ECAI 2010
H. Coelho et al. (Eds.)
IOS Press, 2010

doi:10.3233/978-1-60750-606-5-867

Solving Pseudo-Boolean Modularity Constraints

Carlos Ansótegui and **Ramón Béjar** and **Cèsar Fernández** and **Francesc Guitart** and **Carles Mateu** [1]

Abstract. This paper introduces new solving strategies for the resolution of Pseudo-Boolean Modularity (PBMod) constraints. In particular, we deal with modular arithmetic constraints on Boolean variables. On the one hand, we analyze translations to Pseudo-Boolean (PB) constraints and apply PB solvers. We also look at those PB solvers that have shown that a transformation to the SAT problem can be an effective solving strategy for PB problems. Among the existing translation techniques we focus on the encoding based on a network of sorters. We extend this encoding technique to generate directly a SAT formula from the PBMod constraints. We compare our approach to other standard techniques such as Satisfiability Modulo Theories (SMT) solvers with support for the Quantifier Free Linear Integer Arithmetic (QF_LIA) theory and the GLPK package for Mixed Integer Programming. In order to conduct our experimental investigation we present a generator of random PBMod constraints and study the impact of the several parameters on the hardness of the instances.

1 Introduction

Modular arithmetic appears in many fields such as number theory, group theory, ring theory, abstract algebra, cryptography and computer science. In computer science, modular arithmetic is applied for several purposes such as; bit-wise operations and other operations involving fixed-width, cyclic data structures, systems with parity constraints, resolution of Diophantine equations, etc. Linear modular arithmetic equations are used in several applications. To mention some examples: in [11] the addition of modularity constraints is used to get bounds on the number of solution in CSP problems; in [10] linear modular arithmetic constraints are used to prune the search space in an algorithm for optimally solving bin packing problems; and in some frequency assignment problems, like in [7, 14], frequency domains for sites are organized in groups that are congruence equivalence classes, so that adjacency constraints between pairs of frequencies can be defined with modular arithmetic.

In this work we focus our study on modularity constraints of the following form: $\sum_i c_i \cdot l_i \equiv r \pmod{m}$ where c_i, r and m are positive integers and l_i is a literal of Boolean variables. We will refer to this constraint as a *pseudo-Boolean Modular (PBMod) constraint*. In particular, in our work we are interested in solving efficiently formulas which are conjunctions of PBMod constraints. The ultimate goal is its integration into a more general purpose framework such as Satisfiability Modulo Theories (SMT) [15].

A straightforward method to deal with PBMod Constraints is through their translation to PB constraints of the form: $\sum_i c_i \cdot l_i \rhd k$ where k, c_i are integer positive constants, l_i is a literal of Boolean variables and $\rhd$ is one of the operators of $\{=, <, \leq, >, \geq\}$. Then, it is possible to use a PB solver with an empty objective function to solve the problem. Some PB solvers also employ, as their solving strategy, the translation to the SAT problem and the usage of SAT solvers. There are several works on the translation of PB constraints to SAT [9, 5, 16, 4] and cardinality constraints (PB constraints where all coefficients are equal to 1) to SAT [3, 18, 2].

In [9] several encoding schemes into the SAT problem are analyzed: network of adders, Binary Decision Diagrams (BDD), and network of sorters. As we show in the experimental analysis that we have carried out, for the problem we address in this work, the encodings based on a network of sorters are the best performing ones. It is well known that an eager transformation may lose some desirable properties/information from the original model. Therefore, we introduce in this work a specialized translation for the PBMod constraints into the SAT problem extending the encoding based on networks of sorters.

We have also compared this approach with other standard solving techniques. On the one hand, we have analyzed the performance of SMT Solvers. SMT solvers rely on a tight integration of two components: a theory solver that can handle conjunctive constraints, and a DPLL-based SAT engine that does the search without knowing the semantics of the literals. We have focused on those solvers that were the best performing ones for the Quantifier Free formulae over Integer Linear Arithmetic theory (QF_LIA) of the SMT competition 2009: Sateen(v3.5) [2] and Yices (v2.0). We have also used the software package GLPK (GNU Linear Programming Kit) for solving our Integer Linear Programming (ILP) formulations of the problem.

In order to conduct our experimental investigation we have developed a generator of random PBMod constraints. As in many problems we indeed observe the existence of a phase transition region, i.e. a region where there is an abrupt descent on the ratio of satisfiable instances. We have also studied how the different generator parameters such as: the values of the remainder, the modulo, the coefficients of the variables, and the length of the modular constraints impact on the hardness of the instances and the performance of the different solving strategies. Such study allows to select the appropriate solving technique depending on the structure of the problem.

The remainder of the paper is structured as follows. Section 2 introduces a set of basic definitions. It is followed by the description of the PBMod constraints, its translation to PB constraints in section 3. Section 4 shows the encoding strategy based on a network of sorters to translate PBMod constraints into SAT formulas. Section 5, introduces a generator of random PBMod constraints and the solving strategies through SMT solvers and the usage of GLPK. We also report our experimental results for the different solving techniques. Finally, section 6 concludes the paper.

