w_i (called *weights*) and k are integers, $\triangleright$ is one of the following classical relational operators $=, >, <, \leq$ or $\geq$, and $1 \leq i \leq n$, where n is the number of variables in the PB constraint. W.l.o.g. in this work we consider only PB constraints that use the $\leq$ operator and where each weight is $1 \leq w_i \leq k$. As given in [9,17], each PB constraint can be transformed into such an equivalent PB constraint over the same set of Boolean variables. For more details we point the reader to [17].

We define the multiplication of Boolean variables, an interpretation J and integers as follows: for each integer $a \in \mathbb{Z}$ we define $a \cdot x = a$ if $J(x) = 1$ and $a \cdot x = 0$ if $J(x) = 0$. A PB constraint is *consistent* or *satisfiable* if there exists an interpretation for all variables of the constraint, such that $\sum_{i=1} w_i x_i \leq k$ holds, and is inconsistent otherwise. The *PB decision problem* asks if for a set of PB constraints there exists an interpretation such that all PB constraints are satisfied.

3 Sequential Weight Counter Encoding

In this section we present the *sequential weight counter* (SWC) encoding, which is a new encoding for PB constraints of the form $\sum_i w_i x_i \leq k$ into SAT. The SWC encoding is a modification of the sequential counter (SEQ) encoding [18], which translates cardinality constraints into SAT. Due to small changes, the SWC encoding needs the same amount of clauses as the SEQ encoding, viz. at most $n(2k+1)$ clauses and $(n-1)k$ auxiliary variables, and – like SEQ – maintains generalized arc (GAC) consistency by unit propagation. On the one hand, the SWC encoding needs more clauses and variables than an adder network for PB constraints, which requires $\mathcal{O}(n\,log(k))$ variables and clauses [9], but adder

Table 1: Distribution k with respect to k in PB constraints

Number of Constraints	$k > n^2$	$n^2 \geq k > n$	$k \leq n$
22 014 154	0.56 %	0.23 %	99.2 %

networks do not maintain generalized arc consistency by unit propagation. On the other hand, if $k \leq n^2$ the SWC encoding needs less variables and clauses than the watchdog encoding [5] – which produces $\mathcal{O}(n^3\, log(n)\, log(k))$ clauses and $\mathcal{O}(n^2\, log(n)\, log(k))$ variables. The watchdog encoding is the currently best known encoding of PB constraints that maintaining generalized arc consistency by unit propagation [5].

We have analyzed the set of PB instances from recent PB competitions, where we only considered PB constraints where at least one weight is $w_i > 1$. Table 1 shows the distribution of PB constraints in the instances of the PB benchmark 2011 and 2010[1]. The analysis reveals that $k \leq n$ holds for 99 % of the considered PB constraints. Comparing the two GAC encodings for the extreme case $k = n$, the SWC encodes at most $2n^2 + n$ clauses and the watchdog encoding generates $\mathcal{O}(n^3\, log(n)\, log(n))$. In this case, our encoding has a quadratic complexity and the watchdog encoding a ternary complexity. Only in the rare case where $k \geq n^3$ the watchdog encoding results in less clauses. Hence the novel encoding improves the state of the art.

In the following, we briefly discuss the SEQ encoding and define the SWC encoding in Section 3.1. In Section 3.2 we prove that the SWC encoding detects inconsistency and maintains generalized arc consistency by unit propagation.

3.1 From Sequential Counters to Sequential Weight Counters

Sequential Counters Setting all weights w_i in a PB constraint to 1 results in a *cardinality constraint* $\sum_i x_i \leq k$, allowing at most k variables to be assigned to 1, where $1 \leq i \leq n$. These constraints and their encodings into SAT instances are well studied (see e.g. [18,4]).

