New Encodings of Pseudo-Boolean Constraints into CNF

Olivier Bailleux[1], Yacine Boufkhad[2], and Olivier Roussel[3]

[1] LERSIA – Université de Bourgogne
olivier.bailleux@u-bourgogne.fr
[2] LIAFA, CNRS, Gang team INRIA, Université Paris Diderot, France
boufkhad@liafa.jussieu.fr
[3] Université Lille-Nord de France, Artois, F-62307 Lens – CRIL, F-62307 Lens – CNRS UMR 8188, F-62307 Lens
olivier.roussel@cril.univ-artois.fr

Abstract. This paper answers affirmatively the open question of the existence of a polynomial size CNF encoding of pseudo-Boolean (PB) constraints such that generalized arc consistency (GAC) is maintained through unit propagation (UP). All previous encodings of PB constraints either did not allow UP to maintain GAC, or were of exponential size in the worst case. This paper presents an encoding that realizes both of the desired properties. From a theoretical point of view, this narrows the gap between the expressive power of clauses and the one of pseudo-Boolean constraints.

Keywords: Pseudo-Boolean, SAT translation.

1 Introduction

Many practical problems can be expressed as constraint satisfaction problems and several formalisms for constraint satisfaction have been defined: integer linear programming (ILP) [7], constraint satisfaction problems (CSP), pseudo-Boolean constraints (PB) [2], propositional satisfiability (SAT) and Quantified Boolean Formulae (QBF) to name only a few. These formalisms differ by the expressivity of their constraints, the power of the inference rules that can be used and the efficiency of the corresponding solvers.

Complexity theory is a first approach to compare these formalisms. For instance, QBF has a higher complexity than SAT, PB, CSP and ILP. Both SAT, PB, CSP[1] and ILP are NP-complete problems. This implies that, from the point of view of complexity theory, all these formalisms have the same expressive power in the sense that there exist polynomial reductions from any of these problems to another.

The existence of a polynomial reduction from a formalism F to another formalism F' means that a problem of F can always be solved by translating it into

[1] Under the assumption that each constraint of the CSP can be checked in polynomial time.

O. Kullmann (Ed.): SAT 2009, LNCS 5584, pp. 181–194, 2009.

a problem of F', and that such an indirect approach only implies a polynomial overhead which can be considered as negligible for NP problems. This approach can also be quite interesting in practice if solvers for F' are more efficient than solvers for F. The efficiency of modern SAT solvers explains why SAT has become very popular to encode and solve a number of problems. Some examples are Model checking [6], Symbolic reachability [1], Planning [8], Scheduling [12], Diagnosis [10], etc.

There may exist many encodings of a given problem in SAT and these encodings are not equivalent regarding SAT solving methods. For example, in [11] it is shown that a problem can be encoded to SAT in two ways, one that is exponential for resolution and another which is polynomial although the two encodings are of polynomial size. Similarly in [4], some difficult benchmarks for SAT solvers are shown to be easy if encoded in a different way. Thus, to make the polynomial reduction to SAT a practical approach for solving problems, the question is to find a polynomial reduction that preserves the basic inferences in a given problem through the basic inferences used in SAT solvers. More precisely, this paper focuses on Unit Propagation as the basic mechanism of inference in complete SAT solvers.

For example, it is known that maintaining Arc Consistency on CSP instances defined with only constraints in extension is equivalent to applying Unit Propagation on a polynomial SAT encoding of the constraints named "support encoding" [9]. Another encoding with the same property is proposed in [3]. These encodings show that SAT and CSP with only extensional constraints are very close problems. The same approach is used in this paper to compare the pseudo-Boolean formalism with the Satisfiability formalism.

Pseudo-Boolean constraints can be seen as an extension of the clauses of SAT, or as a special case of integer programming constraints. The following observations suggest that pseudo-Boolean constraints are stronger than SAT constraints (clauses). While clauses can be considered as constraints defined in extension (a clause forbids one single partial assignment), pseudo-Boolean constraints are defined in intention through a mathematical formula. Another observation is that a SAT encoding of a PB constraint which doesn't introduce extra variables requires an exponential number of clauses in the worst case. Besides, so far, all known SAT encodings of a PB constraint were either of exponential size or did not allow unit propagation to maintain GAC. This article presents a polynomial SAT encoding of PB constraints which proves that the basic inferences on PB constraints (Generalized Arc Consistency) polynomially reduce to Unit Propagation in SAT. To the best of our knowledge, the existence of such an encoding was an open question. This result narrows the gap between the PB and SAT formalisms.

Section 2 introduces the definitions and notations used throughout this paper. The general idea of the new encoding is to sum the terms on the left side of the PB constraint and compare it to the right term. The property required for the encoding of the addition is that it must give a result even when some input variables are unassigned. To reach this goal, the encoding is based on a unary

representation of numbers (see section 3). For conciseness, coefficients in the constraints are decomposed in binary, and a unary representation is used for each power of two. A kind of carry between unary representations ensures that the most significant unary representation computes the sum of the coefficients of the true literals in the constraint. The key point in the encoding is to add a constant to each side of the PB constraint so that the right term becomes a multiple of a power of two, which makes the comparator trivial.

