On extending SAT solvers for PB problems

Daniel Le Berre and Anne Parrain

CRIL-CNRS FRE 2499, Université d'Artois, Lens, France
{leberre,parrain}@cril.univ-artois.fr

Abstract. The use of SAT solvers to handle practical problems has grown dramatically over the last decade. SAT solvers are now mature enough to be used in hardware or software model checkers and to have an impact on our everyday's life of computer users. Many researchers are working on extensions of the current framework to more general constraints: pseudo-boolean (PB) constraints, satisfiability modulo theories, etc. We present first pseudo boolean constraints and their relationship with plain clauses. We review current available solutions for solving PB-constraints in Conflict Driven Clause Learning solvers, the most successful architecture of SAT solvers for SAT instances resulting from a translation of the initial problem into SAT. Then we focus on the current limits of our own implementation SAT4JPseudo that participated to the three PB evaluations. We conclude by pointing out some features that require attention for building in the future efficient PB solvers within the CDCL architecture.

1 Introduction

The satisfiability problem receives currently a lot of attention from various areas, because of the availability of numerous efficient SAT solvers for CNF derived from "real problems" [22, 23]. We will refer to that class of solvers as Conflict Driven Clause Learning (CDCL) solvers in the rest of the paper. Among the reason of the steady improvements of SAT solvers, the organization of competitive events such as the SAT Competitions[1] or the SAT Race[2], and the availability of numerous benchmarks (SATLIB[3], Velev[4], IBM[5], etc) play certainly an important role. However, the most important factor in our opinion is that many SAT solvers have been developed and made freely available to the community in such a way that algorithmic improvements for unit or boolean constraint propagation, conflict analysis, memory management, etc. are shared and contribute to an overall improvement of SAT solvers.

SAT technology might even reach everybody's desktop computer in a short term. Indeed, SAT solving is currently being used in the EDOS project[6] to improve package depencies in Linux systems [19]; and some deficiencies in the latest Linux kernel have been identified by the SAT-based bug finder SATURN[7] [29, 30]. Furthermore, several tools in software engineering are currently using SAT solvers in formal specification (Alloy[17], Kodkod[8]), feature modeling [3], etc. Finally, the huge success of the open source SAT solvers MiniSat and SatELite[9] in the SAT 2005 competition pushed further the use of SAT solvers in unexpected research area, like the termination problem for rewrite systems and logic programs. The most efficient termination solver of the 2005 termination competition, Aprove [12], but also Jambox [11] and Matchbox [28], participated to the 2006 termination competition[10] with SatElite or MiniSat as helper [15, 7]. Those solvers have

[1] `http://www.satcompetition.org/`
[2] `http://fmv.jku.at/sat-race-2006/`
[3] `http://www.satlib.org/`
[4] `http://www.ece.cmu.edu/~mvelev`
[5] `http://www.haifa.ibm.com/projects/verification/RB_Homepage/bmcbenchmarks.html`
[6] `http://www.edos-project.org`
[7] `http://glide.stanford.edu/saturn/`
[8] `http://web.mit.edu/emina/www/kodkod.html`
[9] `http://www.cs.chalmers.se/Cs/Research/FormalMethods/MiniSat/`
[10] `http://www.lri.fr/~marche/termination-competition/`

been awarded the 2006 competition: Aprove best for term rewriting systems (except the relative termination subcategory) and for logic programs, Jambox best for string rewriting systems and relative termination of term rewriting [21]. So CDCL solvers can be considered nowadays a mature technology.

Another research area has received a lot of attention in recent years: Satisfiability Modulo Theories[11]. It can be considered as a generalisation of SAT in which the constraints belong to a specific theory. Two years ago, a competition of SMT solvers was organized[12], and the SMT solver BarcelogicTools [24] was awarded in all the four categories in which it competed (last year, it was defeated by Yides [9], but it was still a very good solver). The BarcelogicTools solver participated to the SAT Race 2006 and showed very good performances (6 out of 16) for "parameterized" SAT solver. Such good results could suggest that all the techniques used in CDCL SAT solvers could be generalized and efficiently used in a more generic kind of solver (so called DPLL(T)) where T if the theory specific inference engine.