The idea of the SEQ encoding is to sequentially count from left to right the number of variables which have been assigned to 1 by the current interpretation J. This process can be encoded by circuits [18]. Each intermediate sum is encoded by a unary representation with the help of auxiliary variables $s_{i,j}$, $1 \leq i \leq n$ and $1 \leq j \leq k$, such that $s_{i,k}$ is the most significant digit. The value of j is used to represent the value of the ith sum. Intuitively, if and only if among the variables $x_1, x_2, \ldots, x_i$ of the PB constraint at least j variables are set to 1 by J, the variable $s_{i,j}$ should also be set to 1 by UP and the encoding. Therefore,

[1] `http://www.cril.univ-artois.fr/PB11/benchs/PB11-SMALLINT.tar`
`http://www.cril.univ-artois.fr/PB11/benchs/PB11-BIGINT.tar`
`http://www.cril.univ-artois.fr/PB10/benchs/PB10-selected-benchs.tar`

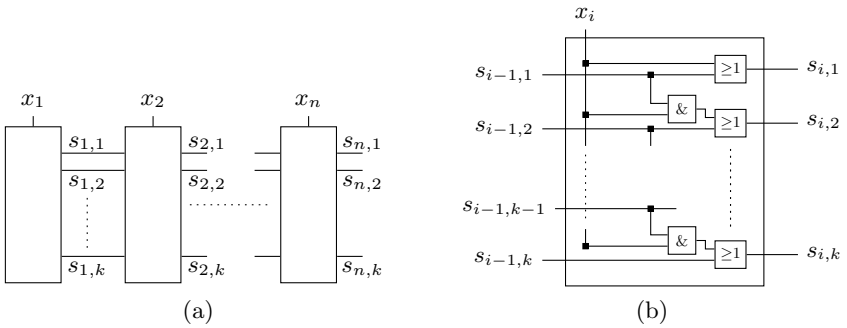

Fig. 1: SEQ encoding: (a) An overview over the whole circuit showing the connection of the input bits and output bits between the single circuits for each variable x_i. (b) The detailed circuit for a single input variable x_i.

we introduce a number $s^i_{\geq}$ for the ith sum that is defined as:

$$s^i_{\geq} := \begin{cases} j & s_{i,j} = 1 \wedge s_{i,j+1} \neq 1 \wedge j < k \\ k & s_{i,k} = 1 \\ 0 & else \end{cases}$$

Hence, $s^i_{\geq}$ represents the number of variables $x_1, x_2, \ldots, x_i$ which are assigned to 1 by J (see Fig. 1). As an example, consider the cardinality constraint $x_1 + x_2 + x_3 + x_4 \leq 3$, assume that $x_1 = x_3 = 1$ and $x_2 = x_4 = \mathit{undef}$, and suppose we are interested in the question how many of the first three variables are assigned to 1 by J. In this case, $s_{3,1} = s_{3,2} = 1$, $s_{3,3} = \mathit{undef}$, and $s^3_{\geq} = 2$.

The counting mechanism is illustrated in Fig.1(a) and can be implemented by gates: An OR gate in Fig.1(b) ensures that if the input bit $s_{i-1,j}$ is set to 1, then the output bit $s_{i,j}$ is set to 1 as well. Thus, for the two sums $s^i_{\geq}$ and $s^{i+1}_{\geq}$ in unary representation it holds: $s^i_{\geq} \leq s^{i+1}_{\geq}$. An output bit $s_{i,j}$ is also set to 1 if the input variable x_i and the previous input bit $s_{i-1,j-1}$ are set to 1. This behavior is ensured by the AND gate.

For the encoding of the circuit the Tseitin transformation [20] is used. Additionally a formula is added, disallowing the sum to become greater than k:

$$\begin{aligned} s_{i,1} &\Leftrightarrow x_i \vee s_{i-1,1} && \text{for } 1 \leq i \leq n, \\ s_{i,j} &\Leftrightarrow (x_i \wedge s_{i-1,j-1}) \vee s_{i-1,j} && \text{for } 1 \leq i \leq n,\ 1 < j \leq k, \\ \bot &\Leftrightarrow x_i \wedge s_{i-1,k}, \end{aligned}$$

where $\bot$ denotes a formula, which is always false. Because the OR and AND gates in the SEQ encoding occur only positively, only the $\Leftarrow$ directions are needed for the transformation into conjunctive normal form [16]. For more information about the SEQ encoding we refer to [18].