This general encoding is detailed in section 4.1. It can be declined in two different versions: GPW that detects inconsistencies (section 4.2) and LPW that maintains Generalized Arc Consistency (section 4.3). LPW is the first SAT encoding of a pseudo-Boolean constraint of n variables with a maximum coefficient of a_{max} which is both polynomial ($O(n^2 \log(n) \log(a_{max}))$ variables and $O(n^3 \log(n) \log(a_{max}))$ clauses) and which lets Unit Propagation maintain Generalized Arc Consistency. Section 5 compares the new method with the previous encodings of PB constraints into SAT. At last, some perspectives are given.

2 Definitions and Notations

In this paper, we only consider constraints which are defined over a finite set of Boolean variables x_j. Boolean variables can take only two values 0 (*false*) and 1 (*true*). A literal l_j is either a Boolean variable x_j or its negation $\overline{x_j}$ (with $\overline{x_j} = 1 - x_j$). A *linear pseudo-Boolean constraint* is a constraint over Boolean variables defined by $\sum_j a_j l_j \triangleright M$ where a_j and M are integer constants, l_j are literals and $\triangleright$ is one of the classical relational operators ($=, >, \geq, <$ or $\leq$). Without loss of generality, these constraints can be rewritten to use only the less operator and positive coefficients a_j (since $-a.x$ can be rewritten as $a.\overline{x} - a$). A *clause* is a disjunction of literals. A clause $l_1 \vee l_2 \vee \ldots \vee l_n$ is equivalent to $l_1 + l_2 + \ldots + l_n \geq 1$. So clauses are a special case of pseudo-Boolean constraints where each $a_j = 1$ and $M = 1$. An assignment is a mapping of Boolean variables to their value (0 or 1). In the CSP context, an instantiation is a mapping of variables to a value in their domain.

Unit propagation (UP) is the fundamental mechanism used in most SAT solvers. Whenever each literal of a clause but one is false, the remaining literal must be set to true in order to satisfy the clause. The derivation of a literal l by unit propagation from formula f will be denoted $f \vdash_{UP} l$. The derivation of the empty clause is denoted $f \vdash_{UP} \bot$. *Generalized Arc Consistency* (GAC) is one of the fundamental inference rules in CSP. Let $scp(C)$ denote the scope of a constraint C, which is the set of variables constrained by C. A value a of a variable X is generalized arc consistent in a constraint C if $X \notin scp(C)$ or, when $X \in scp(C)$, if there exists an instantiation I of the other variables in the scope of C such that $I \cup \{X = a\}$ satisfies C. A value a of X is generalized arc consistent if it is generalized arc consistent in every constraint. A CSP instance is generalized arc consistent if each value of each variable is generalized arc consistent. Enforcing GAC consists in removing values which are not generalized arc consistent from the domain of their variable. For example, let us consider the

PB constraint $x_1 + 2x_2 + 4x_3 < 3$, value 1 for x_3 is not generalized arc consistent because no assignment of x_1, x_2 can satisfy the constraint once $x_3 = 1$. Enforcing GAC on this constraint will remove 1 from the domain of x_3 and hence assign 0 to x_3 .

An *encoding* E of a source language L_S to a target language L_T is a mapping of constraints of L_S to sets of constraints of L_T such that any formula f of L_S is equivalent to $E(f)$ in a generalized sense: any model of f can be extended to obtain a model of $E(f)$ (and conversely any model of $E(f)$ can be projected on the vocabulary of f to get a model of f). In the following definitions, we only consider languages where variables are Boolean[2]. An encoding E is said to *UP-detect inconsistency* if, for any constraint C of the source language and any assignment A of the source language, $C \wedge A \models \bot \Leftrightarrow E(C) \wedge A \vdash_{UP} \bot$. An encoding E is said to *UP-maintain GAC* if, for any constraint C, any assignment A and any literal l of the source language, $C \wedge A \models l \Leftrightarrow E(C) \wedge A \vdash_{UP} l$.

For example, let us consider the pseudo-Boolean constraint $x_1 + 2x_2 + 4x_3 < 6$. Given the partial assignment $\{x_1 = 1, x_3 = 1\}$, any encoding which UP-maintains GAC will allow unit propagation to fix $x_2 = 0$. Given the partial assignment $\{x_2 = 1, x_3 = 1\}$, any encoding which UP-detects inconsistency must allow unit propagation to produce the empty clause.

Obviously, all things being equal, the more a solver propagates, the more efficient it is. On the other hand, encodings which UP-maintain GAC generally produce larger formulae than the other ones because they must encode each potential implication of a literal. Of course, larger formulae slow down unit-propagation. It is then not always clear which is the best trade-off between the size of encodings and their ability to enforce propagations.

3 Unary Representation and Cardinality Constraints

First, let us recall briefly the notion of unary representation of integer intervals. The details are in [4].

An integer variable u taking its values in the range $0..k$ is represented by a vector of k Boolean variables $U = \langle u_1, ..., u_k \rangle$. At any time, only variables on the left of this vector can be assigned 1, only variables on the right can be assigned 0 and variables in between are unassigned. More formally, U takes its values in a set $\mathcal{U}_k \subset \{0, 1, *\}^k$ ($*$ standing for unassigned) such that there exists two ranks a and b ($0 \leq a \leq b \leq k$) having the following property: $u_i = 1$ if $i \leq a, u_i = *$ if $a < i \leq b$ and $u_i = 0$ if $i > b$.