[1] Department of Computer Science, University of Lleida, C/ Jaume II 69, Lleida 25001, Spain

[2] The SMT solver Sateen (v.3.5) was the winner at the SMT Competition 2009 for the QF_LIA category, however we do not include yet the results since we have found some buggy answers.

2 Preliminaries

Boolean variables b_i can only take two values: false (0) and true (1). A literal l_i is either a variable b_i or its negation $\neg b_i$. A clause is a disjunction of literals, $l_1 \vee l_2 \cdots \vee l_n$. A SAT formula is a conjunction of clauses. An assignment is a mapping of Boolean variables to their value (true or false). A literal of the form b_i ($\neg b_i$) is satisfied by an assignment if b_i is mapped to value $true$ ($false$). A clause is satisfied by an assignment if at least one of its literals is satisfied. A SAT formula is satisfied by an assignment if all its clauses are satisfied. The SAT problem of a formula Γ consists in determining whether there exists an assignment to the Boolean variables of Γ that satisfies the formula.

A linear *pseudo-Boolean constraint* (PB constraint) over Boolean variables is defined by $\sum_i c_i \cdot l_i \rhd k$ where c_i, the coefficients, and k, are integer constants, l_i are literals and $\rhd$ is one of the operators of $\{=, <, \leq, >, \geq\}$. The left-hand side will be abbreviated by LHS, and the right-hand constant k referred to as RHS. Without loss of generality, these constraints can be rewritten to use the $\geq$ operator and positive coefficients (notice that $-c_i \cdot b_i$ can be rewritten as $c_i \cdot \neg b_i - c_i$). A coefficient c_i is said to be activated under a partial assignment if its corresponding literal l_i is assigned to true. Assuming that $\rhd$ is the $\geq$ operator, a pseudo-Boolean constraint is said to be satisfied under an assignment to its Boolean variables if the sum of its activated coefficients exceeds or is equal to k.

3 Modularity constraints as pseudo-Boolean constraints

A modularity constraint on Boolean variables, that we will call it *pseudo-Boolean modularity constraint*, is defined by:

$$\sum_i c_i \cdot l_i \equiv r \pmod m \tag{1}$$

where the coefficients c_i and the remainder r are integer constants, the modulo m is a positive integer constant, and the l_i symbols are literals on Boolean variables.
Without loss of generality, this constraint can be rewritten to use just positive integer constants through its normalization. This is achieved by replacing the coefficients and the remainder by their remainder modulo m.

Now, we can naturally translate Eq. 1 to a linear integer arithmetic constraint as follows:

$$\sum_i c_i \cdot l_i - k \cdot m = r \tag{2}$$

where k is a positive integer variable. We will refer to the translation model of Eq. 2 as ModLIA.

Notice that k is actually bounded, i.e., $k \leq \lfloor \frac{C}{m} \rfloor$, where $C = \sum_i c_i$. As we will see in section 5, in order to solve a problem involving linear integer arithmetic constraints we can use, among several approaches, a SMT solver with support for the Quantifier Free Linear Integer Arithmetic (QF_LIA) theory or a package like GLPK for Mixed Integer Programming. However, since our modularity constraint is defined on Boolean variables, we may rather be interested on translating Eq. 2 into PB constraints. In particular, we just need to express $k \cdot m$ as an arithmetic expression involving only Boolean variables as follows:

$$\sum_i c_i \cdot l_i - \sum_j k_j \cdot m = r \tag{3}$$

where k_j are Boolean variables and $j \in \{1, \ldots, \lfloor \frac{C}{m} \rfloor\}$.
We will refer to the translation model of Eq. 3 as ModPB-A.

We can still consider a slight variation that can potentially reduce the search space, by reducing the natural symmetry in Eq. 3. Notice that the sum of the k_j variables can be the same under different assignments. We can avoid this problem as follows:

$$\sum_i c_i \cdot l_i - \sum_j k_j \cdot j \cdot m = r \ \wedge \ \sum_j k_j \leq 1 \tag{4}$$

We will refer to the translation model of Eq. 4 as ModPB-B.

As we can see, any consistent assignment to the k_j variables will set at most one variable to true.

Another alterative, which uses fewer auxiliary variables, is to use a binary encoding:

$$\sum_i c_i \cdot l_i - \sum_{j'} k_{j'} \cdot 2^{j'} \cdot m = r \wedge \sum_{j'} k_{j'} \leq \left\lfloor \frac{C}{m} \right\rfloor \tag{5}$$

where $j' \in \{0, \ldots, \lfloor \log_2 \lfloor \frac{C}{m} \rfloor \rfloor\}$
We will refer to the translation model of Eq. 5 as ModPB-Bin. The term on the right is mandatory if $\lfloor \frac{C}{m} \rfloor$ is not a power of 2.

