Journal on Satisfiability, Boolean Modeling and Computation 2 (2006) 191-200

Research Note

A Translation of Pseudo-Boolean Constraints to SAT

Olivier Bailleux olivier.bailleux@u-bourgogne.fr
Université de Bourgogne — LERSIA
Avenue Alain Savary — BP 47870
F-21078 Dijon Cedex — France

Yacine Boufkhad yacine.boufkhad@liafa.jussieu.fr
LIAFA — Université Denis Diderot — Case 7014
2, place Jussieu
F-75251 Paris Cedex 05 — France

Olivier Roussel olivier.roussel@cril.univ-artois.fr
CRIL
rue de l'Université — SP 16
F-62307 Lens Cedex — France

Abstract

This paper introduces a new CNF encoding of pseudo-Boolean constraints, which allows unit propagation to maintain generalized arc consistency. In the worst case, the size of the produced formula can be exponentially related to the size of the input constraint, but some important classes of pseudo-Boolean constraints, including Boolean cardinality constraints, are encoded in polynomial time and size. The proposed encoding was integrated in a solver based on the zChaff SAT solver and submitted to the PB05 evaluation. The results provide new perspectives in the field of full CNF approach of pseudo-Boolean constraints solving.

KEYWORDS: *pseudo-Boolean, SAT translation*

Submitted October 2005; revised January 2006; published March 2006

1. Introduction

This paper proposes a new way to encode pseudo-Boolean constraints into CNF formulae.

The proposed technique allows unit propagation, which is the basic filtering technique implemented on all DLL-based SAT solvers, to enforce generalized arc-consistency. This guarantees that the SAT solver will achieve at least the same deductions on the resulting clauses as a minimalist enumerative solver would do on the original constraints.

In the worst case, the size of the resulting CNF formula is exponentially related to the size of the encoded constraint. Yet, Boolean cardinality constraints - and some other classes of pseudo-Boolean constraints that will be detailed in section 3 - are encoded in polynomial time and size.

Section 2 describes the proposed CNF encoding of pseudo-Boolean constraints and proves its ability to allow unit propagation to enforce arc-consistency. Section 3 gives some complexity results relating the size of the resulting CNF formulae to the size of the encoded constraint. Section 4 describes the implementation of the proposed encoding based

on the zChaff SAT solver. Section 5 presents the experimental results obtained at the Pseudo-Boolean evaluation of the SAT 2005 competition. Section 6 presents other works related to CNF encoding of pseudo-Boolean and arc-consistent CNF encoding of constraint satisfaction problems. Section 7 concludes this paper.

2. Proposed encoding

2.1 Notations

Let V be a set of variables where each variable v can take a value among a finite domain d_v. An *assignation* I of V is a function mapping a value of d_v to each variable v in V. Given an assignation I on V, a variable v in V and a value x in d_v, $v =_I x$ means that I gives the value x to v. The notation $v = x$ is used instead of $v =_I x$ whenever there is no ambiguity. A *constraint* Q on V is the specification of a set of legal assignations of V. Each of these legal assignations is said to *satisfy* Q. Other assignations of V are said to *falsify* Q. Given a value x of a variable v in V, a *support* of x with respect to Q is an assignation of V satisfying Q and such that $v = x$. We will consider pseudo-Boolean constraints specified as $w_1x_1 + w_2x_2 + \ldots + w_nx_n <= K$, where $w_1, \ldots, w_n$ are integers called *weights*, K is an integer called *bound*, and $x_1, \ldots, x_n$ are literals, i.e., Boolean variables or negated Boolean variables.

Let v be a Boolean variable. The literal v is said to be *fixed* to 0 (resp. 1) if $d_v = \{0\}$ (resp. $\{1\}$). The literal $\overline{v}$ is said to be *fixed* to 0 (resp. 1) if $d_v = \{1\}$ (resp. $\{0\}$). The literals v and $\overline{v}$ are said to be *free* if $d_v = \{0, 1\}$.