An integer u such that $u = m$ is represented by the vector having $u_1 = u_2 = ... = u_m = 1$ and $u_{m+1} = ... = u_k = 0$. The advantage of unary vectors is that they allow the representation of integer intervals. For example $a \leq u \leq b$ is represented by a vector that assigns 1 to the a first Boolean variables and 0 to the $k - b$ last ones, the remaining variables being unassigned.

[2] However, the generalization to languages where a variable X can take multiple values v_i is straightforward. For example, an encoding E UP-maintains GAC if, $\forall C, \forall A, \forall X, \forall v$, $C \wedge A \models X \neq v \Leftrightarrow E(C) \wedge E(A) \vdash_{UP} E(X \neq v)$.

The other advantage of this representation is that it allows to encode an addition in such a way that unit propagation is able to do the expected inferences, even when some variables are unassigned. Let U and V be two unary vectors representing respectively two integers u and v and let W be the unary representation of their sum $w = u + v$. The encoding of this addition contains clauses of the type $\overline{u_a} \vee \overline{v_b} \vee w_{a+b}$ stating that whenever $u \geq a$ and $v \geq b$ for some values a and b then $w \geq a + b$. We will denote by $\psi(U \oplus V = W)$ the conjunction of all the clauses of that type that ensure that $w \geq u + v$ through their unary representations. The sum of integers is naturally extended to the sum of their representations through the operator $\oplus$. Formally, for two unary vectors $U = \langle u_1, u_2, .., u_k \rangle$, $V = \langle v_1, v_2, .., v_l \rangle$ and $W = U \oplus V = \langle w_1, w_2, .., w_{k+l} \rangle$ with the convention $u_0 = v_0 = w_0 = 1$:

$$\psi\left(W = U \oplus V\right) = \bigwedge_{a=0}^{k} \bigwedge_{b=0}^{l} (\overline{u_a} \vee \overline{v_b} \vee w_{a+b})$$

Some other clauses ensuring that $w \leq u + w$ are needed to obtain the encoding of [4] but are omitted because they are not relevant in this paper. Clearly the number of clauses in $\psi\left(W = U \oplus V\right)$ is $O(n^2)$ when the numbers are of size n.

In [4], the unary representation is used to efficiently encode cardinality constraints. The vector of variables involved in the cardinality constraint called input variables is connected to a unary vector called output vector representing its number of 1s through a CNF formula called a Totalizer. The Totalizer is in charge of transforming the input vector in an output vector which contains the same values but which also satisfies the requirements of the unary representation (all 1s on the left, all 0s on the right and all unassigned variables in the middle). In essence, this Totalizer plays the same role as a sorting network.

The Totalizer used in the encoding schemes described in this paper is simpler than the one used in [4] because we never use the 0s in the output. All is needed is that the Totalizer generate an output vector with all 1s on the left (as many as in the input vector) and all other variables unassigned. For any vector X of Boolean variables, let $U(X)$ denote the vector of the unary representation of the number of its 1s as enforced by the Totalizer. Let $\Phi(X)$ be the Totalizer which transforms X into $U(X)$. It is built in a recursive manner as described in the recursive procedure $\Phi(X)$ of Algorithm 1. Indeed, the unary representation of a single variable is the variable itself, the unary representation of the number of 1s of vector $X = \langle x_1, x_2, ..., x_n \rangle$ is the sum of the unary representations of $X_1 = \langle x_1, x_2, ..., x_{\lfloor n/2 \rfloor} \rangle$ and $X_2 = \langle x_{\lfloor n/2 \rfloor + 1}, ...x_n \rangle$, and the CNF formula enforcing this conjunction is $\psi(U(X_1) \oplus U(X_2) = U(X))$. The whole formula of the totalizer of some vector X is denoted by $\Phi(X)$. It is the conjunction of the formulae ψ.

The fundamental property of the formula $\Phi(X)$ as it will be used later is that for any partial assignment to the variables X unit propagation enforces $U(X)$ to be the unary representation of the number of ones in X. The number of variables created by the encoding is clearly $O(n \log(n))$ and the number of clauses is $O(n^2 \log(n))$ since the procedure makes $O(log(n))$ recursive calls.

Algorithm 1. $\Phi(X)$

Require: A vector $X = \langle x_1, x_2, ..., x_k \rangle$
  if $k = 1$ **then**
    $U(X) \leftarrow \langle x_1 \rangle$
    return true
  else
    $U(X) \leftarrow \langle u_1, u_2, ..., u_k \rangle$ {u_i are obtained from a global unique variable generator}

    $X_1 \leftarrow \langle x_1, x_2, ..., x_{\lfloor n/2 \rfloor} \rangle$
    $X_2 \leftarrow \langle x_{\lfloor n/2 \rfloor + 1}, ...x_n \rangle$
    return $\Phi(X_1) \wedge \Phi(X_2) \wedge \psi(U(X_1) \oplus U(X_2) = U(X))$
  end if

In the rest of the description of the encoding, it is necessary to define an operator $\frac{1}{2}$ on the vectors of unary representations of integers such that for some vector $W = \langle w_0, w_1, ..., w_{2i}, w_{2i+1}, ... \rangle$, $W^{\frac{1}{2}} = \langle w_1, w_3, ..., w_{2i+1} ... \rangle$ is the vector of variables of odd ranks in the original. Clearly if W is the unary representation of some integer w then $W^{\frac{1}{2}}$ is the unary representation of $\lfloor \frac{w}{2} \rfloor$.