For Eq. 3, Eq. 4 and Eq. 5 we can now apply a PB solver.

4 Modularity Constraints as SAT formulas

For the sake of clarity during this section we will work on the following PBMod constraint which is already normalized:

$$3 \cdot b_1 + 2 \cdot b_2 + 5 \cdot b_3 + 3 \cdot b_4 = 1 \pmod 6 \tag{6}$$

As we can observe, $C = \sum_i c_i = 13$ and $\lfloor \frac{C}{m} \rfloor = \lfloor \frac{13}{6} \rfloor = 2$.

Then, taking into account the different translation models described in the previous section, we obtain the following PB constraints:

- ModPB-A:

$$3 \cdot b_1 + 2 \cdot b_2 + 5 \cdot b_3 + 3 \cdot b_4 - (k_1 \cdot 6 + k_2 \cdot 6) = 1$$

- ModPB-B:

$$3 \cdot b_1 + 2 \cdot b_2 + 5 \cdot b_3 + 3 \cdot b_4 - (k_1 \cdot 6 + k_2 \cdot 12) = 1 \wedge (k_1 + k_2 \leq 1)$$

- ModPB-Bin:

$$3 \cdot b_1 + 2 \cdot b_2 + 5 \cdot b_3 + 3 \cdot b_4 - (6 \cdot 2^0 \cdot k_o + 6 \cdot 2^1 \cdot k_1) \wedge (2^0 \cdot k_o + 2^1 \cdot k_1) \leq 2$$

As we can observe, with the ModPB-A model there are two possible ways to sum up to 6 with the k variables, while for the ModPB-B there is only one. In general, for n k-variables, the ModPB-A model considers 2^n consistent assignments while ModPB-B just considers n, thanks to the at most one constraint on the k-variables. ModPB-Bin also uses 2 auxiliary variables for this example, however, in general, the number is logarithmic in $\lfloor \frac{C}{m} \rfloor$.

These PB constraints can now be solved with a PB solver. From the different solving techniques of the state-of-the-art solvers, now, we focus on the one that consists in translating the PB constraints to a SAT formula [9]. There exist several translation techniques [9, 5, 16, 4]. Whether a translation is suitable or not depends both on the size of the encoding and the level of consistency that can be achieved under a partial assignment to the Boolean variables. In particular, in [9] three main approaches to how a SAT formula con be generated from a PB-constraint are studied. Parameter n is the total number of digits in all the coefficients.

- Translate the PB constraint into a BDD, which can be treated as a circuit of ITEs (if-then-else gates) and translated into clauses by the Tseitin transformation [19]. This approach guarantees that the resulting encoding is arc-consistent but its size is exponential in the worst case.
- Translate the PB constraint into a network of adders. The approach used in [9] is similar to the what is used for Data Multipliers to sum up the partial products [8]. The size of the translation is $O(n)$, however the resulting encoding is not Arc-consistent.
- Translate the PB constraint into a network of sorters. A sorter is a circuit of n input gates and n output gates where the k lowest output gates are set to true and the rest to false if there are exactly k input gates set to true. The size of the translation used in [9] is $O(n \cdot \log^2 n)$, and although it is not yet arc-consistent it is closer than the translation through Adders.

We have used the minisat+ [9] solver to convert the PB constraints, representing modularity constraints expressed with ModPB-A and ModPB-B, to SAT formulas using the three different translation techniques provided by minisat+: networks of adders, BDDs, and networks of sorters. For the size of problems that will be considered in our experimental evaluation, translation through networks of adders has shown poor performance (some orders of magnitude worse), BDDs do better but neither is as good as networks of sorters. This last translation technique together with ModPB-B is the best performing approach so far, consequently, in our experimental evaluation the other two translation techniques, adders and BDDs will not be shown.

Therefore, we have decided to explore whether we can achieve a better encoding if we translate directly the PBMod constraint to a SAT formula based on a network of sorters, instead of having an intermediate state where the PBMod constraint is translated to a PB constraint. In our translation we used the OddEvenMerge sorters [6]. For a more detailed explanation for OddEven mergesort see [12]. In our work the function compare(i_1,i_2) on two input gates (Boolean variables) is translated to a SAT formula, with two new auxiliary Boolean variables o_1, o_2 (the output gates), representing $(o_2 \leftrightarrow (i_1 \wedge i_2)) \wedge (o_1 \leftrightarrow (i_1 \vee i_2))$ with the following clauses:

$$\begin{array}{cc} (o_2 \vee \neg i_1 \vee \neg i_2) & (\neg o_1 \vee i_1 \vee i_2) \\ (\neg o_2 \vee i_1) & (o_1 \vee \neg i_1) \\ (\neg o_2 \vee i_2) & (o_1 \vee \neg i_2) \end{array}$$

