New Encoding for Translating Pseudo-Boolean Constraints into SAT

Amir Aavani and **David Mitchell** and **Eugenia Ternovska**
Simon Fraser University, Computing Science Department
{aaa78,mitchell,ter}@sfu.ca

Abstract

A Pseudo-Boolean (PB) constraint is a linear arithmetic constraint over Boolean variables. PB constraints are and widely used in declarative languages for expressing NP-hard search problems. While there are solvers for sets of PB constraints, there are also reasons to be interested in transforming these to propositional CNF formulas, and a number of methods for doing this have been reported. We introduce a new, two-step, method for transforming PB constraints to propositional CNF formulas. The first step re-writes each PB constraint as a conjunction of PB-Mod constraints, and the second transforms each PB-Mod constraint to CNF. The resulting CNF formulas are compact, and make effective use of unit propagation, in that unit propagation can derive facts from these CNF formulas which it cannot derive from the CNF formulas produced by other commonly-used transformation. We present a preliminary experimental evaluation of the method, using instances of the number partitioning problem as a benchmark set, which indicates that our method out-performs other transformations to CNF when the coefficients of the PB constraints are not small.

Introduction

A Pseudo-Boolean constraint (PB-constraint) is an equality or inequality on a linear combination of Boolean literals, of the form

$$\sum_{i=1}^{n} a_i l_i \ \mathbf{op} \ b$$

where $\mathbf{op}$ is one of $\{<, \leq, =, \geq, >\}$, $a_1, \cdots a_n$ and b are integers, and $l_1, \cdots, l_n$ are Boolean literals. Under truth assignment $\mathcal{A}$ for the literals, the left-hand evaluates to the sum of the coefficients whose corresponding literals are mapped to *true* by $\mathcal{A}$. PB-constraints are also known as 0-1 integer linear constraints. By taking the variables to be propositional literals, rather than 0-1 valued arithmetic variables, we can consider the combination of PB-constraints with other logical expressions. Moreover, a propositional clause $(l_1 \vee \ldots \vee l_k)$ is equivalent to the PB-constraint $\sum_{i=1}^{k} l_i \geq 1$. Thus, PB-constraints are a natural generalization of propositional clauses with which it is easier to describe arithmetic properties of a problem. For example, the Knapsack problem has a trivial representation as a conjunction of two PB-constraints:

$$\sum_{i=1}^{n} w_i l_i < C \ \wedge \ \sum_{i=1}^{n} v_i l_i > V,$$

but directly representing it with a propositional CNF formula is non-trivial.

Software which finds solutions to sets of PB-constraints (PB solvers) exist, for example PBS (Aloul et al. 2002) and PUEBLO (Sheini and Sakallah 2006), but there is not a sustained effort to produce continually updated high-performance solvers. Integer linear programming (ILP) systems can be used to find solutions to sets of PB-constraints, but they are generally optimized for performance on certain types of optimization problems, and do not perform well on some important families of search problems. Moreover, the standard ILP input is a set of linear inequalities, and many problems are not effectively modelled this way, such as problems involving disjunctions of constraints, such as $(p \wedge q) \vee (r \wedge s)$. There are standard techniques for transforming these, involving additional variables, but extensive use of these techniques causes performance problems. (Transforming problems to propositional CNF also requires adding new variables, but there seems to be little performance penalty in this case.)

Another approach to solving problems modelled with PB-constraints is to transform them to a logically equivalent set of propositional clauses and then apply a SAT solver. There are at least two clear benefits of this approach. One is that high-performance SAT solvers are being improved constantly, and since they take a standard input format, there is always a selection of good, and frequently updated, solvers to make use of. A second is that solving problems involving Boolean combinations of constraints is straightforward. This approach is particularly attractive for problems which are naturally represented by a relatively small number of PB constraints together which a large number of purely Boolean constraints.

The question of how best to transform a set of PB constraints to a set of clauses is complex. Several methods have been reported, but there is still much to be learned. Here, we describe a new method of transformation, and to present some preliminary evidence of its utility.

We define a PBMod-constraint to be of the form:

$$\sum_{i=1}^{n} a_i l_i \equiv b \text{ (mod M)}$$

where $a_1, \cdots a_n$ and b are non-negative integers less than M, and $l_1, \cdots, l_n$ are literals.

Our method of transforming a PB-constraint to CNF involves first transforming it to a set of PB-Mod constraints, and then transforming these to CNF. Thus, we replace the question of how best to transform an arbitrary PB-constraint to CNF with two questions: how to choose a set of PB-Mod constraints, and how to transform each of these to CNF. There are benefits of this, due to properties of the PB-Mod constraints. For example, we show that there are many PB-constraints whose unsatisfiability can be proven by showing the unsatisfiability of a PBMod-constraint, which is much simple.

We present two methods for translating PBMod-constraints to CNF. Both these encodings allow unit propagation to infer inconsistency if the current assignment cannot be extended to a satisfying assignment for that PBMod-constraint, and hence unit propagation can infer inconsistency for the original PB-constraint. We also show that the number of PB-constraints for which unit propagation can infer inconsistency, given the output of proposed translation, is much larger than for the other existing encodings. We also point out that it is impossible to translate all PB-constraints in the form $\sum a_i l_i = b$ into polynomial size arc-consistent CNF unless P=NP.