2.2 Generalized Arc Consistency

Let Q be a constraint on a set V of variables. Q is said to be *generalized arc consistent* if and only if for each domain D of any variable in V, every value of D has a support with respect to Q.

The generalized arc consistency of a pseudo-Boolean constraint $Q : w_1x_1 + w_2x_2 + \ldots + w_nx_n <= K$ can be enforced in the following way. If the sum of the weights of the literals fixed to 1 exceeds K then Q cannot be satisfied; else for each free literal x_i, if the sum of the weights of the literals fixed to 1 exceeds $K - w_i$ then x_i must be fixed to 0. The principle of our encoding is to use some clauses and additional variables to allow unit propagation to detect all the literals that have to be fixed with respect to generalized arc consistency.

2.3 Encoding method

Without loss of generality, we consider pseudo-Boolean constraints (PBC for short) of the type $w_1x_1 + w_2x_2 + ... + w_nx_n \leq K$ where the weights are ordered i.e. $w_1 \leq w_2 \leq ... \leq w_n$. The triple $< W_n, X_n, K >$ where W_n is the ordered tuple $(w_1, w_2, ..., w_n)$ and X_n is the tuple of Boolean variables $(x_1, x_2, ..., x_n)$ fully characterizes the corresponding PBC. For the proposed encoding, there is a need to consider, for some $1 \leq i \leq n$, the partial tuples $W_i = (w_1, w_2, ..., w_i)$ and $X_i = (x_1, x_2, ..., x_i)$. For some bound b, the triple $< W_i, X_i, b >$ represents the PBC $w_1x_1 + w_2x_2 + ... + w_ix_i \leq b$. When the tuples W_n and X_n are fixed, a triple $< W_i, X_i, b >$ representing a PBC is defined with no ambiguity by the integer i and the bound b. To build the proposed CNF encoding, some new Boolean variables representing

the satisfaction of such constraints are introduced beside the Boolean variables in X_n. These variables are denoted $D_{i,b}$ and are equivalent to the PBC $< W_i, X_i, b >$ in the sense $D_{i,b} = 1$ if and only if $< W_i, X_i, b >$ is satisfied. $D_{n,K}$ is then the variable representing $< W_n, X_n, K >$ and the correctness of the encoding is conditioned by the fact that an assignation satisfies $< W_n, X_n, K >$ if and only if it satisfies the encoded CNF formula and fixes $D_{n,K} = 1$.

Some of these Boolean variables have a special status and are said to be *terminal variables*. These are the variables $D_{i,b}$ such that $b \leq 0$ or $b \geq \sum_{j=1}^{i} w_j$.

The CNF encoding is built in the following way. It starts with a set of variables containing the variables of the pseudo-Boolean constraint x_i and the variable $D_{n,K}$. The variables x_i of the PBC are marked. At each step, an unmarked variable $D_{i,b}$ is considered. If $D_{i,b}$ is not a terminal variable then the two variables $D_{i-1,b}$ and $D_{i-1,b-w_i}$ are added to the set of variables if they are not already in it and the following four clauses are added to the set of clauses:

$$\overline{D}_{i-1,b-w_i} \vee D_{i,b}, \overline{D}_{i,b} \vee D_{i-1,b}, \overline{D}_{i,b} \vee \overline{x}_i \vee D_{i-1,b-w_i}, \overline{D}_{i-1,b} \vee x_i \vee D_{i,b}$$

After this, $D_{i,b}$ is marked so it will not be considered further.

If $D_{i,b}$ is a terminal variable then, by definition either $b \leq 0$ or $b \geq \sum_{j=1}^{i} w_j$. We consider separately the case $b \neq 0$. In this case $D_{i,b}$ must be fixed as follows:

$$D_{i,b} = \begin{cases} 0 & \text{if } b < 0. \text{ The unit clause } \overline{D}_{i,b} \text{ is added to the formula} \\ 1 & \text{if } \sum_{j=1}^{i} w_j \leq b. \text{ The unit clause } D_{i,b} \text{ is added to the formula} \end{cases}$$

When $b = 0$, every variable in the constraint must be fixed to 0. To achieve this, for every $1 \leq j \leq i$, the binary clauses $\overline{D}_{i,0} \vee \overline{x}_j$ are added together with the clause $x_1 \vee x_2 \vee ... \vee D_{i,0}$.