A straightforward approach to convert a PBMod constraint into a SAT formula consists in flattening the LHS and using a sorter with as many inputs as the sum of the coefficients. Taking into account our example, once flattened,

$$\overbrace{b_1 + b_1 + b_1}^{3 \cdot b_1} + \overbrace{b_2 + b_2}^{2 \cdot b_2} + \overbrace{b_3 + b_3 + b_3 + b_3 + b_3}^{5 \cdot b_3} + \overbrace{b_4 + b_4 + b_4}^{3 \cdot b_4} \quad = \quad 1 \pmod 6$$

the sorter would have 13 inputs and we should add the circuit to check if the last output gate activated represents a number congruent to 1 modulo 6. Obviously, this approach is not good in terms of the size of the resulting encoding. So, a better approach consists in connecting several sorters, i.e., creating a network of sorters. We will refer to this encoding as ModCS. In order to apply the encoding based on sorters the numbers must be represented in unary instead of binary. The sorters play then the role of adders of unary numbers. Notice that the unary representation allows the use of any base for the coefficients. The first phase of the encoding process is to find a base such that the sum of all the digits of the coefficients written in that base, is as small as possible, since we want to minimize the entries to the sorters to get the lowest possible size of the encoding. Since, this is already a hard optimization problem, we use a brute-force search trying all primer numbers less than 20, as it is done in minisat+.

In our working example the base we use is $< 1, 3 >$. The coefficients in this base are represented as follows:

$$\begin{aligned} 2 &= (2,0)_{<\mathbf{1},\mathbf{3}>} = 2 \cdot \mathbf{1} + 0 \cdot \mathbf{3} \\ 3 &= (1,3)_{<\mathbf{1},\mathbf{3}>} = 1 \cdot \mathbf{1} + 3 \cdot \mathbf{3} \\ 5 &= (2,1)_{<\mathbf{1},\mathbf{3}>} = 2 \cdot \mathbf{1} + 1 \cdot \mathbf{3} \end{aligned}$$

As we can see in Fig. 1, in our example we use two sorters. One for the contribution of every coefficient to the 1-bits weight and another for the 3-bits weight. Notice that one of the input gates of the 3-bits sorter is connected to the third output gate of the 1-bits sorter. This is the way we represent the carry between sorters. Then, taking into account that the network of the sorters represents the sum of the LHS, we just need to add the additional circuitry, i.e. the comparator, that checks whether $LHS \in \{1, 7, 13\}$, the possible numbers congruent to 1 mod 6. For example, in order to assure that $LHS = 1$ we just need to check if the first output gate of the sorter 1-bits is true, the second is false and if the first output gate of the sorter 3-bits is also false. Notice that if the k_{th} output gate of a sorter is false then the rest of the upper gates must also be false. This circuitry can now be reused to test the conditions $LHS = 7$ and $LHS = 13$.

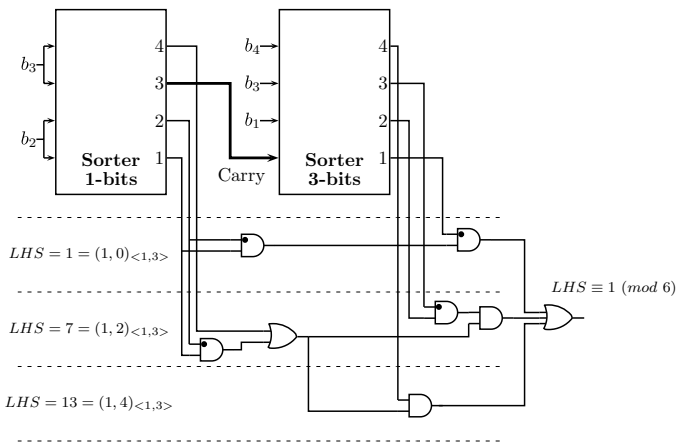

Figure 1. Schema for the ModCS encoding applied to example of Eq. 6

The motivation of this approach is to reduce the size of the encoding as much as possible both in terms of clauses and variables while preserving the good propagation properties. The PB solver minisat+ uses, in its translation of the PB constraints to a CNF formula, an intermediate circuit representation where any two syntactically identical nodes are merged by the so-called structural hashing. This reduces effectively the size of the encoding; however our approach still generates smaller formulas. Table 1 compares for two sets of the experimental results the size of our encoding approach, ModCS, to the conversion through minisat+ with the sorter option on model ModPB-B.