4 Global and Local Polynomial Watchdog Encoding Schemes

We present in this section two SAT encoding schemes LPW and GPW standing respectively for *Local Polynomial Watchdog* and *Global Polynomial Watchdog*. A watchdog is a formula which will set a Boolean variable to 1 as soon as a constraint gets falsified. LPW UP-maintains GAC while producing formulae of polynomial size. GPW, which UP-detects inconsistencies, is more of practical interest since it produces smaller formulae at the cost of losing the property of UP-maintaining GAC.

4.1 Polynomial Watchdog

In the following we will consider without loss of generality a unique constraint C defined by the sequence of positive integer coefficients $(a_i)_{i=1..n}$ and an integer M constraining $\sum_{i \in I} a_i x_i < M$ where $I = \{1, 2, ..., n\}$ is a set of indices ranging from 1 to n the number of Boolean variables involved in this constraint. We consider only non trivial constraints i.e $\sum_{i \in I} a_i > M$. For some integer a, let $b_j(a)$ be the value of the bit of rank j in the binary representation of a.

A polynomial watchdog (PW) associated with a constraint C is a CNF formula denoted by $PW(C)$ based on the following sets of variables: the input variables $\{x_i | i \in I\}$ of the constraint C, and a set of additional variables called encoding variables. $PW(C)$ has a single output variable ω. The formula $PW(C)$ is built in such a way that it has the following property: for every partial assignment to the input variables that violates the constraint C, unit propagation applied to $PW(C)$ assigns the value 1 to the output variable ω.

The idea used to construct $PW(C)$ is to decompose each coefficient of the constraint in its binary representation and sum each bit having the same weight 2^k in a single Totalizer. There are as many Totalizers as the number of bits of the greatest coefficient. Half of the value of the totalizer for weight 2^k is computed with operator $\frac{1}{2}$ and integrated in the totalizer for weight 2^{k+1} (this is a kind of carry). The value represented by the different Totalizers must be compared to M. To make this comparison trivial, the constraint is first rewritten so that the right term becomes a multiple of the weight of the last Totalizer. Once this is done, the value of the comparison is represented by one single bit of the last Totalizer. All computations can be performed by unit propagation, even when some input variables are unassigned.

Let us now detail how the formula $PW(C)$ is built. The binary representation of the coefficients is considered. Let p be the index of the most significant bit in the greatest a_i. In other words, p is the integer such that $p+1$ is the number of bits necessary to represent the largest coefficient, namely $p = \lfloor \log_2(max_{i=1..n}(a_i)) \rfloor$.

An important feature used by the Polynomial Watchdog encoding is the tare which is an integer denoted by T. It turns out that the comparison with the right side of the constraint is trivial when it is a multiple of 2^p. For this reason, we define the tare T as the smallest integer such that $M + T$ is a multiple of 2^p. Let m denote the integer such that $M + T = m2^p$ and let $t_{p-1}...t_1t_0$ denote the binary representation of T over $p-1$ bits ($T < 2^p$). Considering this, the constraint can be rewritten to an equivalent form $T + \sum_{i \in I} a_i x_i < m2^p$.

For every j such that $0 \leq j \leq p$, let B_j be the set containing the input variables with the bit of rank j equal to 1 in the binary representation of their coefficient plus the constant t_j (the jth bit of the tare) if $t_j = 1$. More formally $B_j = \{x_i | b_j(a_i) = 1\} \cup \{t_j \text{ if } t_j = 1\}$ for $0 \leq j \leq p$. The sets B_j are called buckets.

Example 1. For the constraint $2x_1 + 3x_2 + 5x_3 + 7x_4 < 11$, we have $p = 2$, $T = 1$ ($t_0 = 1$, $t_1 = 0$) and the buckets are $B_0 = \{1, x_2, x_3, x_4\}$, $B_1 = \{x_1, x_2, x_4\}$ and $B_2 = \{x_3, x_4\}$. Figure 1 represents the different buckets and generated circuits.

The formula $PW(C)$ is built recursively by cascading $p+1$ Totalizers (see Section 3). Let $PW_j(C)$ denote the Totalizer number j and S_j denote its output. The CNF encoding of the totalizers and their inputs are defined recursively as follows. $\langle B_j \rangle$ is a vector formed by the elements of the bucket B_j taken in an arbitrary order:

- For $j = 0$, let $PW_0(C) = \Phi(\langle B_0 \rangle)$. The output is $S_0 = U(\langle B_0 \rangle)$.
- For any $1 \leq j \leq p$, $PW_j(C) = \Phi(\langle B_j \rangle) \wedge \psi(S_j = U(\langle B_j \rangle) \oplus S_{j-1}^{\frac{1}{2}})$. The output unary vector is $S_j = U(\langle B_j \rangle) \oplus S_{j-1}^{\frac{1}{2}}$ enforced through the formula $\psi(S_j = U(\langle B_j \rangle) \oplus S_{j-1}^{\frac{1}{2}})$.