Sequential Weight Counters To extend the encoding of a cardinality constraint to an encoding of a PB constraint we replace the coefficients 1 by weights $1 \leq w_i \leq k$ for each variable x_i. If $J(x_i) = 1$ we have to set the output bits $s_{i,j+1}$, $s_{i,j+2}$,...,$s_{i,j+w_i}$ to 1, where j is the largest index with $s_{i-1,j} = 1$, thus we sum up the values of the weights w_i for each assigned variable $x_i = 1$ instead of counting the number of assigned variables $x_i = 1$. The new mechanism is achieved by modifying one input of the AND gates. The equivalence $s_{i,j} \Leftrightarrow (x_i \wedge s_{i-1,j-1}) \vee s_{i-1,j}$ of the SEQ encoding is replaced by

$$s_{i,j} \Leftrightarrow (x_i \wedge s_{i-1,j-w_i}) \vee s_{i-1,j}.$$

If $j - w_i \leq 0$ we can skip the AND gate and just use the OR gate with $s_{i,j} \Leftrightarrow (x_i \vee s_{i-1,j})$. Fig.2(b) illustrates this substitution. The connections of the counters remains unchanged as shown in Fig.2(a). The final modification is to force the sum to be smaller or equal to k:

$$\bot \Leftrightarrow x_i \wedge s_{i-1,k+1-w_i}.$$

Since we have to ensure that the sum is smaller or equal than k, we can drop the circuit for x_n and the gates of the actual sum, because the formula

$$\bot \Leftrightarrow x_n \wedge s_{n-1,k+1-w_n}$$

Already achieves this property. For a PB constraint $\sum_i w_i x_i \leq k$ with $1 \leq i \leq n$ and the Tseitin transformation the following formula encodes the constraint into SAT:

$$\overline{s_{i-1,j}} \vee s_{i,j} \qquad \text{for } 2 \leq i < n,\ 1 \leq j \leq k, \tag{1}$$

$$\overline{x_i} \vee s_{i,j} \qquad \text{for } 1 \leq i < n,\ 1 \leq j \leq w_i, \tag{2}$$

$$\overline{s_{i-1,j}} \vee \overline{x_i} \vee s_{i,j+w_i} \qquad \text{for } 2 \leq i < n,\ 1 \leq j \leq k - w_i, \tag{3}$$

$$\overline{s_{i-1,k+1-w_i}} \vee \overline{x_i} \qquad \text{for } 2 \leq i \leq n. \tag{4}$$

Hence, the SWC encoding requires $2nk - 4k + w_1 + n - 1$ clauses and $k\,(n-1)$ auxiliary variables. As shown in Fig.2(b) the structure of the encoding is simple to understand and the formula can be easily encoded. We will show that the SWC encoding correctly implements a PB constraint $\sum_{i=1}^{n} w_i x_i \leq k$, where $1 \leq w_i \leq k$ and $k \geq 1$, in Theorem 4 in the next section.

3.2 Properties of the SWC Encoding

In this section, we prove properties of the SWC encoding, i.e. we show that it allows to detect inconsistencies as well as it maintains generalized arc consistency by unit propagation.

Following [3], in a constraint $C \subseteq D_1 \times \cdots \times D_k$ on the variables $x_1, \ldots, x_k$ with domains $D_1, \ldots, D_k$, a variable x_i is *generalized arc consistent (GAC)* – also known as *hyper-arc consistent* – if for every $a \in D_i$ there exists a $d \in C$ such that $a = d[i]$, where $d[i]$ denotes the ith element of d. The constraint C is

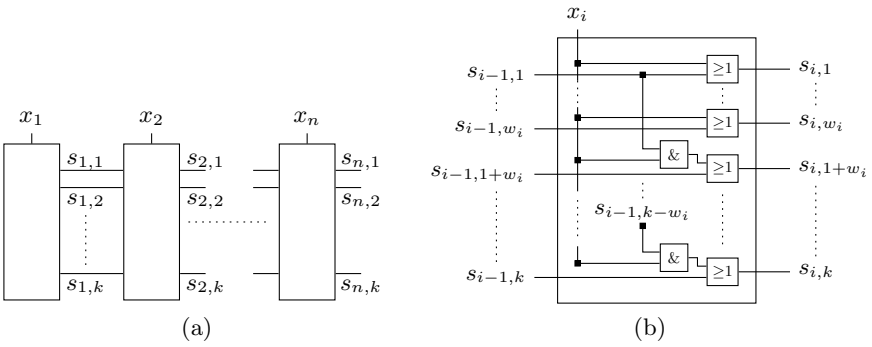

Fig. 2: SWC encoding: (a) Overview. (b) The detailed circuit for x_i

generalized arc consistent (GAC) if all variables x_j with $1 \leq j \leq n$ are GAC. One should observe that an inconsistent constraint C cannot be GAC because there does not exists any solution $d \in C$. In other words, if a constraint C is not GAC, then there exist an element a in the domain of some variable x_i which can be removed from the domain of x_i without removing any solution of C. It is beneficial to remove such unnecessary elements as soon as possible in order to prune the search space.