Table 1. Comparison of the encoding size for models ModPB-B (minisat+ with option sorters) and ModCS

(v,n,M,f)	Variables	Clauses	Vars. × Clauses
100,10,5,56			
ModPB-B	9,089	25,414	$2.3 \cdot 10^8$
ModCS	7,203	18,258	$1.31 \cdot 10^8$
		Ratio	1.75
100,50,5,22			
ModPB-B	60,988	179,466	$1.09 \cdot 10^{10}$
ModCS	33,249	89,880	$2.98 \cdot 10^9$
		Ratio	3.66

5 Experimental Analysis

In order to conduct our experimental investigation we have developed a generator of random pseudo-Boolean Modularity constraints, *randPBMod(v,n,M,f)*. The generator creates a formula consisting on f PBMod constraints, each one with n Boolean variables out of a total of v Boolean variables. The constraints are already normalized according to their respective modulo. For each PBMod constraint, the modulo m, the remainder r, the coefficients c_i and the variables b_i are selected from a uniform random distribution in the intervals $m \in [2, M]$, $r \in [0, M-1]$, $c_i \in [1, M-1]$ and $b_i \in [1, v]$. The variables are selected with no repetition for every PBMod constraint.

We have used several solving techniques for the experimental analysis:

- SAT solver: Precosat (v.236). Winner at the SAT Competition 2009 for the application category.
- PB solver: Bsolo (v.3.1). Winner at the Pseudo-Boolean Evaluation 2009 for the category "no optimization, small integers, linear constraints" in sat+unsat answers.
- SMT solver: Yices (v.2). Second winner at the SMT Competition 2009 for the Quantifier Free Linear Integer Arithmetic (QF_LIA) category.
- GLPK [3] advanced ILP solver with cutting planes. We have used the advanced branch and bound GLPK ILP solver, that preprocesses the input ILP problem for obtaining a simplified version of it, and we have also used the option of adding cover, clique and Gomory cutting planes, to further improve the pruning of the search space. For a general introduction to cutting planes, see for example [13].

In this section we show results for each of these solvers (labeled as `psat`, `bsolo`, `yices` and `gplk`) using each one at least one of the following encodings: ModPB-A, ModPB-B, ModPB-Bbin, ModCS, and ModLIA (§ Section 4). Take into account that the encodings ModPB-A, ModPB-B, and ModPB-Bbin, in order to be used by a SAT solver, should be first translated to SAT with minisat+ solver using the sorting networks option.

Our experiments have been run on machines with the following specs: Rocks Cluster 5.2 Linux 2.6.18 Operating System, AMD Opteron 248 Processor clocked at 1.6 GHz, 1.0GB Memory, and GCC 4.1.2 Compiler.

In order to evaluate the different encodings performance we have run our experiments for a wide range of parameters. The number of PBMod constraints in a formula (f) determines its satisfiability. As plotted on Fig. 2, the probability of having an unsatisfiable problem grows with f, as it becomes more constrained. In this case, when problems have $v \simeq n$, the satisfiability transition occurs at low values of f and there seems to be a sharp transition, like the one observed in other NP-complete problems, as SAT [17, 1] or CSP [20]. By relaxing this condition and allowing values of v larger than n, we can build satisfiable problems for larger values of f.

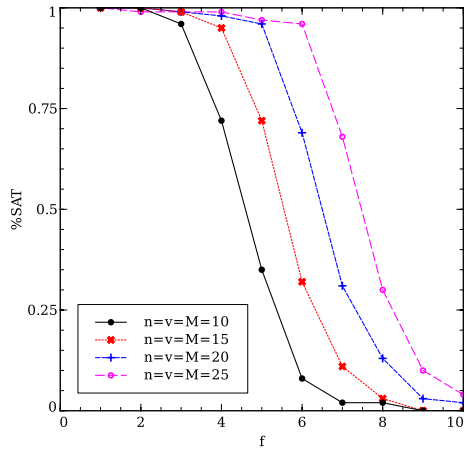

Figure 2. Percentage of satisfiable instances as as function of f

As a first benchmark, we have designed a set of experiments for $v = 100$; $n = 10, 20, 50$; and $M = 5, 10, 20$; testing values for f up to a given time out and solving 100 independent instances per point. This benchmark shows a first picture about the problem hardness parameter dependency, and proves that the problems become harder as n, M, and f grow. It also helps to look at the performance differences of our encodings in distinct benchmark conditions.

Figure 3 shows the ratio of performance, considering solving time using Precosat, between two of the best performing encodings for SAT solvers: ModCS and ModPB-B converted to SAT with minisat+ using the sorting networks option. Measures are done for the values of v, n (legend), and M (y-axis) detailed above. The boxes cover the region of the number of PBMod constraints (f, x-axis) where both encodings solve problems in the allotted time (the time out used has been 30 minutes), and still give significant times, i.e. more than 1 second. Inside each box figures represent the performance ratio of ModCS over ModPB-B, it is measured as the mean of the performance ratios over the considered range of f. As an example, a ratio of 1.77 means that ModCS is 1.77 times faster than ModPB-B.

A first conclusion from Fig. 3 is that ModCS performs better as the number of variables per constraint (n) increases. It should be noted that as n increases, so it does the sum of the coefficients (C), leading to more distinct interpretations of a congruence for a given M, and finally, giving more chances of reusing circuitry to ModCS encoder (as explained in the previous section), reducing the formula size. Second, as a result of the same effect, one can also observe, for $n = 50$, an increase of relative performance for ModCS encoder as M decreases, due to the larger number of constraints (f).