We also present the results of an experimental study, using instances of the number partitioning problem as a benchmark, which indicates that our new method outperforms others in the literature.

For the sake of space, proofs are omitted from this paper. All proofs can be found in (Aavani 2011).

Notation and Terminology

Let X be a set of Boolean variables. An assignment $\mathcal{A}$ to X is a possibly partial function from X to $\{true, false\}$. Assignment $\mathcal{A}$ to X is a *total assignment* if it is defined at every variable in X. For any $S \subseteq X$, we write $\mathcal{A}[S]$ for the assignment obtained by restricting the domain of $\mathcal{A}$ to the variables in S. We say assignment $\mathcal{B}$ extends assignment $\mathcal{A}$ if $\mathcal{B}$ is defined on every variable that $\mathcal{A}$ is, and for every variable x where $\mathcal{A}$ is defined, $\mathcal{A}(x) = \mathcal{B}(x)$.

A literal, l, is either a Boolean variable or negation of a Boolean variable and we denote by var(l) the variable underlying literal l. Assignment $\mathcal{A}$ satisfies literal l, written $\mathcal{A} \models l$, if l is an atom x and $\mathcal{A}(x) = true$ or l is a negated atom $\neg x$ and $\mathcal{A}(x) = false$.

A clause $C = \{l_1, \cdots, l_m\}$ over X is a set of literals such that $var(l_i) \in X$. Assignment $\mathcal{A}$ satisfies clause $C = \{l_1, \cdots, l_m\}$ if there exists at least one literal l_i such that $\mathcal{A} \models l_i$. A total assignment falsifies clause C if it does not satisfy any of its literals. An assignment satisfies a set of clauses if it satisfies all the clauses in that set.

A PB-constraint Q on X is an expression of the form:

$$a_1 l_1 + \cdots + a_n l_n \ \textbf{op} \ b \tag{1}$$

where $\textbf{op}$ is one of $\{<, \leq, =, \geq, >\}$, for each i, a_i is an integer and l_i a literal over X, and b is an integer. We call a_i the coefficient of l_i, and b the bound.

Total assignment $\mathcal{A}$ to X satisfies PB-constraint Q on X, written $A \models Q$, if $\sum_{i:\mathcal{A}\models l_i} a_i \ \textbf{op} \ b$, that is, the sum of coefficients for literals mapped to *true* (the left hand side) satisfies the given relation to the bound (the right hand side).

Canonical Form

In this paper, we focus on translating PB equality constraints with positive coefficients:

$$a_1 x_1 + \cdots + a_n x_n = b \tag{2}$$

where integers ($a_1 \cdots a_n$ and b) are all positive.

Definition 1 *Constraints Q_1 on X and Q_2 on $Y \supseteq X$ are equivalent iff for every total assignment $\mathcal{A}$ for X which satisfies Q_1, there exists an extension of $\mathcal{A}$ to Y which satisfies Q_2, and every total assignment $\mathcal{B}$ to Y which satisfies Q_2 also satisfies Q_1.*

It is not hard to show that every PB-constraint has an equivalent PB-constraint of the form (2). For sake of space, we do not include the details but refer interested readers to (Aavani 2011).

Valid Translation

Definition 2 *Let Q be a PB-constraint or PB-Mod constraint over variables $X = \{x_1, \cdots, x_n\}$, Y a set of Boolean variables (called auxiliary variables) disjoint from X, and v a Boolean variable not occurring in $X \cup Y$, and $C = \{C_1, \cdots, C_m\}$ a set of clauses on $X \cup Y \cup \{v\}$. Then we say the pair $\langle v, C\rangle$, is a* valid translation *of Q if*

1. *C is satisfiable, and*
2. *if $\mathcal{A}$ is a total assignment for $X \cup Y \cup \{v\}$ that satisfies C, then*

$$\mathcal{A} \models Q \iff \mathcal{A} \models v.$$

Intuitively, C ensures that every v always takes the same truth value as Q.

In (Bailleux, Boufkhad, and Roussel 2009), a translation is defined to be a set of clauses C such that $\mathcal{A} \models Q$ off some extension of $\mathcal{A}$ (to the auxiliary variables of C) satisfies C. If $\langle v, C\rangle$ is a valid translation by Definition 2, then $(v) \cup C$ is a translation in this other sense, and if C is a translation in the other sense, then $\langle v, D\rangle$, where D is equivalent to $v \leftrightarrow C$, is a valid translation. So these two definitions are essentially equivalent, except that our definition makes available a variable which always has the same truth value as Q, which can be convenient. For example, it makes it easy to use Q conditionally.

Example 1 *Let Q be the unsatisfiable PB-constraint $2x_1 + 4\neg x_2 = 3$. Then the pair $\langle v, \{C_1\}\rangle$, where $C_1 = \{\neg v\}$, is a valid translation of Q.*

Example 2 *Let Q be the satisfiable PB-constraint $1x_1 + 2x_2 = 2$. Then $\langle v, C\rangle$, where C is any set of clauses logically equivalent to $(v \leftrightarrow \neg x_1) \wedge (v \leftrightarrow x_2)$ is a valid translation of Q. Here, $X = \{x_1, x_2\}$ and $Y = \emptyset$.*

In describing construction of translations, we will sometimes overload our notation, using a symbol for both a variable and a translation. For example, if D is a valid translation, we may use D as a variable in a clause for constructing another translation. Thus, D is the pair $\langle D, C\rangle$.