Returning to PB constraints, we recall that the domain of each variable occurring in a PB constraint of the form $\sum_i w_i x_i \leq k$ is initially the set $\{1, 0\}$. We define the minimum sum $min\_sum(C, J)$ of a PB constraint C with respect to an interpretation J as

$$min\_sum(C, J) = \sum \{w_i \mid J(x_i) = 1\},$$

where the sum of a finite set $\{e_1, e_2, \ldots, e_m\}$ of integers is $\sum_{i=1}^{m} e_i$.

We can now apply GAC to PB constraints: A variable x_i of a consistent PB constraint C of the form $\sum_i w_i x_i \leq k$, where $1 \leq w_i \leq k$, is GAC with respect to an interpretation J if

$$J(x_i) = 0 \text{ or } J(x_i) = 1 \text{ or } min\_sum(C, J) \leq k - w_i.$$

If a variable x_i is a already assigned, then this variable must be generalized arc consistent because, otherwise, the PB constraint would not be consistent. The third condition states that there exist a solution for the PB constraint where $x_i = 1$. Note that in a consistent PB constraint in the given form, assigning a variable x_i to 0 always leads to a consistent PB constraint. We can assign every variable x_i which does not meet one of the conditions to 0 until the constraint is GAC without loosing a solution for the constraint.

Now we can define the properties that a PB encoding into SAT should meet. Let $E(C)$ be an encoding of a PB constraint C into a SAT instance, where C is of the form $\sum_{i=1}^{n} w_i x_i \leq k$:

- $E(C)$ is said to *detect inconsistency by unit propagation* if the following holds: Whenever C is inconsistent with respect to an interpretation J, then UP detects a conflict in $E(C)$ with respect to J.
- $E(C)$ is said to *maintain generalized arc consistency by unit propagation* if the following holds: If C is not GAC with respect to an interpretation J, then $E(C), J \vdash^*_{up} E(C), J'$ such that for all variables x_i of C which are not GAC with respect to J we find $\overline{x} \in J'$.

Let J be an interpretation that maps arbitrary many variables x_i to truth values, but let all auxiliary variables unassigned: $J(s_{i,j}) = undef$. From now on every variable assignment is considered w.r.t. an interpretation J', where J' is achieved by UP: $E(C), J \vdash^*_{up} E(C), J'$ and $E(C)$ is the SWC encoding for the PB constraint C. This can be done w.l.o.g. because for each interpretation $J'' \supseteq J$ that satisfies $E(C)$, $J' \subseteq J''$ holds.

In the rest of this section we prove that the SWC encoding detects inconsistency and maintains GAC by UP.

Lemma 1. $\sum\{w_j \mid x_j = 1, 1 \leq j \leq i\} = s^i_{\geq}$

If we arbitrary assign the variables x_i to 1 or 0, $s^i_{\geq}$ is the value of the sum $\sum\{w_j \mid x_j = 1, 1 \leq j \leq i\}$. The clauses (1),(2) and (3) imply the auxiliary variables $s_{i,j}$ for every variable x_i in exactly that way.

Now we can prove that SWC detects consistency by UP:

Corollary 2. *The Sequential Weight Counter encoding detects inconsistency by UP.*

Proof. With lemma 1 and the clause $\overline{s_{i-1,k-w_i}} \vee \overline{x_i} \in E(C)$ *this follows directly, since* $\sum\{w_j \mid x_j = 1, 1 \leq j \leq n\} > k$ *implies that there exists a variable* $x_i = 1$ *with* $s^{i-1}_{\geq} + w_i \geq k$*, hence* $s_{i-1,k-w_i} = 1$.

In analogy to $s^i_{\geq}$ we define $s^i_{<}$, where $s^i_{\geq}$ is counting the sum from left to right, i.e.: $\sum_{a=1}^{i} w_a x_a$, and $s^i_{<}$ from right to left, i.e.: $\sum_{a=n}^{i+1} w_a x_a$.

$$s^i_{<} = \begin{cases} k - u + 1 & \text{where } u \text{ is the smallest number with } s_{i,u} = 0 \\ 0 & else \end{cases}$$

The auxiliary variable $s_{i,j}$ is set to 0 if and only if $\sum\{w_a \mid x_a = 1, i < j \leq n\} \geq k - j + 1$.