The polynomial watchdog of the constraint C can now be defined as: $PW(C) = \bigwedge_{j=0}^{p} PW_j(C)$ The mth variable of the vector S_p is the output variable ω (S_p has at least m bits because the constraint is not trivial). The algorithm 2 describes the steps in the computation of the formula $PW(C)$ and Figure 1 shows $PW(C)$ on the constraint of Example 1.

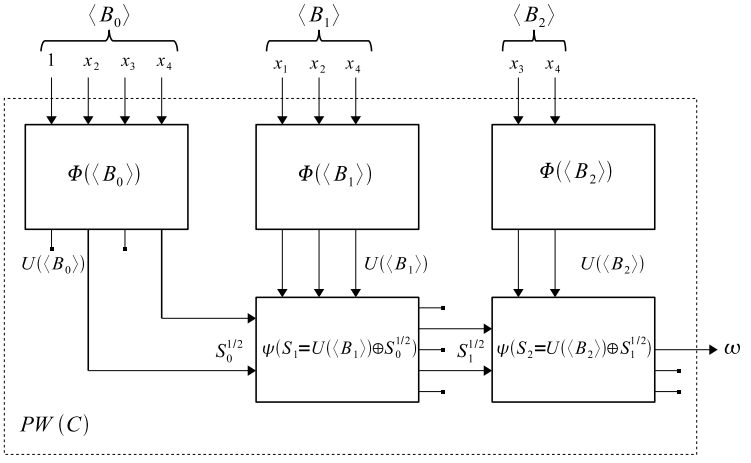

Fig. 1. Schematic representation of PW($2x_1 + 3x_2 + 5x_3 + 7x_4 < 11$)

Algorithm 2. PW(C)

Require: a constraint $\sum_{i=1}^{n} a_i x_i < M$
  $p \leftarrow \log_2(max_{i=1..n}(a_i))$
  $T \leftarrow (m2^p - M)$ s.t. m is the smallest integer having $m2^p \geq M$
  for $j = 0$ to p **do**
    $B_j \leftarrow \{x_i | b_j(a_i) = 1\} \cup \{t_j \text{ if } t_j = 1\}$
  end for
  $F \leftarrow \Phi(\langle B_0 \rangle)$
  $S_0 \leftarrow U(\langle B_0 \rangle)$
  for $j = 1$ to p **do**
    $F \leftarrow F \wedge \Phi(\langle B_j \rangle)$
    $S_j \leftarrow U(\langle B_j \rangle) \oplus S_{j-1}^{\frac{1}{2}}$
    $F \leftarrow F \wedge \psi(S_j = U(\langle B_j \rangle) \oplus S_{j-1}^{\frac{1}{2}})$
  end for
  return F

A polynomial watchdog is based on $p+1$ Totalizers requiring each $O(n \log(n))$ variables and $O(n^2 \log(n))$ clauses. Then, in general, a constraint involving n Boolean variables and having coefficients of at most a_{max} generates at most $O(n \log(n) \log(a_{max}))$ variables and $O(n^2 \log(n) \log(a_{max}))$ clauses.

Lemma 1. *For any partial assignment to the variables of C, Unit Propagation on $PW(C)$ assigns 1 to ω if and only if this partial assignment is inconsistent with C.*

Proof. Let s_i be the number of 1s in bucket B_i whose unary representation is $U(\langle B_i \rangle)$. The lemma follows from the fact that UP enforces at any time S_i for $0 \leq i \leq p$ to be the unary representation of $\lfloor \frac{\sum_{j=0}^{i} s_j 2^j}{2^i} \rfloor$. In particular, for p,

since $\lfloor \frac{\sum_{j=0}^{p} s_j 2^j}{2^p} \rfloor = \lfloor \frac{T+\sum_{j=0}^{p} a_j x_j}{2^p} \rfloor$, UP assigns 1 to the mth bit ω if and only if the left side is greater or equal to $M+T$ and hence the constraint is violated.

The fact that S_i for $0 \leq i \leq p$ is the unary representation of $\lfloor \frac{\sum_{j=0}^{i} s_j 2^j}{2^i} \rfloor$ can be proven by induction on i. For $i = 0$, it is obviously true thanks to the first Totalizer. Suppose now that the property is true for some i i.e. it is true that UP enforces S_i to be the unary representation of $\lfloor \frac{\sum_{j=0}^{i} s_j 2^j}{2^i} \rfloor$, since $S_{i+1} = U(\langle B_{i+1} \rangle) \oplus S_i^{\frac{1}{2}}$, UP will enforce — through the Totalizer $\Phi(\langle B_{i+1} \rangle)$ and $\psi(S_{i+1} = U(\langle B_{i+1} \rangle) \oplus S_i^{\frac{1}{2}})$ — S_{i+1} to be the unary representation of $s_{i+1} + \lfloor \frac{\lfloor \frac{\sum_{j=0}^{i} s_j 2^j}{2^i} \rfloor}{2} \rfloor = s_{i+1} + \lfloor \frac{\sum_{j=0}^{i} s_j 2^j}{2^{i+1}} \rfloor = \lfloor \frac{\sum_{j=0}^{i+1} s_j 2^j}{2^{i+1}} \rfloor$.