To prove the scaling with the number of variables, Table 2 reports the performance ratios for $v = 200$, $n = 50$, and $M = 5, 20$. The shown ratio is measured between ModCS and the best performing between ModPB-B and ModPB-Bbin. As we can see ModPB-Bbin is not competive with ModCS for large values of v, observe the difference between $v = 100$ and $v = 200$ in tables 2, 4, 5, and 6,

[3] http://www.gnu.org/software/glpk/glpk.html

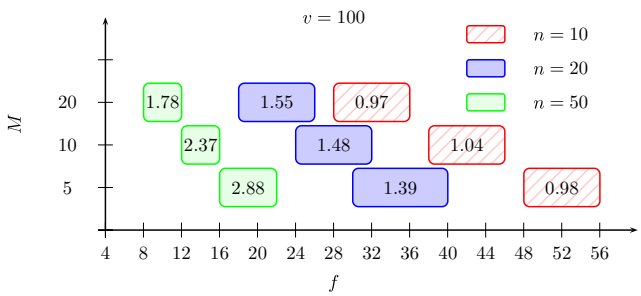

Figure 3. Performance ratio between two of the best encodings -ModCS and ModPB-B-, using Precosat v.236 for $v = 100$ and distinct values for n and f

especially for low values of M.

Table 2. Median time for ModCS, ModPB-B and ModPB-Bbin encodings using Precosat v.236. $v = 200$, $n = 50$ and their relative performance (– means median time larger than 30')

	f	ModCS	ModPB-B	ModPB-Bbin	Ratio
$M = 5$	36	912	1193	–	1.31
	34	158	297	765	1.88
	32	33	82	169	2.48
	30	8.2	32	63	3.90
	28	3.6	16	14	3.89
	26	1.9	9.3	6.1	3.21
	24	1.6	4.8	2.5	1.56
				Mean Ratio	**2.17**
$M = 20$	22	307	371	442	1.21
	20	19	45	32	1.68
	18	3	16	5.3	1.77
	16	2	7.4	2.3	1.15
	14	1.6	4.6	1.7	1.06
				Mean Ratio	**1.37**

Table 3. Median solving time / % of solved instances within alloted time for v,n and M=20 (– means median time larger than 30')

	f				
	6	**7**	**8**	**9**	**10**
bsolo/ModPB-A	–/26	–/8	–/3	–/0	–/1
bsolo/ModPB-B	647/53	–/18	–/9	–/3	–/3
psat/ModPB-B	4.1/100	12/100	13/100	12/100	12/100
psat/ModCS	4.3/100	14/100	16/100	16/100	17/100
glpk/ModPB-A	–/3	–/1	–/0	–/0	–/0
glpk/ModPB-B	55/97	130/96	245/95	354/97	373/95
glpk/ModLIA	12/97	25/97	28/95	25/97	24/97
yices/ModLIA	18/1	53/1	61/1	62/1	51/1

Tables 3, 4, 5, and 6 show the median time to solve all the instances and the percentage of solved instances for a time out of 30 minutes, and a wide range of parameter values. Left column denotes the employed combination solver/encoding as mentioned at the beginning of the current section.

Table 3 shows that ModPB-A is not a good option for any solver. It also shows that for solvers `yices` and `glpk`, none of both PB models (ModPB-A, ModPB-B) is competitive with ModLIA.

On the one side, it also seems clear that neither the SMT (yices/ModLIA) nor the LP (gplk/ModLIA) approaches are competitive with the other best combination encoding/solver. Only for high moduli ($M = 20$) and a large number of variables ($n = 50$), the SMT solver takes advantage of the linear integer arithmetic encoding ModLIA, approaching in time to the best performing approaches.

On the other side, the differences between the SAT and PB solvers are more subtle. With respect to both encodings used for Precosat solver, as mentioned above, our improvement of the translation through network of sorters (ModCS) slightly outperforms the original translation (ModPB-B) at most cases, increasing their differences as the number of variables (n) becomes larger, and particularly for low values of M, that is when the ModCS encoding ameliorates its circuit reusing capabilities. According to the PB solver performance (bsolo/ModPB-B), it does particularly well for under-constrained problems, i.e. when f is low, but tends to solve less instances than psat/ModCS for higher f. In order to reinforce this last point, we have conducted additional experiments for $v = 200$ and $n = 50$ as shown at Table 7.