Lemma 3. $\sum\{w_j \mid x_j = 1, i < j \leq n\} = s^i_{<}$

Proof (sketch). We consider the sum $\sum\{w_j \mid x_j = 1, i < j \leq n\}$ *as a fixed sequence of addends* w_j *in descending order according to* j *. Now we can prove the lemma by induction, starting with the first addend* w_i *in the sum (i.e. there exists no* $x_l = 1$*, with* $l > i$*). With* $\overline{s_{i-1,k-w_i+1}} \vee \overline{x_i} \in E(C)$ *we get* $s^{j-1}_{<} = w_j = \sum\{w_j\}$*. For the induction step we show that for each* $x_a = 1$ *with* $a < i$ *we find a clause*

$\overline{s_{a-1,j}} \vee \overline{x_a} \vee s_{a,j+w_a} \in E(C)$ *such that* $s_{a,j+w_a} = 0$ *is the previous addend of the sum:*

$$s_<^{a-1} = w_a + s_<^a = w_a + \sum\{w_l \mid x_l = 1, a < l \leq n\}$$

For each $x_j \neq 1$ *it follows from the definitions of SWC that* $s_<^{i-1} = s_<^i$*, since* $\overline{s_{i-1,j}} \vee s_{i,j} \in E(C)$*.*

□

From lemma 1 and 3 follows that:

$$x_i \neq 1 \Rightarrow s_\geq^{i-1} = s_\geq^i \tag{5}$$

$$s_\geq^i + s_<^i = min\_sum(C, J) \tag{6}$$

Now we can prove that SWC is an encoding for the PB constraint and that the SWC encoding maintains GAC by UP.

Theorem 4. *The SWC is an encoding for the PB constraint* $\sum_{i=1}^n w_i x_i \leq k$ *in CNF, requiring* $\mathcal{O}(nk)$ *clauses and* $\mathcal{O}(nk)$ *auxiliary variables.*

Proof. From the corollary 2 we know that setting the variables x_i *such set the sum* $\sum_{i=1}^n w_i x_i > k$ *leads to an inconsistent formula by the encoding. Hence we only have to show that setting the variables* x_i *such that* $\sum_{i=1}^n w_i x_i \leq k$ *does not lead to a contradiction. Having only the clauses* (1),(2) *and* (3)*, it follows that any assignment of the variables* x_i *does not lead to a contradiction, since* $x_i = 1$ *only implies an auxiliary variable* $s_{i,j}$ *positively (i.e.* $s_{i,j} = 1$*) and in each of these clauses one* $s_{i,j}$ *occurs positively. Setting* $x_i = 0$ *results in no implication, since* x_i *does not occur positively in any clause. Similar to the proof of lemma 3, we can prove that the implications of the clause* (4) *lead to a contradiction if and only if* $\sum_{i=1}^n w_i x_i > k$*.*

□

Theorem 5. *The Sequential Weight Counter encoding maintains GAC by UP.*

Proof. Assume there is a variable x_i *that is not GAC with respect to* C*, hence* $x_i = \text{undef}$*. Since* x_i *is not GAC* $min\_sum(C, J) > k - w_i$ *holds (i.e. we cannot assign* $x_i = 1$*). With* (5) *and* (6) *we have:*

$$min\_sum(C, J) = s_\geq^i + s_<^i = s_\geq^{i-1} + s_<^i$$

Hence there exists a lower bound l *for the* i*th sum that represents the value of the* $i-1$*th sum (*$l = s_\geq^{i-1}$ *because* $s_{i-1,l} = 1$*) and an upper bound* u *for the* i*th sum* $s_{i,u} = 0$ *such that* $u = k - s_<^i + 1$*. If the difference between* l *and* u *is less equal than* w_i*,* x_i *needs to be set to* $x_i = 0$*:*

$$\begin{aligned} s_\geq^{i-1} + s_<^i > k - w_i \Leftrightarrow \\ l + k - u + 1 > k - w_i \Leftrightarrow \\ l + w_i + 1 > u \end{aligned} \tag{7}$$

case $u \leq w_i$
with $\overline{x_i} \vee s_{i,u} \in E(C)$ *this directly contradict our assumption.*

case $u > w_i$
with (7) *we know that* $l \geq 1$ *and there exists a* $j \leq l$ *with* $j + w_i = u$. *With* $\overline{s_{i-1,j}} \vee \overline{x_i} \vee s_{i,j+w_i} \in E(C)$ *this contradict our assumption.*