The procedure stops when there are no more unmarked variables.

Example 1. *The following example illustrates the preceding encoding of PBC:* $2x_1 + 3x_2 + 4x_3 \leq 6$. *In Figure 1 the way variables are added is represented by a graph where every non terminal node have two children representing the two variables added each time an unmarked non terminal variable is considered. For this example, the graph is a tree but in the general case some node may have more than one parent.*

The formula is $\mathcal{C} = \{\overline{D}_{2,2} \vee D_{3,6}, \overline{D}_{3,6} \vee D_{2,6}, \overline{D}_{3,6} \vee \overline{x}_3 \vee D_{2,2}, \overline{D}_{2,6} \vee x_3 \vee D_{3,6}, D_{2,6}, \overline{D}_{1,-1} \vee D_{2,2}, \overline{D}_{2,2} \vee D_{1,2}, \overline{D}_{2,2} \vee \overline{x}_2 \vee D_{1,-1}, \overline{D}_{1,2} \vee x_2 \vee D_{2,2}, D_{1,2}, \overline{D}_{1,-1}\}$ *which makes* $D_{3,6} = 1$ *only if at least one of* x_2 *or* x_3 *is equal to* 0.

To encode a PBC, a set of clauses is built as described in the procedure then, to ensure that the PBC is satisfied, the unit clause $D_{n,K}$ is added requiring from any assignation satisfying the set of clauses, to satisfy also the PBC.

This encoding is very close to those using a BDD and translating it into clauses [9]. But the latter produce only clauses of length 3 while the proposed translation produces a mixture of binary and ternary clauses allowing a better propagation during solving.

Clearly, the precedent procedure terminates because at each step the subscript i of newly added variables $D_{i,b}$ strictly decreases and necessarily, the variables of type $D_{0,b}$ are terminal variables. At the end of this procedure, a set of variables $\mathcal{V}$ and a set of clauses $\mathcal{C}$ are obtained.

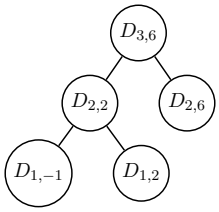

Figure 1. Variables added for encoding the constraint $2x_1 + 3x_2 + 4x_3 \leq 6$

The correctness of the encoding is conditioned by the fact that for any assignation to the variables of $\mathcal{V}$ that satisfies $\mathcal{C}$ and fixes $D_{n,K} = 1$, the restriction of this assignation to the variables X_n satisfies also the pseudo-Boolean constraint $< W_n, X_n, K >$. Moreover any assignation to the variables of X_n that satisfies $< W_n, X_n, K >$ can be extended to an assignation to the variables of $\mathcal{V}$ that satisfies $\mathcal{C}$ and fixes $D_{n,K} = 1$. The connection between BDD and this encoding makes the proof of correction trivial.

Beyond the correctness, we require from the encoding to restore the generalized arc consistency through unit propagation as defined in section 2.2. Namely, considering a partial assignation to the variables of X_n that violates the PBC, the unit propagation generates an empty clause in the formula $\mathcal{C} \cup \{D_{n,K}\}$. Moreover, for any partial assignation to the variables of X_n such that if some literal x_p is fixed to 1 then the PBC is violated, the unit propagation assigns 0 to x_p in the formula $\mathcal{C} \cup \{D_{n,K}\}$. This is the case in our encoding.

Theorem 2.1. *The encoding restores the generalized arc consistency through unit propagation.*

Proof. 1. $n = 1$. Trivial.