Tseitin Transformation

The usual method for transforming a propositional formula to CNF is that of Tseitin(Tseitin 1968). To transform formula ϕ to CNF, a fresh propositional variable is used to represent the truth value of each subformula of ϕ. For each subformula ψ, denote by ψ' the associated propositional variable. If ψ is a variable, then ψ' is just ψ. The CNF formula is the set of clauses containing the clause (ϕ'), and for each sub-formula ψ of ϕ:

1. If $\psi = \psi_1 \vee \psi_2$, the clauses $\{\neg\psi', \psi_1', \psi_2'\}$, $\{\psi', \neg\psi_1'\}$ and $\{\psi', \neg\psi_2'\}$;
2. If $\psi = \psi_1 \wedge \psi_2$, the clauses $\{\neg\psi', \psi_1'\}$, $\{\neg\psi', \psi_2'\}$, and $\{\psi', \neg\psi_1', \neg\psi_2'\}$;
3. If $\psi = \neg\psi_1$, the clauses $\{\neg\psi', \neg\psi_1'\}$ and $\{\psi', \psi_1'\}$.

New Method for PBMod-constraints

We define a normal PBMod-constraint be of the form:

$$\sum_{i=1}^{n} a_i l_i \equiv b \text{ (mod M)}, \qquad (3)$$

where $0 \leq a_i < M$ for all $1 \leq i \leq n$ and $0 \leq b < M$. Total assignment $\mathcal{A}$ is a solution to a PBMod-constraint iff the value of left-hand side summation under $\mathcal{A}$ minus the value of right-hand side of the equation, b, is a multiple of M.

Definition 3 *If Q is the PB-constraint $\sum a_i l_i = b$ and M an integer greater than 1, then by $Q[M]$ we denote the PBMod-constraint $\sum a_i' l_i \equiv b' (mod\ M)$ where:*
1. $a_i' = a_i \bmod M$,
2. $b' = b \bmod M$.

Example 3 *Let Q be the constraint $6x1 + 5x2 + 7x3 = 12$. Then, we have that*
$Q[3]$ is $0x1 + 2x2 + 1x3 \equiv 0(mod\ 3)$, and
$Q[5]$ is $1x1 + 0x2 + 2x3 = 2(mod\ 5)$.

Every solution to a PB-constraint Q is also a solution to $Q[M]$ for any $M \geq 2$. Also, for sufficiently large values of M, each solution to $Q[M]$ is a solution to Q.

Proposition 1 *If Q is a PB-constraint $\sum a_i l_i = b$ and $M > \sum a_i$ then $Q[M]$ and Q have the same satisfying assignments.*

More interesting is that, for a given PB-constraint Q, we can construct sets constraints $Q[M_i]$, none of which are equivalent to Q, but such that their conjunction has the same set of solutions as Q. Our goal will be to choose values of M_i such that the resulting set of PB-Mod constraints is easy to transform to CNF.

Proposition 2 *Let Q be the PB-constraint $\sum a_i l_i = b$, M_1 and M_2 be integers with $M_3 = lcm\ (M_1, M_2)$. Further, let S_1 be the set of satisfying assignments for $Q[M_1]$, and S_2 be the set of assignments satisfying $Q[M_2]$. Then the set of satisfying assignments for $Q[M3]$ is $S_1 \cap S_2$.*

Proposition 2 tells us that in order to find the set of solutions to a PBMod-constraint modulo $M_3 = \text{lcm}(M_1, M_2)$, one can find the set of solutions to two PBMod-constraints (modulo M_1 and M_2) and return their intersection. This generalizes in the obvious way.

Lemma 1 *Let $\{M_1, \cdots, M_m\}$ be a set of m positive integers and $M = lcm(M_1, \ldots, M_m)$. Let Q the PB-constraint $\sum a_i l_i = b$. If $M > \sum a_i$, and S_i is the set of satisfying assignments for $Q[M_i]$, then the set of satisfying assignments of $Q[M]$ is*

$$\bigcap_{i \in 1..m} S_i.$$

We can now easily construct a valid translation of a PB-constraint from valid translations of a suitable set of PB-Mod constraints.

Theorem 1 *Let Q be a PB-constraint $\sum a_i l_i = b$, $\{M_1, \cdots, M_m\}$ a set of positive integers, and $M = lcm(M_1, \cdots, M_m)$ with $M > \sum a_i$. Suppose that, for each $i \in \{1, \cdots m\}$, $\langle v_k, C_k\rangle$ is a valid translation of $Q[M_i]$, each over distinct sets of variables. Then for any set C of clauses logically equivalent to $\cup_i C_i \cup C'$, where C' is a set of clauses equivalent to $v \leftrightarrow (v_1 \wedge v_2 \cdots \wedge v_m)$, the pair $\langle v, C\rangle$, is a valid translation of Q.*

Since $\text{lcm}(2, \cdots, k) \geq 2^{k-1}$, (Farhi and Kane 2009), the set $\mathtt{M}^{\mathtt{N}} = \{2, \cdots, \lceil \log \sum a_i \rceil + 1\}$ can be used as the set of moduli for encoding $\sum a_i l_i = b$.