Table 4. Median solving time / % of solved instances within alloted time for v=100 and n=10 (– means median time larger than 30')

	f				
M=5	**48**	**50**	**52**	**54**	**56**
bsolo/ModPB-B	1.2/99	11/94	107/71	–/36	–/17
psat/ModPB-B	1.1/100	2.5/100	9/99	42/95	286/50
psat/ModPB-Bbin	1.1/100	3/100	13/99	52/99	298/75
psat/ModCS	0.8/100	3.3/100	8.3/95	53/95	299/75
glpk/ModLIA	–/0	–/0	–/0	–/0	–/0
yices/ModLIA	–/3	–/0	–/0	–/0	–/0
M=10	**38**	**40**	**42**	**44**	**46**
bsolo/ModPB-B	6.8/81	–/47	–/21	–/9	–/2
psat/ModPB-B	2.9/100	13/97	130/88	720/71	1391/55
psat/ModPB-Bbin	2.9/100	14/96	131/90	643/72	1299/54
psat/ModCS	2.3/100	15/98	104/83	748/67	1598/51
glpk/ModLIA	–/0	–/0	–/0	–/0	–/0
yices/ModLIA	–/0	–/0	–/0	–/0	–/0
M=20	**28**	**30**	**32**	**34**	**36**
bsolo/ModPB-B	1.3/92	68/74	371/56	292/51	358/67
psat/ModPB-B	0.8/100	1.9/100	2.9/100	3.1/100	3.1/98
psat/ModPB-Bbin	0.6/100	1.7/100	2.6/100	3.0/100	2.4/100
psat/ModCS	0.7/100	1.8/100	3.0/100	3.5/100	3.0/100
glpk/ModLIA	–/0	–/0	–/0	–/0	–/0
yices/ModLIA	–/0	–/0	–/0	–/0	–/0

6 Conclusions

We have studied how to solve efficiently PBMod Modularity constraints. Although these constraints are naturally expressed as Linear Integer Arithmetic constraints, we have shown that is not the best approach at least when the LHS term of the PBMod involves only Boolean variables. We have proposed two possible alternative approaches: (i) a translation to PB constraints (models ModPB-A, ModPB-B and ModPB-Bin) and the usage of *pure* PB solver or a PB solver based on a translation to SAT, and (ii) a direct translation to a SAT formula based on a network of sorters (model ModCS) and the usage of a SAT solver. For the first approach we have introduced two encodings ModPB-B and ModPB-Bbin which are more competitive than the naive conversion to PB constraints, ModPB-A. However, the second approach, ModCS, is even more competitive since it allows to

Table 5. Median solving time / % of solved instances within alloted time for v=100 and n=20 (– means median time larger than 30')

	f				
M=5	**32**	**34**	**36**	**38**	**40**
bsolo/ModPB-B	2.0/100	16/91	128/69	–/31	–/6
psat/ModPB-B	4.8/100	13/100	62/99	340/87	1550/53
psat/ModPB-Bbin	5.9/100	14/100	73/100	393/86	–/45
psat/ModCS	3.1/100	11/100	38/100	312/89	1064/60
glpk/ModLIA	–/0	–/0	–/0	–/0	–/0
yices/ModLIA	–/14	–/0	–/0	–/0	–/0
M=10	**24**	**26**	**28**	**30**	**32**
bsolo/ModPB-B	0.5/100	8.3/95	–/48	–/12	–/5
psat/ModPB-B	2.7/100	11/99	99/92	1427/53	–/20
psat/ModPB-Bbin	1.9/100	11/99	80/96	885/61	–/20
psat/ModCS	1.1/100	9.5/100	85/87	1239/56	–/21
glpk/ModLIA	–/0	–/0	–/0	–/0	–/0
yices/ModLIA	–/12	–/2	–/0	–/0	–/0
M=20	**18**	**20**	**22**	**24**	**26**
bsolo/ModPB-B	0.1/100	1.4/95	1175/47	–/15	–/8
psat/ModPB-B	1.4/100	5.0/100	50/96	1423/55	–/10
psat/ModPB-bin	0.8/100	3.1/100	32/94	1033/55	–/13
psat/ModCS	0.7/100	2.9/100	35/95	1262/58	–/12
glpk/ModLIA	–/0	–/0	–/0	–/0	–/0
yices/ModLIA	–/3	–/0	–/0	–/0	–/0

Table 6. Median solving time / % of solved instances within alloted time for v=100 and n=50 (– means median time larger than 30')

	f			
M=5	**16**	**18**	**20**	**22**
bsolo/ModPB-B	0.7/100	5.8/99	92/86	–/44
psat/ModPB-B	10/100	27/100	100/100	542/80
psat/ModPB-Bbin	5.9/100	27/100	112/98	723/73
psat/ModCS	1.9/100	10/100	49/100	363/93
glpk/ModLIA	1493/53	–/15	–/3	–/0
yices/ModLIA	49/96	709/57	–/23	–/6
M=10	**12**	**14**	**16**	
bsolo/ModPB-B	0.4/100	8.1/89	–/46	
psat/ModPB-B	8.7/100	40/98	379/83	
psat/ModPB-Bbin	3/100	31/100	323/85	
psat/ModCS	2.1/100	24/100	272/83	
glpk/ModLIA	–/13	–/2	–/0	
yices/ModLIA	81/87	–/45	–/11	
M=20	**8**	**10**	**12**	
bsolo/ModPB-B	0.1/100	0.5/97	1559/50	
psat/ModPB-B	2.5/100	12/100	94/98	
psat/ModPB-Bbin	1.1/93	6.3/98	165/97	
psat/ModCS	1/100	3.5/100	96/97	
glpk/ModLIA	1341/57	–/3	–/0	
yices/ModLIA	5/95	714/60	–/13	