2. Suppose that the property holds for any $i \leq n-1$ and let us prove that it still holds for n. Let $\mathcal{C}_0$ the set of clauses encoding the PBC $w_1x_1 + ... + w_{n-1}x_{n-1} \leq K$ and $\mathcal{C}_1$ the set of clauses encoding the PBC $w_1x_1 + ... + w_{n-1}x_{n-1} \leq K - w_n$. Remark that by construction $\mathcal{C} = \mathcal{C}_0 \cup \mathcal{C}_1 \cup \{\overline{D}_{n-1,K-w_n} \vee D_{n,K}, \overline{D}_{n,K} \vee D_{n-1,K}, \overline{D}_{n,K} \vee \overline{x}_n \vee D_{n-1,K-w_n}, \overline{D}_{n-1,K} \vee x_n \vee D_{n,K}\}$. After simplification, $\mathcal{C} \cup \{D_{n,K}\} = \mathcal{C}_0 \cup \mathcal{C}_1 \cup \{D_{n-1,K}, \overline{x}_n \vee D_{n-1,K-w_n}, D_{n,K}\}$.

   Consider a partial assignation B that violates the PBC $< W_n, X_n, K >$ such that x_n is not fixed or $x_n = 0$ by this partial assignation. Because the unit propagation restores generalized arc-consistency for $n-1$, unit propagation applied to $\mathcal{C}_0$ fixes $D_{n-1,K} = 0$ which contradicts the clause $\overline{D}_{n,K} \vee D_{n-1,K}$ in $\mathcal{C} \cup \{D_{n,K}\}$. If $x_n = 1$ then the PBC $< W_{n-1}, X_{n-1}, K - w_n >$ is violated by this partial assignation. Then $D_{n-1,K-w_n} = 0$ which contradicts the clause $\overline{D}_{n,K} \vee \overline{x}_n \vee D_{n-1,K-w_n}$ in $\mathcal{C} \cup \{D_{n,K}\}$.

   Consider now a partial assignation B such that there exists some free literal x_p such that if $x_p = 1$ the PBC is violated. Suppose first that $p \neq n$. If $x_n = 0$ or x_n is free then $x_p = 1$ violates $< W_{n-1}, X_{n-1}, K >$ under B then unit propagation fixes $x_p = 0$ in $\mathcal{C}_0 \cup \{D_{n-1,K}\}$. If $x_n = 1$ then $x_p = 1$ violates $< W_{n-1}, X_{n-1}, K - w_n >$ under B then unit propagation fixes $x_p = 0$ in $\mathcal{C}_1 \cup \{D_{n-1,K-w_n}\}$ ($\{D_{n-1,K-w_n}\}$ comes from the clause $\overline{D}_{n,K} \vee \overline{x}_n \vee D_{n-1,K-w_n}$).

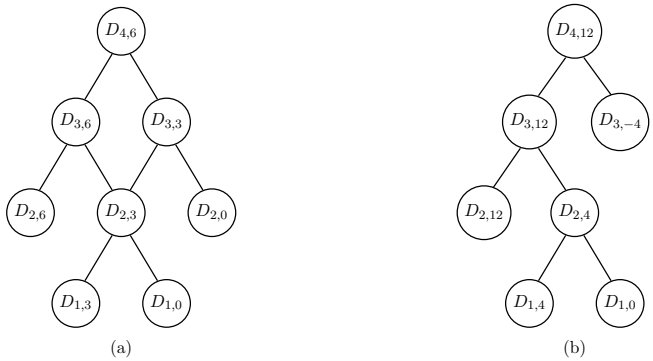

Figure 2. (a) Variables introduced to encode $3x_1+3x_2+3x_3+3x_4 \leq 6$. (b) Variables introduced to encode $2x_1 + 4x_2 + 8x_3 + 16x_4 \leq 12$

If $p = n$ then the PBC $< W_{n-1}, X_{n-1}, K - w_n >$ is necessarily violated by B. Since the encoding is arc consistent for $n-1$, when applied to $\mathcal{C}_1$ unit propagation fixes $D_{n-1,K-w_n} = 0$. In $\mathcal{C} \cup \{D_{n,K}\}$ the clause $\overline{x}_n \vee D_{n-1,K-w_n}$ fixes $x_n = 0$.