Another candidate for set of moduli is the first m prime numbers, where m is the smallest number such that the produce of the first m primes exceeds $\sum a_i$. We will denote this set by $\mathtt{M}^{\mathtt{P}}$. The following proposition gives an estimate for the size of set $\mathtt{M}^{\mathtt{P}}$, and for the value of P_m. As usual, we denote by P_i the i^{th} prime number.

Proposition 3 *Let m be the smallest integer such that the product of the first m primes is greater than S. Then:*
1. $m = |\mathtt{M}^{\mathtt{P}}| = \theta(\frac{\ln S}{\ln \ln S})$.
2. $P_m < \ln S$.

A third candidate is the set

$$\mathtt{M}^{\mathtt{PP}} = \{P_i^{n_i} \mid P_i^{n_i - 1} \leq \lg S \leq P_i^{n_i}\}.$$

It is straightforward to observe that $|\mathtt{M}^{\mathtt{PP}}| \leq \frac{\lg S}{\lg \lg S}$ and the its maximum element is at most $\lg S$.

In general, the size of a description of PB-constraint $\sum a_i l_i = b$ is $\theta(n \log a_{Max})$ where n is the number of literals (coefficients) in the constraint and a_{Max} is the value of the largest coefficient. The description of PBMod-constraint $Q[M]$ has size $\theta(n \log M)$. So, a translation for $Q[M]$ which produces a CNF with $O(n^{k_1} M^{k_2})$ clauses and variables, for some constants k_1 and k_2, (which may be exponential the input size), provides a may to translate PB-constraints to CNF of size polynomial in the representation of the PB-constraint. Two such translations are described in the next section. We describe several others in (Aavani 2011).

Encoding For PB-Mod Constraints

In this section, we describe translations of PBMod-constraints of the form (3) to CNF. Remember that our ultimate goal is not translation of PB-constraints. For simplicity, we assume all coefficients in each PBMod-constraint are non-zero.

Dynamic Programming Based Transformation (DP)

The translation presented here encodes PBMod-constraints using a Dynamic Programming approach. Let D^j_m be a valid translation for $\sum_{i=1}^{j} a_i l_i \equiv m(\text{mod M})$. We can use the following set of clauses to describe the relationship among D^j_m, $D^{j-1}_{m-a_j}$, D^j_m and l_j:

1. If both $D^{j-1}_{m-a_j}$ and l_j are *true*, D^j_m must be *true*, which can be represented by the clause $\{\neg D^{j-1}_{m-a_j}, \neg l_j, D^l_m\}$.
2. If D^{j-1}_m is *true* and l_j is *false*, D^j_m must be *true*, i.e., $\{\neg D^{j-1}_m, l_j, D^j_m\}$.
3. If D^j_m is *true*, either D^{j-1}_m or $D^{j-1}_{m-a_j}$ must be *true*, i.e., $\{\neg D^j_m, D^{j-1}_m, D^{j-1}_{m-a_j}\}$.

For the base cases, when $j = 0$, we have:

1. D^0_0 is *true*, i.e., $\{D^0_0\}$.
2. If $m \neq 0$, D^0_m is *false*, i.e., $\{\neg D^0_m\}$.

Proposition 4 *Let $D = \{D^j_m\}$ and C be the set of clauses used to describe variables in D. Then, pair $\langle D^n_b, C\rangle$ is valid translation for (3).*

By applying standard dynamic programming techniques, we can avoid describing the unnecessary D^j_m, and obtain a smaller CNF.

By adding the following clauses, we can boost the performance of unit propagation.

1. If $D^j_{m_1}$ is *true*, $D^j_{m_2}$ should be *false*$(m_1 \neq m_2)$, i.e., $\{\neg D^j_{m_1}, \neg D^j_{m_2}\}$.
2. There is at least one m such that D^j_m is *true*, i.e., $\{D^j_m | m = 0 \cdots M-1\}$.

Binary Decision Diagrams, BDD, are standard tools for translating constraints to SAT. One can construct a BDD-based encoding for PBMod-constraints similar to BDD-based encoding for PB-constraint described in (Eén and Sorensson 2006). Unit propagation can infer more facts on the CNF generated by boosted version of DP-based encoding than the CNF generated by BDD-based encoding. Comparing BDD-based and DP-based encodings, former produces larger CNF while unit propagation infers the same facts on the output of both encodings.

Remark 1 *In (Aavani 2011), we proved that DP-based encoding, plus the extra clauses, has the following property. Given partial assignment $\mathcal{A}$, if there is no total assignment $\mathcal{B}$ extending $\mathcal{A}$ such that $\mathcal{B}$ satisfies both C and $\sum_{i=1}^{j} a_i l_i \equiv m(\text{mod M})$, then unit propagation infers false as the value for variable D^m_j.*

Divide and Conquer Based Transformation (DC)

The translation presented next reflects a Divide and Conquer approach. We define auxiliary variables in $D = \{D^{s,l}_a\}$ such that variable $D^{s,l}_a$ describes the necessary and sufficient condition for satisfiability of subproblem $\sum_{i=s}^{s+l-1} a_i x_i \equiv a \text{ (mod M)}$.