4.2 Global Polynomial Watchdog

The Global Polynomial Watchdog (GPW) is an encoding that detects inconsistencies. It is based on the PW described in the previous section. It consists simply in adding the unit clause $\overline{w}$. The formula generated to encode a constraint C is then $GPW(C) = PW(C) \wedge \overline{w}$. GPW has the same complexity than $PW(C)$.

Proposition 1. *A partial assignment of the variables of a constraint C is inconsistent with it if and only if unit propagation applied to $GPW(C)$ detects an inconsistency.*

However GPW does not UP-maintain GAC. This can be seen in the following counterexample. GAC on the partial assignment $x_3 = 1$ on Example 1 assigns $x_4 = 0$ but UP will not detect such an assignment. Indeed, all what UP will derive is that the higher bits of $U(\langle B_2 \rangle)$ and $S_1^{\frac{1}{2}}$ cannot be equal to 1 at the same time. This situation is obtained if $x_4 = 1$ but UP cannot foresee it.

Although GPW does not UP-maintain GAC it is not far from doing it. Indeed, GAC can be maintained through UP look-ahead. A UP look-ahead consists in trying to assign 1 to an unassigned input variable and then to run UP. If an inconsistency is detected, the variable must be assigned 0 otherwise it remains unassigned. GAC can be maintained through UP look-ahead on the GPW encoding but few modern SAT solvers perform such tests.

4.3 Local Polynomial Watchdog

This encoding is analogous to the support encoding [9] applied to pseudo-Boolean constraints. Each variable x of the encoded constraint is connected to a watchdog formula that assigns through unit propagation $x = 0$ when the value $x = 1$ has no support.

Consider the constraint $\sum_{i \in I} a_i x_i < M$. Let $I_k = I \backslash \{k\}$ and let C_k be the constraint defined as $\sum_{i \in I_k} a_i x_i < M - a_k$. Clearly, whenever the constraint C_k is inconsistent with a partial assignment, the variable x_k must be fixed to 0.

Consider $PW(C_k)$ the polynomial watchdog encoding of the constraint C_k and ω_k the output variable of this encoding. As described by Algorithm 3,

Algorithm 3. $LPW(C)$

Require: a constraint $\sum_{i=1}^{n} a_i x_i < M$
  $F \leftarrow$ **true**
  for $k = 1$ to n **do**
    $C_k \leftarrow \sum_{i=1..n, i \neq k} a_i x_i < M - a_k$
    $F \leftarrow F \wedge (PW(C_k) \wedge (\overline{\omega_k} \vee \overline{x_k}))$
  end for
  return F

the Local Polynomial Watchdog can now be defined as the CNF: $LPW(C) = \bigwedge_{k=1}^{n} (PW(C_k) \wedge (\overline{\omega_k} \vee \overline{x_k}))$.

Theorem 1. *Any pseudo-Boolean Constraint of integer weights using n variables having a maximum weight of a_{max} can be translated into a CNF formula of $O(n^2 \log(n) \log(a_{max}))$ variables and $O(n^3 \log(n) \log(a_{max}))$ clauses such that Unit Propagation maintains Generalized Arc Consistency.*

Proof. The proof follows from the fact that GAC assigns 0 to some x_k if and only if the corresponding C_k is inconsistent with the partial assignment. In this case UP assigns $\omega_k = 1$ and then $x_k = 0$. The complexity comes from the fact that we have n Polynomial Watchdogs.

4.4 Implementation

The size of the watchdogs used in the two proposed encodings can be reduced by sharing sub-formulae – both into the same watchdog as well as between different ones – in a way to reduce the number of clauses. It is even possible to share sub-formulae between several input constraints. The key of such an optimization is how to split the input variables of each totalizer into the two sets of input variables of its sub-totalizers. A first basic implementation has been done, for validation purpose only. It is based on a static ordering of the variables of the input constraint, which are sorted in decreasing order of their coefficients. No extensive experimental evaluations of the LPW end GPW encodings has been performed yet because the implementation is not yet optimized, and anyway an extensive evaluation is not the purpose of this paper[3].

That said, the first few results suggest that the new encodings could be of practical interest. For exemple, the following unsatisfiable Bin-packing instance was encoded using our basic implementation: 16 objects with weights 211, 203, 202, 201, 200, 199, 198, 197, 196, 194, 175, 167, 166, 165, 164, and 162 must be put into 3 boxes, each with capacity 1000. For each box i and each object j, a Boolean variable x_{ij} denotes whether the object i belongs to the box j.

[3] Some tests were done on 1D bin packing instances, randomly generated instances and hand crafted instances, with the only aim to verify that the size of the LPW and GPW encodings does not make them intractable. We do not have enough space to present these experiments.

Three pseudo-Boolean constraints ensure the capacity requirement of each box, and 16 additional cardinality constraints ensure that each object belongs exactly to one box. The BDD encoding of [5] (see section 5) produces 38077 literals in 15637 clauses, and allows `minisat` to solve the problem within 486 seconds; the LPW encoding produces 58521 literals in 21615 clauses, and allows `minisat` to solve it within 12 seconds; the GPW encoding produces 7108 literals in 2714 clauses, and allows `minisat` to solve it within 3 seconds; the pseudo-Boolean solver `pueblo` [14] solves the initial instance within 23 seconds; `minisat+` solves the initial instance within 8 seconds.