□

3. Complexity of the translation

3.1 Some polynomial cases

The size of the encoding is measured in terms of the number of variables.The number of clauses is related by a constant factor to the number of variables.

At first sight, the procedure described in the previous section generates an exponential number of variables: at each step a non-terminal variables creates two variables that will in turn create two other variables each and so on. This is not true for terminal variables and for variables already considered in the procedure. When a terminal variable is met, it is said a *cut* in the procedure and when a variable already in the set of variables is met, it is said a *merge* in the procedure. By the cuts and merges, the size of encodings can be polynomial in some cases. In the following, two restrictions of the general PBC producing a polynomial size encoding are identified:

- The weights w_i are integers bounded by a polynomial in n: $P(n)$. The potential number of variables $D_{i,b}$ for some i is 2^{n-i} but because of merges this number reduces to a polynomial as shown in the following. Indeed, since the variables $D_{i,b}$ for some i are such that $m \leq b \leq M$ where m is at least equal to $K - \sum_{j=0}^{i} w_{n-j}$ and M is at most K, b can take at most $M - m$ different values and then it can take at most $\sum_{j=0}^{i} w_{n-j}$ different values, which is bounded by $(n-i)P(n)$. Since there is n different possible values of i, the total number of variables is bounded by a polynomial in n. Figure 2(a) shows an example for this case.

- The weights are $w_i = a^i$ where $a \geq 2$. In this case, for every non terminal variable $D_{i,b}$ considered in the procedure, at least one of the variables $D_{i-1,b}$ or $D_{i-1,b-a^i}$ is a terminal variable. This is true because $\sum_{j=0}^{i-1} a^j < a_i$. Either $b \geq a^i$ and then

$\sum_{j=0}^{i-1} a^j < b$ and then $D_{i-1,b}$ is a terminal variable or $b < a^i$ and in this case $D_{i-1,b-a^i}$ is a terminal variable. Consequently there is a cut each time a variable is considered in the procedure as shown in the example of Figure 2(b).

3.2 An exponential case

Unfortunately, it is possible to use a sequence of weights which will give a tree with branches of length $\Omega(n)$ and with no possible merge of nodes (which implies a tree of size $\Omega(2^n)$). The idea here is simply to combine a constant sequence with a geometric sequence. Let n be the length of the pseudo-Boolean constraint Q and let $w_i = a + b^i$ such that $a = b^{n+2}$. The key point is that the geometric term must be negligible compared to the constant term, that is $\sum_{i=0}^{n} b^i < a$. For simplicity, we will choose $b = 2$. Note that in this case, $w_i = 2^{n+2} + 2^i$ which is not bounded by a polynomial in n. Let's choose $K = a.n/2 = n.2^{n+1}$.

A terminal node is reached when we get a term $D_{i,k}$ such that $k \leq 0$ or $k \geq \sum_{j=1}^{i} w_j$. Because the constant term is predominant, the first condition cannot be met before $i = K/a = n/2$. The earliest case where the second condition can be satisfied is when k remains equal to K. We have $\sum_{j=1}^{i} w_j = \sum_{j=1}^{i}(a + b^j) = a.i + \sum_{j=1}^{i} b^j \geq a.i$. Therefore, the earliest case where the second condition can be met is when $a.n/2 = a.i$ which means $i = n/2$. We can conclude that each branch is at least of length $n/2$.

Furthermore, in the encoding, each node of the tree holds the term $D_{i,k}$ which corresponds to the inequation $\sum_{j=1}^{i} w_j.x_j \leq K - \sum_{j \in S} w_j$ where S is a subset of $[i+1..n]$. One key point is that in the binary representation of $K - \sum_{j \in S} w_j$, the n least significant bits directly correspond to the indices in S. Therefore, these n least significant bits of the right term are necessarily different from one node to another. For this reason, no node can be merged. Because of this and since branches are of length at least equal to $n/2$, the size of the tree is at least $2^{n/2}$ and the encoding of this particular constraint is of exponential size.