Let $D^{s,l} = \{D^{s,l}_a : 0 \leq a < M\}$. We can use the following set of clauses to describe the relation among the $3 * M$ variables in sets $D^{s,l}$, $D_{s,\frac{l}{2}}$ and $D_{s+\frac{l}{2},\frac{l}{2}}$:

1. If both $D^{s,\frac{l}{2}}_{m_1}$ and $D^{s+\frac{l}{2},\frac{l}{2}}_{m_2}$ are *true*, $D^{s,l}_{m_1+m_2}$ should be *true*, i.e., $\{\neg D^{s,\frac{l}{2}}_{m_1}, \neg D^{s+\frac{l}{2},\frac{l}{2}}_{m_2}, D^{s,l}_{m_1+m_2}\}$.
2. If $D^{s,l}_{m_1}$ is *true*, $D^{s,l}_{m_2}$ should be *false*$(m_1 \neq m_2)$, i.e., $\{\neg D^{s,l}_{m_1}, \neg D^{s,l}_{m_2}\}$.
3. There is at least one m such that $D^{s,l}_m$ is *true*, i.e., $\{D^{s,l}_m | m = 0 \cdots M-1\}$.

For the base cases, when $l = 1$, we have:

1. $D^{s,1}_0$ is *true* iff x_s is *false*, i.e., $\{\{x_s, D^{s,1}_1\}, \{\neg x_s, \neg D^{s,1}_1\}\}$.
2. $D^{s,1}_1$ is *true* iff x_s is *true*, i.e., $\{\{\neg x_s, D^{s,1}_1\}, \{x_s, \neg D^{s,1}_1\}\}$.

Proposition 5 *Let $D = \{D^{s,l}_a\}$ and C be the clauses which are used to describe the variables in D. Then, pair $\langle D^n_b, C\rangle$ is a valid translation for (3).*

Remark 2 *In (Aavani 2011), we showed another version of DC-based encoding which also has the property we described in Remark 1.*

Theorem 2 *The numbers of clauses and auxiliary variables used in the DP and CD translations of PBMod constraint $\sum a_i x_i \equiv b$ (mod M), and the depths of the formulas implicit in these CNF formulas, are as given in Table 1. These same properties, for the PB-constrain translations obtained DP and DC translations together with M^P or M^{PP} as moduli, are as given in Table 2.*

Encoder	# of Aux. Vars.	# of Clauses	Depth
DP	$O(nM)$	$O(nM)$	$O(n)$
DC	$O(nM)$	$O(nM^2)$	$O(\log n)$

Table 1: Summary of size and depth of translations for $\sum a_i x_i \equiv b$ (mod M).

In the previous section, we described two candidates for sets of moduli, namely `Prime` and `PrimePower`, and in this section, we explained two encodings for transforming PBMod constraints to SAT, namely DP and DC. This give us four different translations for PB constraints to SAT. Table 2 summarizes the number of clauses and variables and the depth of corresponding formula for these translations, and also for the Sorting Network based encoding (Eén 2005), and Binary Adder encoding (Eén 2005).

PBMod Endr	# of Vars.	# of Clauses	Depth
Prime.DP	$O(\frac{\ln(S)}{\ln\ln S})$	$O(\frac{n\ln(S)}{\ln\ln(S)})$	$O(n)$
Prime.DC	$O(n\frac{\ln(S)}{\ln\ln(S)})$	$O(n\left(\frac{\ln(S)}{\ln\ln(S)}\right)^2)$	$O(\log n)$
PPower.DP	$O(\frac{\log(S)}{\log\log S})$	$O(\frac{n\log(S)}{\log\log(S)})$	$O(n)$
PPower.DC	$O(n\frac{\log(S)}{\log\log(S)})$	$O(n\left(\frac{\log(S)}{\log\log(S)}\right)^2)$	$O(\log n)$
BAdder	$O(n\log(S))$	$O(n\log(S))$	$O(\log(S)*\log n)$
SN	$O(n\log(S/n)\log^2(n\log(S/n)))$	$O(n\log(S/n)\log^2(n\log(S/n)))$	$O(\log^2(n\log(S/n)))$

Table 2: Summary of size and depth of different encodings for translating $\sum a_i x_i = b$, where $S = \sum a_i$.

Performance of Unit Propagation

Here we examine some properties of the proposed encodings.

Background

Generalized arc-consistency (GAC) is one of the desired properties for an encoding which is related to the performance of unit propagation, UP, procedure inside a SAT Solver. Bailluex and et. al., in (Bailleux, Boufkhad, and Roussel 2009), defined UP-detect inconsistency and UP-maintain GAC for PB-constraint's encodings. Although, the way they defined a translation is slightly different from us, these two concepts can still be discussed in our context.

Let E be an encoding method for PB-constraints, Q be a PB-constraint on X and $\langle v, C\rangle = E(Q)$ the translation for Q obtained from encoding E. Then,

1. *Encoding E for constraint Q supports UP-detect inconsistency* if for every (partial) assignment $\mathcal{A}$, we have that every total extension of $A[X]$ makes Q false if and only if unit propagation derives $\{\neg v\}$ from $C \cup \{\{x\} \mid \mathcal{A} \models x\}$;
2. *Encoding E for constraint Q is said to UP-maintain GAC* if for every (partial) assignment $\mathcal{A}$ and any literal l where $var(l) \in X$, we have that l is true in every total extension of A that satisfies Q, if and only if unit propagation derives $\{l\}$ from $C \cup \cup\{v\} \cup \{\{x\} \mid \mathcal{A} \models x\}$;

An encoding for PB-constraints is generalized arc-consistent, or simply arc-consistent, if it supports both UP-detect inconsistency and UP-maintain GAC for all possible constraints.