The same kind of ideas lead to the design of Pseudo-Boolean CDCL solvers [1], in which the constraints are linear inequalities with integer coefficients and boolean variables. Extending a SAT solver to handle such constraints is not very difficult since the constraints can be weakened to simple clauses during conflict analysis. However, taking full advantage of those constraints using cutting planes is more challenging [8, 6] since it questions many algorithms and data structures used in CDCL solvers.

In 2005, the first Pseudo-Boolean (PB) evaluation was organized [27]. The most interesting result of that evaluation in our opinion is that a translation of the original problem into a plain CNF was competitive with all the other techniques. Basically the same conclusion can be drawn from the results of the second and third PB evaluations in 2006 and 2007. In some ways, it means that the currently available Pseudo-Boolean (PB) solvers do not take much advantage at working directly with PB constraints on the currently available benchmarks.

In the rest of the paper, we first introduce the so-called pseudo-boolean constraints and the inference rule called *cutting planes*. We briefly describe CDCL algorithm and present methods to extend it to PB-constraints. Then we discuss the results of our extended SAT solver SAT4JPseudo during the three pseudo-boolean evaluations and summarize the lessons learnt when using cutting planes instead of resolution in a pseudo-boolean conflict driven constraint learning solver.

2 (Linear) pseudo-boolean constraints

2.1 Definitions

A linear pseudo-boolean constraint is defined over a set of boolean variables x_i. The value true will be coded as 1 while the value false will be coded by 0. A general linear pseudo-boolean constraint follows the following pattern $\sum_i a_i.x_i \triangleright k$ where a_i and k are integer-valued constants and $\triangleright$ is one of the classical relational operators $(=, >, \geq, <, \leq)$. The multiplication and the sum operators have their usual mathematical meaning. The right hand side of the constraint (k) is called the *degree* of the constraint.

Such general linear pseudo-boolean constraints can be transformed to use only the $\geq$ operator (equality constraints can be split in two constraints ($\geq$ and $\leq$); $<$ and $\leq$ constraints can be multiplied by -1 to obtain $>$ and $\geq$ constraints respectively and at last $\sum_i a_i.x_i > k$ can be transformed into $\sum_i a_i.x_i \geq k+1$. For now, a PB-constraint is in the form

$$\sum_i a_i.x_i \geq k, \qquad a_i, k \in \mathbb{Z} \tag{1}$$

By introducing literals using $\overline{l_i} = 1 - x_i$, we may further transform so that each coefficient is only a positive integer. Let l_i denotes either x_i and $\overline{x_i}$. Every conjunction of linear pseudo-boolean

[11] http://www.smt-lib.org/

[12] http://www.csl.sri.com/users/demoura/smt-comp/2005/

constraints can be transformed to become a conjunction of greater or equal constraints over literals with strictly positive coefficients :

$$\sum_i a_i.l_i \geq k, \qquad a_i, k \in \mathbb{N} \tag{2}$$

For example, $3x_1 - 7x_2 < -2$ is transformed into $3\overline{x_1} + 7x_2 \geq 6$.

Clauses and (normalized) cardinality constraints can be seen as a special case of linear pseudo-boolean constraints. Clause $l_1 \vee l_2 \vee \dots l_n$ translates to $l_1 + l_2 + \dots + l_n \geq 1$, cardinality constraint $atleast(k, \{l_1, l_2, \dots, l_n\})$ translates to $l_1 + l_2 + \dots + l_n \geq k$ and $atmost(k, \{l_1, l_2, \dots, l_n\})$ translates to $\overline{l_1} + \overline{l_2} + \dots + \overline{l_n} \geq n - k$.

2.2 Inference rules

In propositional calculus, there are two essential inference rules which can be applied to a formula in conjunctive normal form: *resolution* and *merging*. The latter one is often underestimated because usually clauses are assimilated to sets of literals and the merge rule is automatically obtained by the set union. However, this merge rule is necessary for the completeness of a refutation procedure. The merge rule also plays a central role in the predicate calculus where literals may be merged by computing the most general unifier. Here again, both resolution and merging are necessary to obtain a complete refutation procedure, but the merge rule is no more straightforward as in propositional calculus.