4. Implementation: the `pb2sat+zchaff` solver

The proposed encoding of pseudo-Boolean constraints into SAT was implemented in a solver named `pb2sat+zchaff`. As indicated in its name, this solver uses the zChaff solver [8] to actually solve the SAT problem. This SAT solver might not be the most efficient solver today but our goal is less to get a faster PB solver than to experiment with the new encoding.

4.1 Encoding of pseudo-Boolean constraints

Each pseudo-Boolean constraint is first normalized, that is transformed into a less or equal inequality, with all weights positive and sorted. Constraints which are clauses are directly sent to the SAT solver. Other constraints are encoded into a set of clauses over new propositional literals $D_{i,b}$ which are true if and only if the inequation $\sum_{j=0}^{i} w_j.x_j \leq b$ holds. The clauses which are sent to the SAT solver are the ones described in section 2. The only difference with the description of section 2 is that the implemented procedure is recursive and enumerates the $D_{i,b}$ in a depth-first manner while section 2 presents an iterative enumeration (in a breadth-first manner).

4.2 Optimization process

Most of the pseudo-Boolean solvers available are able to find a solution which optimizes the value of an objective function (either maximize or minimize its value). Therefore a minimal support for optimization was added to the solver in order to be able to compare our solver to others. Our support is minimal because we didn't want to modify the SAT solver in any way. A more complete support would imply a change in its heuristics in order to prefer models with least values of the objective function (in the case of minimization).

Each objective function to optimize can be normalized so that it takes only positive values. Let $f(x)$ be the normalized objective function to minimize. Our solver first runs a satisfiability test ignoring this function. If no solution could be found, the solver directly answers that the formula is unsatisfiable. Otherwise, let M_0 be the model found by this first satisfiability test which gives value $f(M_0)$. To find the best solution, the optimization process proceeds by binary search on the value of the objective function in $[0..f(M_0)-1]$. For each iteration of the binary search algorithm, we have to encode that $f(x) \leq M$ where M is the middle of the interval. However, we'd rather avoid the costly encoding of this new constraint at each iteration.

To this end, we swap to a relaxed definition of M such that $M = f(M_0) - 1 - B$ where B can be written as $B = \sum b_i.2^i$. By introducing b_i as new propositional variables, we can write a single pseudo-Boolean constraint $f(x) - \sum b_i.2^i \leq f(M_0) - 1$ which we can use to enforce an optimization constraint for each iteration of the binary search algorithm. All the algorithm does is to try to set each b_i to true (starting with the most significant bit). If the formula is still satisfiable, then b_i is definitively set to true, otherwise it is definitively set to false. We then proceed with the next bit.

One problem though it that objective functions may contain thousands of literals and their encoding may get too large. To fix this problem, when the solver is faced with a large objective function, it iteratively searches for the first solution with a value of the objective function in $[i.C, (i+1).C]$ where C is a constant which corresponds to a upper limit on the size of the encoding. Once this first interval is found, the solver proceeds by binary search to find the optimum solution in this interval.

5. Experimental results

The `pb2sat+zchaff` solver was submitted to the 2005 evaluation of pseudo-Boolean solvers (PB05) [7]. Among the 8 submitted solvers, two of them were directly based on a translation to SAT and 4 of them had support for big integers (i.e. weights with an arbitrary number of digits). Table 1 reports the global results of the evaluation. It counts the different answers of the solvers (unsatisfiable, optimum solution found, a solution found but not necessarily the optimum or unknown). TO (Time Out) means that the solver exceeded the time limit and was required to output the best solution it had so far and to stop. MO (Mem. Out.) means that the solver exceeded the memory limit. EC means that the solver exited with a different exit code that the one that was expected.

The difference in the number of instances handled by each solver results from the fact that some solvers were not able to handle big integers or could suffer from other integer overflow problems. For this reason, solvers cannot be directly compared.