In this section, we show that there cannot be an encoding for PB-constraint in form $\sum a_i l_i = b$ which always produces a polynomial size arc-consistent CNF unless P=co-NP. Also we study the arc-consistency of our encoding and discuss why one can expect the proposed encodings to perform well.

Hardness Result

Here, we show that it is not very likely to have a generalized arc-consistent encoding which always produces polynomial size CNF.

Theorem 3 *There does not exist a UP-detectable encoding which always produces polynomial size CNF unless P= co-NP. There does not exist a UP-maintainable encoding which always produces polynomial size CNF unless P= co-NP.*

Proof (sketch) The theorem can be proven by observing that a subset sum problem instance can be written as a PB-constraint, and having a UP-detectable encoding enables us to prove unsatisfiability whenever the original subset problem instance is not satisfiable. The proof for hardness of having UP-maintainable encoding is similar to this argument. For complete proof, see (Aavani 2011).

UP for Proposed Encodings

Although there is no arc-consistent encoding for PB-constraints, both DP-based and DC-based encodings for PBMod-constraints are generalized arc-consistent encodings.

Also, as mentioned before, unit propagation is able to infer inconsistency, on the CNF generated by these encodings, as soon as the current partial assignment cannot be extended to a total satisfying assignment. Notice that what we state here is more powerful than arc-consistency as it considers the auxiliary variables, too. More formally, let $\langle v, C\rangle$ be the output of DP-based (DC-based) encoding for PBMod-constraint Q. Given a partial assignment $\mathcal{A}$ s.t., $v \in \mathcal{A}^+$,

$$\mathcal{A} \not\models C \cup \{v\} \Leftrightarrow \mathcal{A} \not\models_{UP} C \cup \{v\}. \qquad (4)$$

This feature enables SAT solver to detect their mistakes on each of PBMod-constraints as soon as such a mistake occurs.

In the rest of this section, we study the cases for which we expect SAT solvers to perform well on the output of our encoding. Let Q be a PB-constraint on X, $\mathcal{A}$ be a partial assignment and $Ans(\mathcal{A})$ be the set of total assignment, to X, satisfying Q and extending $\mathcal{A}[X]$. There are two situations in which UP is able to infer the values of input variables:

1. Unit Propagation Detects Inconsistency: One can infer the current partial assignment, $\mathcal{A}$, cannot satisfying Q by knowing $Ans(\mathcal{A}) = \emptyset$. Recall that there are partial assignments and PB-constraints such that although $Ans(\mathcal{A}) = \emptyset$, each of the m PBMod-constraints has non-empty solution (but the intersection of their solution is empty).
   If at least one of the m PBMod-constraints is inconsistent with the current partial assignment, UP can infer inconsistency, in both DP and DC encodings.
2. Unit Propagation Infers the Value for an Input Variable: One can infer the value of input variable x_k is *true/false* if x_k takes the same value in all the solutions to Q. For this kind of constraints, UP might be able to infer the value of x_k, too.

If there exists a PBMod-constraint for which all its solutions which extend $\mathcal{A}$, have mapped x_k to the same value, UP can infer the value of x_k.

These two cases are illustrated in the following example.

Example 4 *Let* $Q(X) = x_1 + 2 * x_2 + 3x_3 + 4 * x_4 + 5 * x_5 = 12$.

1. *If $\mathcal{A}$, the current partial assignment, is $\mathcal{A}= \{\neg x_2, \neg x_4\}$ and $M = 5$. There is no total assignment satisfying $1x_1 + 3x_3 + 0x_5 \equiv 2$ (mod 5).*
2. *If $\mathcal{A}$, the current partial assignment, is $\mathcal{A}= \{\neg x_3, \neg x_5\}$ and $M = 2$, there are four total assignments extending A and satisfying PBMod-constraint $1x_1 + 0x_2 + 0x_4 \equiv 0$ (mod 2). In all of them, x_1 is mapped to false.*

A special case of the second situation is when UP can detect the values of all $x \in X$ given the current partial assignment. In the rest of this section, we estimate the number of PB-constraints for which UP can solve the problem. More precisely, we give a lower bound on the number of PB-constraints for which UP detects inconsistency or it expands an empty assignment to a solution given the translation of those constraints.

Let us assume the constraints are selected, uniformly at random, from $\{\sum a_1 l_1 + \cdots + a_n l_n = b : 1 \leq a_i \leq A = 2^{R(n)}$ and $1 \leq b \leq n * A\}$ where $R(n)$ is a polynomial in n and $R(n) > n$. To simplify the analysis, we use the same prime modulos $\mathtt{M}^{\mathtt{P}} = \{P_1 = 2, \cdots, P_m = \theta(R(n)) > 2n\}$ for all constraints.