With PB constraint, we need essentially the same two rules to get a complete procedure [4] but we may also use additional inference rules which have no equivalent in the propositional calculus. The rule corresponding to resolution is called *cutting plane* and computes a positive linear combination of two PB constraints.

For PB-constraints in form (1) :

$$\textit{cutting plane:} \quad \frac{\begin{array}{c}\sum_i a_i.x_i \geq k \\ \sum_i a'_i.x_i \geq k'\end{array}}{\begin{array}{c}\sum_i (\alpha.a_i + \alpha'.a'_i).x_i \geq \alpha.k + \alpha'.k' \\ \text{with } \alpha > 0 \text{ and } \alpha' > 0\end{array}}$$

In general, the coefficients of the linear combination are chosen so as to eliminate at least one variable, i.e. such that $\exists i, \alpha.a_i + \alpha'.a'_i = 0$.

The second essential rule corresponding to the merge rule is called *saturation*. When one coefficient a_j is greater than the degree k, it may be replaced by k (which amounts to merging some occurrences of l_j). This rule can only be applied on PB-constraints in form (2) :

$$\textit{saturation:} \quad \frac{a_j.l_j + \sum_{i \neq j} a_i.l_i \geq k \text{ with } a_j > k}{k.l_j + \sum_{i \neq j} a_i.l_i \geq k}$$

One simple way to justify this rule is to consider that, when l_j is true, the constraint will be satisfied even if a_j is reduced to k, and therefore, reducing the coefficient to k doesn't change anything.

By applying saturation rule over constraints in form (2), we obtain now a normal form for PB-constraints :

$$\sum_i a_i.l_i \geq k, \qquad a_i, k \in \mathbb{N}, \forall i \quad a_i \leq k \tag{3}$$

The *rounding* rule allows us to divide each coefficient as well as the degree by a strictly positive number and round up each number obtained.

$$\textit{rounding up:} \quad \frac{\sum_i a_i.l_i \geq k}{\sum_i \lceil a_i/\alpha \rceil.x_i \geq \lceil k/\alpha \rceil \text{ with } \alpha > 0}$$

One can easily notice that any pseudo-boolean constraint can be weakened to a simple clause by applying the rounding up rule for $\alpha = k$ together with the saturation rule.

One last inference rule that will prove useful is *reduction*. It consists in forgetting one of the variables of the constraint and adjusting the degree in concordance.

$$\textit{reduction:}\ \frac{\sum_i a_i.l_i \geq k}{\sum_{i \neq j} a_i.l_i \geq k - a_j}$$

2.3 Brief historical survey

Cutting planes have been introduced in the framework of integer linear optimization by R.Gomory in 1958 [14]. The links between cutting planes and resolution (cutting planes can be seen as a generalization of resolution) have been investigated by J.N. Hooker [16]. Later, B. Benhamou, L. Saïs and P. Siegel [4] proposed a complete proof system for a particular kind of cardinality constraints. In their article, they consider a cardinality constraint as a pair (list of literals, degree). It means that a literal can appear several times in a constraint, so these constraints are equivalent to pseudo-boolean constraints.

P. Barth [2] proposed the first extension of the Davis-Putnam-Logeman-Loveland algorithm (so-called DPLL in the following) to the case of the 0-1 constraint programming. Since then, a number of new solvers has emerged. We will particularly focus on solvers trying to conjugate the efficiency of SAT solvers such as Chaff [23] with the power of inference rules dedicated to PB constraints. Similar works have been done previously in PBChaff [8] and Galena [6].

For more details regarding cutting planes and propositional calculus, see e.g. [5].