Table 1. Results of the last phase of the PB05 evaluation for all categories

Solver Name	#benchs	unsat.	opt.	satisfiable			unknown				sig.	no	wrong		
					TO	MO		TO	MO	EC	caught	cert.	cert.	opt.	unsat.
bsolo	1172	136	196	326	26	1	391	10	86	0	0	0	0	0	0
galena	690	49	103	157	0	0	319	0	0	0	51	0	0	0	*11*
minisat+	1172	156	226	0	0	0	0	444	49	11	0	*286*	0	0	0
PBS4	690	71	166	28	0	0	0	425	0	0	0	0	0	0	0
Pueblo	690	71	194	298	0	0	48	78	0	0	0	0	*1*	0	0
sat4jpseudo	1172	149	142	29	489	0	6	353	0	0	1	*3*	0	0	0
vallst_0.9.258	499	48	131	33	0	0	0	277	0	0	0	0	*3*	0	*7*
pb2sat+zchaff	1172	60	161	36	173	12	0	193	157	372	8	0	0	0	0

Even if `pb2sat+zchaff` is clearly not among the best solvers of this evaluation, its results remain comparable to the other solvers. minisat+ which is also based on a translation to SAT has particularly good results. Interestingly, it was discovered after the evaluation that minisat+ uses almost the same encoding as ours as well as two other encodings to which it switches when the first encoding gets too large.

Another point to notice is the large number of out of memory (157) or unexpected exit code (372). Among the 372 unexpected exit code, 6 are caused by an exception which is raised to indicate that a constraint is trivially unsatisfiable. As this exception was unfortunately not correctly handled, the solver exited without answering UNSAT. The other 366 unexpected exit codes were caused by a bad allocation exception which is just another manifestation of memory exhaustion. Among the 523 instances (157+366) where the solver exhausted the memory, there are only 105 cases where the memory exhaustion occurred after zChaff was run. In the 418 other cases, the solver ran out of memory during the translation process because it encountered one or more constraints with an exponential translation.

These results clearly suggest that the proposed translation cannot be a standalone method to solve pseudo-Boolean constraint. But at the same time, the global results of the solver indicate that the translation is an effective approach in a number of cases. The conclusion is that a good approach to solving pseudo-Boolean constraints is maybe to mix a clausal translation together with a native handling of pseudo-Boolean constraint. Constraints which can be translated to a low number of clauses are probably best translated to SAT (because this removes the overhead of dealing with weights, which can be important especially with big integers). Constraints whose translation to SAT is too big should be handled as pseudo-Boolean constraints.

6. Related works

In [5], Gent shows the importance of enforcing arc-consistency in CNF encoding of binary constraints. He compares two encodings, namely direct encoding and support encoding. Contrarily to direct encoding, support encoding, introduced in [6], allows unit propagation to maintain arc consistency of the encoded constraints. Using the Chaff SAT solver [8], Gent experiments these two encodings on random CSP instances on which support encoding clearly outperforms direct encoding.

In [2], the principle of an encoding allowing unit propagation to maintain arc-consistency is applied to Boolean cardinality constraints. Given a Boolean cardinality constraint on n variables, the proposed encoding produces a CNF formula of $O(n.\log n)$ variables and $O(n^2)$ clauses. This encoding is compared to Warner's encoding, which reduces any pseudo-Boolean constraint of length n to a CNF formula of $O(n)$ clauses and variables. Although Warner's encoding produces a smaller formula, it does not allow unit propagation to maintain generalized arc-consistency on the encoded constraint. Experimental results obtained with the zChaff solver [8] show that the encoding of [2] largely outperforms Warner's one on some discrete tomography problems. [3] shows that this encoding can be used for solving optimization problems and can even be competitive with the dedicated pseudo-Boolean solvers PBS [1] and OPBDP [4] on some maxsat and maxones instances.

In [10], Sinz proposes two other encodings for Boolean cardinality constraints. The first one is based on a sequential counter using a binary representation of integers. Like the encoding presented in [2], it allows unit propagation to maintain generalized arc-consistency. The second one is based on a parallel counter using a binary representation of integers. Like Warner's encoding, it does not allow unit propagation to maintain generalized arc consistency. No experimental results are given for these two encodings.