Consider the following PBMod-constraints:

$$1x_1 + \cdots + 1x_{n-1} + 1x_n = n + 1 (\bmod P_m) \quad (5)$$

$$1x_1 + \cdots + 1x_{n-1} + 1x_n = n (\bmod P_m) \quad (6)$$

It is not hard to verify that (5) does not have any solution and (6) has exactly one solution. It is straightforward to verify that UP can infer inconsistency given a translation obtained by DP-based (DC-based) encoding for (5), even if the current assignment is empty. Also, UP expands the empty assignment to an assignment mapping all x_i to *true* on a translation for (6) obtained by either DP-based encoding or DC-based encoding. *Chinese Remainder Theorem*, (Ding, Pei, and Salomaa 1996), implies that there are $(A/P_m)^{n+1} = 2^{(n+1)R(n)}/R(n)^{n+1}$ different PB-constraints in the form $\sum a_1 l_i = b$ such that their corresponding PBMod-constraints, where the modulo is P_m, are the same as (5). The same claim is true for (6).

The above argument shows that, for proposed encoding, the number of easy to solve PB-constraints is huge. In (Aavani 2011), we showed that this number is much smaller for Sorting network:

Observation 1 *(Aavani 2011) There are at most $(\log A)^n$ instances where the CNF produced by Sorting Network encoding maintains arc-consistency, while this number for our encoding is at least $(A/\log(A)))^n$. So, if $A = 2^{R(n)}$, almost always we have $2^{R(n)}/R(n) \gg R(n)$.*

Observation 2 *(Aavani 2011) There is a family of PB-constraints whose translation through totalizer-based encoding is not arc-consistent but the translation obtained by our encoding is arc-consistent.*

Experimental Evaluation

By combining any modulo selection approach and any PBMod-constraint encoder, one can construct a PB-constraint solver. In this section, we selected the following configurations: Prime with DP (Prime.DP), Prime with DC (Prime.DC). We used CryptoMiniSAT as the SAT solver for our encodings, as it performed better than MiniSAT, on our initial benchmarking experiments.

To evaluate the performance of these configurations, we used the Number Partitioning Problem, NPP. Given a set of integers $S = \{a_1, \cdots, a_n\}$, NPP asks whether there is a subset of S such that the summation of its members is exactly $\sum a_i/2$. Following (Gent and Walsh 1998), we generated 100 random instances for NPP, for a given n and L as follows:

Create set $S = \{a_1, \cdots, a_n\}$ such that each of a_i is selected independently at random from $[0 \cdots 2^L]$.

We ran each instance on our two configurations and also on two other encodings, Sorting Network based encoding (SN), Binary Adder Encoding (BADD)(Eén and Sorensson 2006), provided by MiniSAT+[1]. All running times, reported in this paper, are the total running times (the result of summation of times spent to generate CNF formulas and time spent to solve the CNF formulas). We also tried to run the experiments with BDD encoder, but as the CNF produced by BDD encoder is exponentially big, it failed to solve medium and large size instances.

Before we describe the result of experiments, we discuss some properties of the number partitioning problem.

Number Partitioning Problem

The Number partitioning problem is an NP-Complete problems, and it can also be seen as a special case of subset sum problem. In the SAT context, an instance of NPP can be rewritten as a PB-constraint whose comparison operator is "=". Neither this problem nor subset sum problem has received much attention by the SAT community.

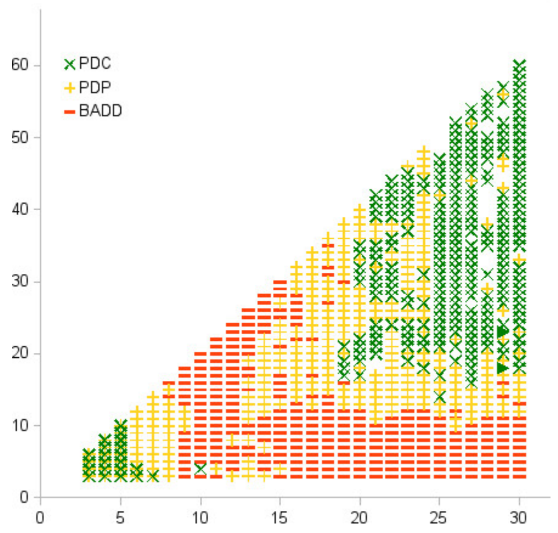
Size of an instance of NPP, where set S with n elements and a_{Max} is the maximum absolute value in S, is $\theta(n * \log(a_{Max})) + n)$. It is known that if the value of a_{Max} is polynomial wrt n, the standard dynamic programming approach can solve this problem in time $O(na_{ax})$, which is polynomial time wrt to the instance size. If a_{Max} is too large, $2^{\Omega(2^{\theta(n)})}$, the naive algorithm, which generates all the 2^n subsets of S, works in polynomial time wrt the instance size. The hard instances for this problem are those in which a_{Max} is neither too small nor too large wrt n.

In (Borgs, Chayes, and Pittel 2001), the authors defined $k = L/n$ and showed that NPP has a phase transition at $k = 1$: for $k < 1$, there are many perfect partitions with probability tending to 1 as $n \mapsto \infty$, while for $k > 1$, there are not perfect partitions with probability tending to 1. as $n \mapsto \infty$.