2.4 Conflict Driven Constraint Learning Solvers

Complete SAT solvers are searching the boolean space using a binary tree on the variables truth value. Traditional DPLL solvers are exploring explicitly that search tree. By contrast, CDCL solvers are using a specific learning scheme to explore the boolean search space. The idea is the following:

1. Any truth value propagation is either the consequence of one of the constraints, in that case the constraint is called the reason of the propagation and the truth value is said *forced*, or the consequence of the heuristics, in that case the truth value is called a *decision value*. This is true for any search based SAT solver. The basic case in which a clause does propagate a truth value is when that clause contains only one literal (a so called *unit clause*).
2. propagation of the truth values may end with no more variables to assign: in that case the formula is proved SATisfiable
3. most often, the propagation will end with a conflict: a constraint will be falsified. In that case, a classical backtracking algorithm would undo the latest decision value and try the opposite one. In CDCL solvers, the conflict analysis engine will analyse the reason of the conflict and produce both a *clause* and a *backtrack level* such that the clause becomes assertive, i.e. forces a truth value, at the proposed backtrack level after simplification. Note that a CDCL solver may backtrack higher than the latest decision level (back-jumping) and may not flip a decision value when backtracking: the flip implied by the new clause may occur for any truth value that was fixed below the backtracking level. As a consequence, each time a conflict is found, a decision value becomes forced. The inconsistency is proven when a conflict results from forced only truth values.

Extending a SAT solver to pseudo-boolean constraints consists in replacing the usual resolution inference rule by the above rules. In a conflict driven clause learning solver, resolution is used to derive the clauses needed for non-chronological backtracking and learning in the conflict analysis procedure. In a more general conflict driven constraint learning solver, the conflict analysis procedure will use all the above rules to ensure that the derived constraint has some desirable properties (see e.g. [6, 8] for details).

D. Chai and A. Kuehlmann present in [6] a generalization of the structure of the CDCL solvers [22] which takes as input clauses, cardinality or pseudo-boolean constraints and learn any kind of clauses, cardinality or pseudo-boolean constraints. The purpose of the following subsections is to describe the differences between dealing with pseudo-boolean constraints or propositional clauses concerning boolean constraint propagation and conflict analysis algorithms. We use algorithms proposed in the Chai *et al*'s CDCL solver for the pseudo-boolean case.

2.5 Implicative and Assertive constraints

In the following, pseudo-boolean constraints are in normal form (3).

The definition of unit clauses can be extended to the pseudo-boolean constraints case. However, the term *unit* refers to the syntactic form of such clauses. This term is not meaningful for pseudo-boolean constraints, so we will prefer to name such constraints *implicative constraints.* A unit clause implies exactly one literal, which is no longer true for pseudo-boolean constraints. An *implicative constraint* is a constraint which implies *at least* one literal. More formally, a constraint C is *implicative* iff $\exists a_i \in C$ such that $a_i > \sum a_j - k$. For example, for x_1, x_2, x_3 unassigned literals, the pseudo-boolean constraint $3x_1 + 2x_2 + x_3 \geq 5$ implies x_1 and x_2 are satisfied (assigned to 1).

In classical CDCL solvers, the aim of the conflict analysis algorithm is to learn a clause which will be unit when backtracking. These clauses are called *assertive clauses.* This definition can be extended easily to the case of pseudo-boolean constraints: an *assertive constraint* is a constraint that will become implicative when backtraking.

2.6 Detecting implicative constraints

Dixon *et al* [13] name *poss* the difference between the sum of the coefficients of the literals which are not falsified and the degree of the constraint. The same value is named *slack* by Chai *et al.* It allows to detect easily some states of the constraint:

- a constraint is falsified iff its *poss* value is negative;
- a constraint is implicative iff there exists a literal l_i with coefficient c_i such that $poss - c_i$ is negative.

As a result, maintaining the value of *poss* during the search is sufficient to detect when a constraint become falsified or implicative.

The watched literals scheme proposed by [23] can also be used to detect those states, as proposed in [6]. While it is sufficient to watch two unfalsified literals for a clause, the set of watched literals must contain unassigned literals such that the sum of their coefficients plus the coefficients of the satisfied literals is greater than the sum of the degree of the constraint and the largest coefficient of the unassigned literals. When a literal of this set is falsified, it is replaced by the minimal number of literals which allows to verify the condition. As a result, the size of the set may vary during the resolution, which is not the case for a clause. ¿From that property, one can also deduce how many literals must be watched for cardinality constraints: exactly its degree + 1, because the largest coefficient of unassigned literals is 1. It means that the larger the degree, the larger the set of literals to watch. Clauses are the best cases for that approach. As a consequence, the extended watched literal approach is not used for handling PB constraints in our solver SAT4JPseudo.