□

4 Results

In this section we want to show the usefulness of the proposed sequential weight counter (SWC) encoding. The first advantage of the encoding is its simple structure. When a PB constraint should be translated to SAT by a BDDs, the algorithm is more complicated [9,2].

As a basis for the experiments we use all decision PB instances of PB competitions 2010 and 2011[1]. Note, that from the *big int* PB instances none of the selected solving methods can solve a single instance within the timeout. Therefore, we decided to drop these instances from the benchmark again. In total, there are 278 PB instances in the benchmark. The experiments have been performed on an AMD Opteron CPU with 2.66 GHz, a memory limit of 2 GB and a timeout of 1800 s.

Before all the single constraints are translated into SAT, we simplified them accordingly. For a constraint $\sum_i w_i x_i \leq k$ we immediately assign x_i to 0, if $w_i > k$. Furthermore, all constraints with $\sum_i w_i \leq k$ are removed. Constraints of the form $\sum_i l_i \geq 1$ are encoded as a single clause. Finally, $\sum_i l_i \leq k$ is translated by an appropriate cardinality constraint encoding [4]. We have not used the watchdog encoding for several reasons: (i) this encoding almost always produces more clauses than the SWC encoding, (ii) the encoding is highly complex to be implemented and (iii) using the tool that has been used in [5] would also encode all special PBs with the watchdog encoding.

To compare the impact of the novel encoding, we translated all PB instances into SAT and solved them with the SAT solver GLUCOSE 2 because of its high performance in recent SAT competitions[2]. Table 2 compares the number of solved PB instances among the encodings and gives the average time that has been used to solve a single instance. Encoding PB constraints with BDD has been done according to [9]. We furthermore added the configuration BEST, that selects for each PB the encoding that produces the least number of clauses. By fixing the encoding, both BDD and SWC solve already a high number of instances. However, there is no clear benefit for either of the two encodings. SWC can solve exactly the same instances as by using BDD and another three instances more. For 58 instances of the 126 commonly solved instances, BDD can solve the instance faster whereas for the remaining 68 instances SWC returns an answer more quickly. As already seen in other fields, a portfolio approach

[2] We provide the tool at `http://tools.computational-logic.org`.

Table 2: Comparing the performance of PB solving approaches

Encoding	BDD	SWC	BEST	BSOLO	CLASP
Solved instances	126	129	**141**	98	120
Run time	180.49 s	193.74 s	142.77 s	136.43 s	138.08 s

increases the performance of solvers [21]. By choosing always the best encoding the configuration BEST solves another 12 instances and also decreases the run time per instance. Thus, for the translation to SAT the SWC encoding provides a clear benefit.

Since PB can be solved also natively or by handling PB constraints inside a SAT solver, we furthermore compare our approach with successful systems of the last PB competition. BSOLO is a native PB solver [13] and CLASP [1] is a SAT solver that can handle PB constraints inside the solver without a translation to SAT. These solvers are also compared to the translation to SAT in Table 2. Again, the configuration BEST solves 21 more instances then the best of the native solvers, and solves all the instances that have been solved by the native solvers. Summarizing the evaluation it can be stated that adding the SWC encoding to the portfolio of available PB encodings results in a noticeable performance improvement for PB solvers.

5 Conclusion and Future Work

In this work we presented the SWC encoding, a new encoding for PB constraints of the form $\sum_{i=1}^{n} w_i x_i \leq k$ into SAT. The SWC encoding allows unit propagation to quickly prune the search space by maintaining GAC and needs at most $n\,(2k+1)$ clauses and $(n-1)k$ auxiliary variables. This is a significant improvement to the state of the art for PB constraints with $k \leq n^2$: To the best of our knowledge the local watchdog encoding generates the fewest clauses, namely $\mathcal{O}(n^3\, log(n)\, log(k))$ clauses, while maintaining GAC. This contribution is highly relevant, because for 99 % of the PB constraints even $k \leq n$ holds.