In some cases, our *basic* version of the LPW encoding seems to produce a prohibitive number of clauses. For exemple, to put 50 objects into 5 boxes, each with capacity 1000, it required 2553715 literals in 884945 clauses, while GPW produced "only" 95675 literals in 34200 clauses.

5 Related Work

In [16], Warner proposes a linear CNF encoding of pseudo-Boolean constraints. It uses a binary adder network, which does not allow unit propagation to detect whether any input constraint is falsified by a given partial assignment.

[4] proposes an encoding which UP-maintains GAC on cardinality constraints. It is based on an extended version of the totalizers described in section 3 and requires $O(n \log n)$ additional variables and $O(n^2)$ clauses of size at most 3 to encode a constraint with n variables. [13] also introduces an encoding based on a unary representation but uses a odd-even merge sorting network, thereby reducing the number of clauses to $O(n \log^2 n)$.

In [15], Sinz introduces two other encodings. The first one uses a sequential adder network with a unary representation of integers. It maintains arc-consistency and, given a cardinality constraint $\sum_{j=1}^{n} x_i < M$, it produces a formula of size $O(nM)$. The second one uses a parallel adder network with a binary representation of integers. It does not detects local inconsistencies and produces a formula of size $O(n)$, but smaller than the one produced by Warner's encoding.

In 2006, Eén and Sörensson released the pseudo-Boolean solver `minisat+` [13], one of the best performers in the PB'06 competition. It is based on a conversion of pseudo-Boolean constraints to a CNF formula, which is submitted to the `minisat` solver. `minisat+` uses some heuristics to choose between three encoding techniques based respectively on binary decision diagram (BDD), adder networks, and sorting network. Another variant of BDD based encoding is simultaneously (and independently) introduced in [5].

The BDD based encoding transforms each pseudo-Boolean constraint into a binary decision diagram. Each node in the BDD represents a pseudo-Boolean constraint and the satisfaction of this constraint is reified by a propositional variable in the encoding. The root of the BDD represents the constraint to encode. Each node has at most two children which are obtained by assigning the first variable of the constraint to the two possible values 0/1 and simplifying

the resulting constraints. Nodes corresponding to trivial constraints are pruned. Two nodes can also share a common child. The relation between the truth of a node and the truth of its children only depends on the variable chosen to decompose the constraint and the variables corresponding to the nodes. In [13], this relation is encoded in six ternary clauses. In [5], a slightly different encoding is used, which translates each node of the BDD into two binary clauses and two ternary clauses. These two encodings maintain arc-consistency, but can produce an exponential number of clauses in the worst case [5].

The encoding based on adder networks produces a number of clauses (of length at most 4) linearly related to the size of the encoded constraint, as [16], but using a different structure. All the variables with a bit of a given weight in the base 2 representation of their coefficients are bundled in a *bucket*. The number of bits set to 1 in each bucket is computed using a binary adders network. The results are then combined thanks to additional adders. The resulting binary value feeds a comparator, which is optimized to deal with the constant bound of the constraint. Like Warner's one, this linear encoding does not detect inconsistencies, then cannot maintain arc-consistency.

In [13], the encoding based on sorting networks is founded on the unary representation of numbers [4,13] (see section 3). To compress the representation, a number is represented by several buckets in unary notation and each bucket has its own weight. Instead of using weights which are a power of a base b $(1, b, b^2, b^3, ..)$, [13] uses a general increasing sequence of positive integers. This is in fact a generalization of the usual representation of numbers in a base b with the exception that the ratio of the weights of two successive digits is no more a constant. The GPW encoding presents similarities with this encodings but there are several differences: (1) each of our bucket is related to a power of two, while the encoding used in `minisat+` uses arbitrary weights; (2) instead of our totalizers, `minisat+` uses odd even merge sorting networks; (3) `minisat+` does not uses a tare, which is an essential feature of GPW to ensure that the bound of the constraint is a round number of the weight of the last sorter. As a consequence, instead of a simple unit clause, a more complex non monotone circuit is used to establish whether the constraint is satisfied or not. More importantly, it does not maintain arc-consistency and it is not proved in [13] whether it UP-detects inconsistency or not.

Let us mention the standard exponential transformation with no auxiliary variables. Except for the trivial cases (i.e., constraints with only one literal and constraints that are either impossible to satisfy or to falsify), a constraint $\sum_{i=1}^{n} a_i x_i < M$ is translated as two sets of clauses. The first one encodes $(x_n = 0) \vee (\sum_{i=1}^{n-1} a_i x_i < M - a_n)$, and the second one encodes $\sum_{i=1}^{n-1} a_i x_i < M$. Unit resolution achieves the same propagations in this encoding as it does in the BDD one, then it maintains arc-consistency and detects local inconsistencies. Contrarily to the BDD one, direct encoding does not require additional variables, but it often produces bigger formulae because each clause corresponds to a *path* of the BDD. However, small constraints (with typically less than 6 variables) tend to produce more concise formulae.

6 Synthesis and Perspectives

This paper provides a theoretical contribution on the encoding of pseudo-Boolean constraints into CNF formulae. Now, it is known that there exists a polynomial encoding that allows unit propagation (implemented in all DPLL-based SAT solvers) to restore generalized arc consistency on the initial constraints. Clearly, this result opens new questions and new perspectives in the field of indirect resolution of pseudo-Boolean problems.