2.7 Detecting assertive constraints

CDCL solvers detect assertive clauses using a syntactical test based on the definition of a Unique Implication Point: assertive clauses contain only one variable of the current decision level. This can be done either by making some cut in the implication graph[13], or by applying iteratively resolution on the conflict clause and the reasons of the previous assignments, until an assertive clause is found.

For this, the search benefits from a nice property of unit clauses: a clause is unit at most once in each branch in the resolution tree. This property does not stand anymore in the pseudo-boolean case. For example, for x_1, x_2, x_3 unassigned literals, the pseudo-boolean constraint $3x_1+2x_2+x_3 \geq 4$ implies x_1 is satisfied. As x_2 and x_3 are unassigned, it is also easy to see that, as soon as x_2 is falsified, the constraint will imply x_3 to be satisfied, and vice-versa. Then, the same constraint can be the conflict constraint and a reason of a previous assignment.

Furthermore, the cutting plane of the falsified constraint and the reason of the conflictual assignment does not always result in a falsified constraint, as opposed to the clausal case. This is because an assignment forced by an implicative constraint can over satisfy that constraint, i.e. *poss* is strictly greater than the degree. The *poss* value represents the constraint's margin with respect to the unsatisfiability. To obtain a conflictual constraint by applying resolution, constraints must have *poss* values such that their sum is strictly negative. Reduction of constraints by deleting unassigned or satisfied literals can be needed to reduce the *poss* value of a constraint. This can be seen as concentrating the conflict analysis on (falsified) literals which have been responsible for the conflict.

2.8 Example

Let us consider the following example to illustrate the discussion.

$$\begin{cases} (a)\ 5x_1 + 3x_2 + 2x_3 + 2x_4 + x_5 = 8 \\ (b)\ \qquad\qquad\qquad x_1 + x_3 + x_4 \geq 2 \end{cases}$$

Using the transformation discussed previously, this formula is translated in

$$\begin{cases} (a_1)\ 5x_1 + 3x_2 + 2x_3 + 2x_4 + x_5 \geq 8 \\ (a_2)\ 5\overline{x_1} + 3\overline{x_2} + 2\overline{x_3} + 2\overline{x_4} + \overline{x_5} \geq 5 \\ (b)\ \qquad\qquad\qquad x_1 + x_3 + x_4 \geq 2 \end{cases}$$

At the top decision level (`DL = 0`), x_5 is assigned 0. (a_1) becomes implicative and x_1 is assigned to 1.

At the following decision level (`DL = 1`), x_4 is assigned 0. (b) becomes implicative and x_3 is assigned to 1. (a_1) becomes also implicative and x_2 is assigned to 1, which leads to a conflict with (a_2).

Resolution should be made on (a_1) and (a_2) wrt x_2. But at decision level 1, the *poss* value of (a_1) is $+2$, of (a_2) is -2. The sum is equal to 0, so (a_1) has to be reduced. Reduction can be obtained by deleting x_1, then x_3. Note that after a reduction or a cutting plane, a saturation may be needed. The constraint obtained is now

$$(a_3) \quad x_2 + x_4 + x_5 \geq 1$$

which has a *poss* value of 0. Resolution on (a_3) and (a_2) returns (literal value and decision level are noted `value@level`)

$$(c) \quad 2\overline{x_1}(0@0) + 2\overline{x_3}(0@1) + x_4(0@1) + 2x_5(0@0) \geq 2$$

which is conflictual at decision level 1, and assertive at decision level 0 ($\overline{x_3}$ is implied to 1). This constraint contains two variables on the current decision level which could not be the case for clause learning. Furthermore, this constraint is stronger than

$$x_4 + x_5 \geq 1$$

which is a clause that could have been learned (by negation of the decisions $\neg(\overline{x_4} \wedge \overline{x_5}) = x_4 \vee x_5$).