The new encoding is not only a nice and simple encoding, but also provides a performance improvement for solving PB instances. By always choosing the encoding that requires the smallest number of clauses, our PB solver can solve 12 instances more than by forcing to use a single encoding only. With our approach 21 more instances of the PB benchmark can be solved compared to successful solvers from recent PB competitions.

For future work we leave a detailed comparison between the known encodings, the SWC, binary decision diagrams, local watchdog and the non-GAC encodings. With the help of a detailed empirical investigation we want to extend our current research to a competitive SAT-based PB solver that can also solve optimization instances fast.

References

1. Potsdam answer set solving collection, `http://potassco.sourceforge.net/`
2. Abío, I., Nieuwenhuis, R., Oliveras, A., Rodríguez-Carbonell, E.: BDDs for pseudo-boolean constraints: revisited. In: Proc. SAT. pp. 61–75 (2011)
3. Apt, K.: Principles of Constraint Programming. Cambridge University Press (2003)
4. Asín, R., Nieuwenhuis, R., Oliveras, A., Rodríguez-Carbonell, E.: Cardinality networks and their applications. In: Proc. SAT. pp. 167–180 (2009)
5. Bailleux, O., Boufkhad, Y., Roussel, O.: New encodings of pseudo-boolean constraints into cnf. In: Proc. SAT. pp. 181–194 (2009)
6. Biere, A., Cimatti, A., Clarke, E.M., Fujita, M., Zhu, Y.: Symbolic model checking using sat procedures instead of BDDs. In: Proc. DAC. pp. 317–320 (1999)
7. Davis, M., Logemann, G., Loveland, D.: A machine program for theorem-proving. Commun. ACM 5(7), 394–397 (Jul 1962)
8. Eén, N., Biere, A.: Effective preprocessing in sat through variable and clause elimination. In: Proc. SAT. pp. 61–75 (2005)
9. Eén, N., Sörensson, N.: Translating pseudo-boolean constraints into sat. JSAT 2(1-4), 1–26 (2006)
10. Eibach, T., Pilz, E., Völkel, G.: Attacking Bivium using SAT solvers. In: Proc. SAT. pp. 63–76 (2008)
11. Großmann, P., Hölldobler, S., Manthey, N., Nachtigall, K., Opitz, J., Steinke, P.: Solving Periodic Event Scheduling Problems with SAT. In: IEAAIE, to appear in (2012)
12. Kaiss, D., Skaba, M., Hanna, Z., Khasidashvili, Z.: Industrial strength sat-based alignability algorithm for hardware equivalence verification. In: Proc. FMCAD. pp. 20–26 (2007)
13. Manquinho, V.M., Silva, J.P.M.: On using cutting planes in pseudo-boolean optimization. JSAT 2(1-4), 209–219 (2006)
14. Marques-Silva, J.a.P., Sakallah, K.A.: Grasp: A search algorithm for propositional satisfiability. IEEE Trans. Comput. 48(5), 506–521 (May 1999)
15. Moskewicz, M.W., Madigan, C.F., Zhao, Y., Zhang, L., Malik, S.: Chaff: engineering an efficient SAT solver. In: Proc. DAC. pp. 530–535 (2001)
16. Plaisted, D.A., Greenbaum, S.: A structure-preserving clause form translation. J. Symb. Comput. 2(3), 293–304 (Sep 1986)
17. Roussel, O., Manquinho, V.: Pseudo-Boolean and Cardinality Constraints, Frontiers in Artificial Intelligence and Applications, vol. 185, chap. 22, pp. 695–733. IOS Press (February 2009)
18. Sinz, C.: Towards an optimal CNF encoding of Boolean cardinality constraints. In: Proc. CP. pp. 827–831 (Oct 2005)
19. Sörensson, N., Biere, A.: Minimizing learned clauses. In: Proc. SAT. pp. 237–243 (2009)
20. Tseitin, G.S.: On the complexity of derivation in propositional calculus. Studies in constructive mathematics and mathematical logic 2(115-125), 1013 (1968)
21. Xu, L., Hutter, F., Hoos, H.H., Leyton-Brown, K.: Satzilla: portfolio-based algorithm selection for SAT. J. Artif. Int. Res. 32(1), 565–606 (Jun 2008)