Experiments

All the experiments were performed on a Linux cluster (Intel(R) Xeon(R) 2.66GHz). We set the time limit for the to

[1]http://minisat.se/



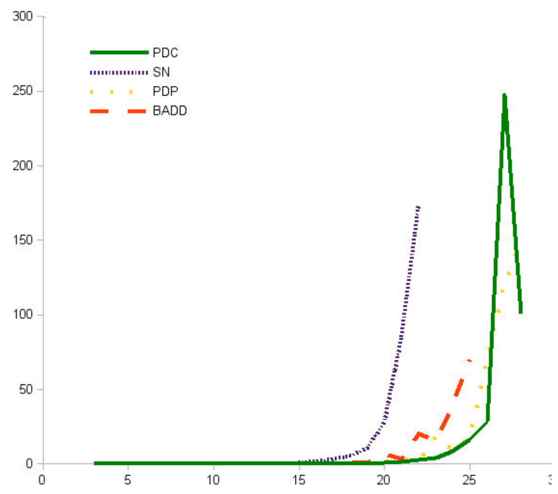


Figure 1: The left hand figure plots the best solver for pairs n and L ($n = 3 \cdots 30$, $L = 3 \cdots 2n$). The right hand figure shows the average solving time, in second, of the engines which solved all the 100 instances in 10 minutes timeout, for $n = L = \in 3 \cdots n$.

be 10 minutes. During our experiments, we noticed that the sorting network encoding in MiniSAT+ incorrectly announces some unsatisfiable instances to be satisfiable (an example of which is the following constraint). We did not investigate the reason of this issue in the source code of MiniSAT+, and all the reported timings are using the broken code.

$$5x_1 + 7x_2 + 1x_3 + 5x_4 = 9.$$

In our experiments, we generated 100 instances for $n \in \{3..30\}$ and $L \in \{3..2 * n\}$. We say a solver wins on a set of instances if it solves more instances than the others and in the case of a tie, we decide the winner by looking at the average running time. The instances on which each solver performed the best are plotted on Figure 1. As the Sorting Network solver was never a winner on any of the sets, it did not show up in the graph.

One can observe the following patterns from the data presented in Figure 1:

1. For $n < 15$, all solvers successfully solve all the instances.
2. Sorting network fails to solve all the instances where $n = 20$.
3. BADD solves all the instances when $n = L = 24$ in a reasonable time, but it suddenly fails when the $n(L)$ gets larger.
4. For large enough n ($n < 15$) BADD is the winner only when L is small.
5. For large enough n ($n < 15$) either PDC or PDP is the best performing solver.

Conclusion and Future Work

We presented a method for translating Pseudo-Boolean constraints into CNF. The size of produces CNF is polynomial with respect to the input size. We also showed that for exponentially many instances, the produced CNF is arc-consistent. The number of arc-consistent instances, for our encodings, is much bigger than that of the existing encodings.

In our experimental evaluation section, we described a set of randomly generated number partitioning instances with two parameters, n and L, where n describes the size of our set and 2^L is the maximum value in the set. The experimental result suggests that Prime.DP and Prime.DC encoding outperform Binary Adder and Sorting Network encodings.

Future work

The upper bounds for our encodings, presented in Table 2, are not tight. We hope to improve these and give the exact asymptotic sizes. Further experimental evaluation is needed to determine the relative performance of the various methods on more practical instances, and on instances with larger numbers of variables. Finally, we hope to develop heuristics for automatically choosing the best encoding to use for any given PB constraint.

References

Aavani, A. 2011. Translating pseudo-boolean constraints into cnf. *CoRR* abs/1104.1479.

Aloul, F.; Ramani, A.; Markov, I.; and Sakallah, K. 2002. PBS: a backtrack-search pseudo-boolean solver and optimizer. In *Proceedings of the 5th International Symposium on Theory and Applications of Satisfiability*, 346–353. Citeseer.

Bailleux, O.; Boufkhad, Y.; and Roussel, O. 2009. New Encodings of Pseudo-Boolean Constraints into CNF. *Theory and Applications of Satisfiability Testing-SAT 2009* 181–194.

Borgs, C.; Chayes, J.; and Pittel, B. 2001. Phase transition and finite-size scaling for the integer partitioning problem. *Random Structures & Algorithms* 19(3-4):247–288.

Ding, C.; Pei, D.; and Salomaa, A. 1996. *Chinese remainder theorem: applications in computing, coding, cryptography*. World Scientific Publishing Co., Inc. River Edge, NJ, USA.

Eén, N., and Sorensson, N. 2006. Translating pseudo-boolean constraints into SAT. *Journal on Satisfiability, Boolean Modeling and Computation* 2(3-4):1–25.

Eén, N. 2005. *SAT Based Model Checking.* Ph.D. Dissertation, Department of Computing Science, Chalmers University of Technology and Goteborg University.

Farhi, B., and Kane, D. 2009. New results on the least common multiple of consecutive integers. In *Proc. Amer. Math. Soc*, volume 137, 1933–1939.

Gent, I. P., and Walsh, T. 1998. Analysis of heuristics for number partitioning. *Computational Intelligence* 14(3):430–451.

Sheini, H., and Sakallah, K. 2006. Pueblo: A hybrid pseudo-boolean SAT solver. *Journal on Satisfiability, Boolean Modeling and Computation* 2:61–96.

Tseitin, G. 1968. On the complexity of derivation in propositional calculus. *Studies in constructive mathematics and mathematical logic* 2(115-125):10–13.