The space complexity of the proposed encoding is $O(n^3 \log(n) \log(a_{max}))$ but in practice many sub-formulae could be shared by several totalizers – both into the same watchdog as well as among different ones – in a way to reduce the number of produced clauses. The key is the order of the variables into the vector related to each bucket $\langle B_j \rangle$. Some work must be done to search for relevant ordering heuristics.

Another issue is the existence of structurally more concise encodings that maintain arc consistency. The underlying theoretical aim is to establish the minimum size for any such encoding. At this point, it is not known if there is a gap – in terms of space complexity[4] – between encodings which "only" detect local inconsistencies and encodings maintaining arc consistency. Furthermore, it is not clear whether the former outperforms the latter with modern SAT solvers. Answering these questions will probably require an extensive amount of future work.

Moreover, the existing encodings could be combined in a way to use a specific encoding for each individual constraint (and even the possibility to use redundant encodings could be considered). The `minisat+` solver always uses such an approach, but it may be improved and extended to the new encodings introduced in the present paper. These new encodings could also be improved by using a sorting network, as in [13], instead of a totalizer.

An even more crucial question is whether solving pseudo-Boolean problems with SAT solvers is actually relevant. On the one hand, this approach proved its efficiency (see `minisat+` at PB05) despite the fact that state-of-the-art encodings are not mature. On the other hand, one can hardly expect a SAT solver to outperform a native pseudo-Boolean solver when this technology becomes mature.

References

1. Abdulla, P.A., Bjesse, P., Eén, N.: Symbolic reachability analysis based on SAT-solvers. In: Schwartzbach, M.I., Graf, S. (eds.) TACAS 2000. LNCS, vol. 1785, pp. 411–425. Springer, Heidelberg (2000)
2. Aloul, F.A., Ramani, A., Markov, I.L., Sakallah, K.A.: Generic ILP versus Specialized 0-1 ILP: An Update. In: Proc. of Int. Conf. on Computer Aided Design (ICCAD 2002), pp. 450–457 (2002)
3. Bacchus, F.: Gac via unit propagation. In: Rossi, F. (ed.) CP 2003. LNCS, vol. 2833, pp. 133–147. Springer, Heidelberg (2003)

[4] Regarding the degree of the polynomial.

4. Bailleux, O., Boufkhad, Y.: Efficient CNF encoding of boolean cardinality constraints. In: Rossi, F. (ed.) CP 2003. LNCS, vol. 2833, pp. 108–122. Springer, Heidelberg (2003)
5. Bailleux, O., Boufkhad, Y., Roussel, O.: A Translation of Pseudo Boolean Constraints to SAT. Journal on Satisfiability, Boolean Modeling and Computation 2, 191–200 (2006)
6. Biere, A., Cimatti, A., Clarke, E.M., Fujita, M., Zhu, Y.: Symbolic Model Checking using SAT procedures instead of BDDs. In: Proc. of Design Automation Conference (DAC 1999), pp. 317–320 (1999)
7. Brand, S., Duck, G.J., Puchinger, J., Stuckey, P.J.: Flexible, Rule-based Constraint Model Linearisation. In: Hudak, P., Warren, D.S. (eds.) PADL 2008. LNCS, vol. 4902, pp. 68–83. Springer, Heidelberg (2008)
8. Ernst, M., Millstein, T., Weld, D.S.: Automatic SAT-Compilation of Planning Problems. In: IJCAI 1997, pp. 1169–1176 (1997)
9. Gent, I.P.: Arc Consistency in SAT. In: Proc. of the Fifteenth European Conference on Artificial Intelligence (ECAI 2002), pp. 121–125 (2002)
10. Grastien, A., Anbulagan, A., Rintanen, J., Kelareva, E.: Diagnosis of Discrete-Event Systems Using Satisfiability Algorithms. In: AAAI-2007, pp. 305–310 (2007)
11. Hertel, A., Hertel, P., Urquhart, A.: Formalizing Dangerous SAT Encodings. In: Marques-Silva, J., Sakallah, K.A. (eds.) SAT 2007. LNCS, vol. 4501, pp. 159–172. Springer, Heidelberg (2007)
12. Baker, A.B., Crawford, J.M.: Experimental Results on the Application of Satisfiability Algorithms to Scheduling Problems. In: Proc. of the Twelfth National Conference on Artificial Intelligence, pp. 1092–1097 (1994)
13. Eén, N., Sörensson, N.: Translating Pseudo-Boolean Constraints into SAT. Journal on Satisfiability, Boolean Modeling and Computation 2, 1–26 (2006)
14. Sheini, H.M., Sakallah, K.A.: Pueblo: a modern pseudo-Boolean SAT solver. In: Design, Automation and Test in Europe, 2005. Proc., pp. 684–685 (2005)
15. Sinz, C.: Towards an Optimal CNF Encoding of Boolean Cardinality Constraints. In: van Beek, P. (ed.) CP 2005. LNCS, vol. 3709, pp. 827–831. Springer, Heidelberg (2005)
16. Warners, J.P.: A linear-time transformation of linear inequalities into conjunctive normal form. Information Processing Letters 68(2), 63–69 (1998)