Table 7. Median solving time / % of solved instances within alloted time for v=200 and n=50

	f			
M=5	**30**	**32**	**34**	**36**
bsolo/ModPB-B	2.9/100	16/98	131/91	1562/52
psat/ModCS	8.2/100	33/100	158/91	912/69
M=20	**18**	**20**	**22**	
bsolo/ModPB-B	0.3/100	3.4/98	189/63	
psat/ModCS	3/100	19/100	307/82	

better exploit the expressiveness of the original model by producing smaller encodings while preserving the propagation properties.

ACKNOWLEDGEMENTS

This research was partially supported by projects TIN2007-68005-C04-02 and TIN2009-14704-C03-01 funded by the *Ministerio de Ciencia e Innovación*.

We would like to thank the referees for their insightful comments, which helped improve this paper considerably with their suggestions.

REFERENCES

[1] Dimitris Achlioptas and Yuval Peres, 'The threshold for random k-sat is $2^k \log 2 - \mathcal{O}(k)$', *Journal of the American Mathematical Society*, **17**(4), 947–973, (2004).

[2] Roberto Asín, Robert Nieuwenhuis, Albert Oliveras, and Enric Rodríguez-Carbonell, 'Cardinality networks and their applications', in *SAT'09*, pp. 167–180, (2009).

[3] Olivier Bailleux and Yacine Boufkhad, 'Efficient cnf encoding of boolean cardinality constraints', in *CP*, pp. 108–122, (2003).

[4] Olivier Bailleux, Yacine Boufkhad, and Olivier Roussel, 'A translation of pseudo boolean constraints to sat', *JSAT*, **2**(1-4), (2006).

[5] Olivier Bailleux, Yacine Boufkhad, and Olivier Roussel, 'New encodings of pseudo-boolean constraints into cnf', in *SAT*, pp. 181–194, (2009).

[6] K.E. Batcher, 'Sorting networks and their applications', in *Proc. AFIPS Spring Joint Comput. Conf.*, pp. 307–314, (1968).

[7] Mats Carlsson and Mats Grindal, 'Automatic frequency assignment for cellular telephones using constraint satisfaction techniques', in *Int. Conf. on Logic Programming, (ICLP'93)*, pp. 647–665, (1993).

[8] L. Dadda., 'Some schemes for parallel multipliers', *Alta Frequenza*, **2**, 14–17, (1968).

[9] Niklas Eén and Niklas Sörensson, 'Translating pseudo-boolean constraints into sat', *JSAT*, **2**(1-4), 1–26, (2006).

[10] Ian P. Gent and Toby Walsh, 'From approximate to optimal solutions: Constructing pruning and propagation rules', in *IJCAI'97*, pp. 1396–1401, (1997).

[11] Carla Gomes, Willem van Hoeve, Ashish Sabharwal, and Bart Selman, 'Counting csp solutions using generalized xor constraints', in *Proceedings of the 22nd National Conference on Artificial Intelligence*, (2007).

[12] H. W. Lang, 'Sequential and parallel sorting algorithms', in *http://www.inf.fh-flensburg.de/lang/algorithmen/sortieren/algoen.htm*.

[13] Hugues Marchand, Alexander Martin, Robert Weismantel, and Laurence A. Wolsey, 'Cutting planes in integer and mixed integer programming', *Discrete Applied Mathematics*, **123**(1-3), 397–446, (2002).

[14] Roberto Nieuwenhuis and Albert Oliveras, 'On sat modulo theories and optimization problems', in *SAT'06*, pp. 156–169, (2006).

[15] Silvio Ranise and Cesare Tinelli, 'The smt-lib standard: Version 1.2.', in *Technical report, Department of Computer Science, The University of Iowa. Available at www.SMT-LIB.org*, (2006).

[16] Olivier Roussel and Vasco M. Manquinho, 'Pseudo-boolean and cardinality constraints', in *Handbook of Satisfiability*, 695–733, (2009).

[17] B. Selman and S. Kirkpatrick, 'Critical behaviour in the computational cost of satisfiability testing', *Artificial Intelligence Journal*, **81**, 273–295, (1996).

[18] Carsten Sinz, 'Towards an optimal cnf encoding of boolean cardinality constraints', in *CP*, pp. 827–831, (2005).

[19] G. Tseitin, 'On the complexity of derivation in propositional calculus', *Studies in Cosntr. Math. and Math. Logic*, (1968).

[20] K. Xu and W. Li, 'Exact phase transition in random constraint satisfaction problems', *Journal of Artificial Intelligence Research, (JAIR)*, **12**, 93–103, (2000).