Concomitantly to the present work, [9] proposes another pseudo-Boolean solver named minisat+, which is based on translating pseudo-Boolean constraints into CNF. Preferably, minisat+ uses a translation technique based on BDD (Binary Decision Diagram), which produce a CNF formula logically equivalent to the one produced with our translation technique, with the same advantage i.e., unit propagation enforces generalized arc consistency, and the same drawback, i.e., the size of the resulting formula can be exponentially related to the size of the original constraints. If this preferred translation technique produces too many clauses, another technique, based on sorting networks, is tried. At last, if this second technique produces too many clauses, a translation based on an adder network is used, which guarantees that a polynomial number of clauses is generated.

We do not know any reference of polynomial size CNF encoding allowing unit propagation to enforce generalized arc consistency. We would like to cash in on this paper for opening the question of the existence of such an encoding.

7. Conclusion

We proposed a new CNF encoding of pseudo-Boolean constraints. Compared with the reference encoding of Warner, this new encoding presents a benefit and a drawback. The benefit is that it allows unit propagation to maintain arc consistency of the encoded constraints. The drawback is that in the worst case, the encoded formula can have a size exponentially related to the size of the encoded formula. Yet, some useful classes of pseudo-Boolean constraints, like Boolean cardinality constraints, can be encoded in polynomial time and space. This encoding was implemented in a solver based on the zChaff SAT solver and presented to the PB05 competition. The obtained results show that the proposed encoding is not suitable to encode any pseudo-Boolean constraint. However, these results certainly prove that large classes of pseudo-Boolean constraints can advantageously be encoded in SAT to avoid the cost of dealing with weights. These results show that the CNF encoding is not a unreasonable option for solving pseudo-Boolean constraints. They also contribute

to open the question of the existence of a polynomial CNF encoding that would really put the resolution of pseudo-Boolean constraints in the field of application of pure SAT solvers. An important objective for future researches is either to find a polynomial arc-consistent CNF encoding for pseudo-Boolean constraints, or to prove that such an encoding does not exist.

8. Acknowledgments

The authors are grateful to the anonymous reviewers for their comments. We would like to thank Dominique Rossin for helpful discussions.

References

[1] F. Aloul, A. Ramani, I. Markov, and K. Sakallah. PBS: A Backtrack Search Pseudo-Boolean Solver. In *Symposium on the Theory and Applications of Satisfiability Testing (SAT 2002)*, pages 346–353, 2002.

[2] O. Bailleux and Y. Boufkhad. Efficient CNF encoding of boolean cardinality constraints. In *Proceedings of the 9th International Conference on Principles and Practice of Constraint Programming, CP 2003*, volume **2833**, pages 108–122. LNCS, 2003.

[3] O. Bailleux and Y. Boufkhad. Problem encoding into SAT : the counting constraints case. In *Proceedings of The Seventh International Conference on Theory and Applications of Satisfiability Testing (SAT 2004)*, 2004.

[4] P. Barth. A Davis-Putnam based Enumeration Algorithm for Linear Pseudo-Boolean Optimization, Technical Report MPI-I-95-2-003. Technical report, Max-Planck-Institut Für Informatik, 1995.

[5] I.P. Gent. Arc Consistency in SAT. In *Proceedings of the Fifteenth European Conference on Artificial Intelligence (ECAI 2002)*, 2002.

[6] S. Kasif. On the parallel complexity of discrete relaxation in constraint satisfaction networks. *Artificial Intelligence*, **45**:275–286, 1990.

[7] V. Manquinho and O. Roussel. The Pseudo Boolean Evaluation 2005, 2005. http://www.cril.univ-artois.fr/PB05.

[8] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an Efficient SAT Solver . In *39th Design Automation Conf.*, pages 530–535, June 2001.

[9] N. Eén and N. Sörensson. Translating Pseudo-Boolean Constraints into SAT. *Journal on Satisfiability, Boolean Modeling and Computation*, 2006. This issue.

[10] Carsten Sinz. Towards an Optimal CNF Encoding of Boolean Cardinality Constraints. In *Proceedings of the 11th International Conference on Principles and Practice of Constraint Programming, CP 2005*, 2005.