[13] An implication graph is a graph representation of the variable dependencies.

Type		MiniSat+	SAT4JPseudo	Pueblo	PBS	Bsolo
Decision	UNSAT/SAT	43/(35)	52/17	**61/42**	**61**/28	36/8
	SMALL	10/**176**/(296)	10/120/(346)	10/160/342	10/133/133	10/159/339
Optimization	MEDIUM	0/24/(97)	**2**/19/126	0/**34**/108	0/33/33	0/28/110
Unsat/Opt/Sat	BIG	**103/26**/(90)	85/3/(174)	-	-	90/9/92

Fig. 1. Partial results of the PB05 evaluation. See `http://www.cril.univ-artois.fr/PB05/results/` for detailed results.

3 Solving Pseudo-Boolean problems during the PB evaluations

We are reporting here partial results of the Pseudo-Boolean evaluations organized in 2005[27] and 2006. We restricted ourselves to the solvers that participated to the two evaluations, and that we consider as extended SAT solvers.The results in each cell denote either the number of UNSAT/SAT benchmarks solved in the first row (decision problems), or the number of UNSAT/OPT/SAT in the other ones (optimization problems). An OPT answer means that the solver has found a solution, and proved it to be optimal. A SAT answer means that the solver found at least a solution, optimal or not. This corresponds to the um of OPT and SAT answers during th evaluations. The numbers between parenthesis denote cases for which some certificates could not be verified (minor output bug in MiniSat+ and reactivity problem with SAT4J) during the evaluations: they are computed as the sum of OPT, SAT, SAT-TIMEOUT and NO-CERT values appearing on the evaluations web sites. The categories are related to the size of the integer in the constraints: MEDIUM and BIG categories need arbitrary precision integer arithmetic to avoid overflow. For more details, see e.g. [27] or the evaluations web sites[14].

3.1 The solvers

Bsolo [20] is a branch'n'bound PB solver inheriting most features from CDCL solvers, plus additional lower bounding techniques and heuristics based on Integer Linear Programming. It is the only solver really dedicated to optimization problems. As a consequence, it usually performs better on optimization problems than on decision problems. Note also that Bsolo adapt its features to the category: it disables some sophisticated features for MEDIUM and BIG categories. So Bsolo shows full power only in the SMALL category.

MiniSat+ [26] uses a sophisticated translation of the pseudo boolean constraints into CNF to feed the powerful MiniSat solver [10]. It is possible to make the translation by converting a PB-constraint into a BDD, and then converting the BDD into clauses. In MiniSat+, the coding is done using the best translation via either BDD, adder networks or sorter networks.

Pueblo [25] is an extended CDCL solver using both resolution and cutting plane based inference for conflict analysis. The pseudo-boolean constraints derived during the search are removed periodically.

PBS [1] was one of the first extension of CDCL solvers to handle PB constraints. Conflict analysis is performed using resolution. The version submitted to PB06 has also additional cutting-plane based processing (unpublished). Note that PBS is not able to answer SAT when an optimization problem needs to be solved (interaction problem with the evaluation framework).

SAT4JPseudo Our own extended CDCL solver in which cutting planes replace resolution during conflict analysis, in the spirit of PBChaff [8] and Galena [6]. The source code of SAT4JPseudo is available from `http://www.sat4j.org`.

3.2 Partial results from the PB evaluations

Some results from the first PB evaluation are depicted in Fig. 1. One can note that the MiniSat+ solver was quite competitive with the extended SAT solvers. Pueblo and PBS are quite good

[14] `http://www.cril.univ-artois.fr/PB06/`

	PB06					Additional
	MiniSat+	SAT4J Heuristics	Pueblo 1.4	PBS 4.1L	Bsolo	SAT4JRes
UNSAT/SAT	172/148	79/92	**204/153**	199/144	111/118	165/121
SMALL	43/405/655	**54**/357/660	37/385/708	29/352/352	40/**409**/689	35/367/(634)
MEDIUM	0/3/12	0/4/13	0/4/19	0/5/5	0/**6**/13	0/5/(14)
BIG	38/33/85	37/57/134	-	-	30/14/83	**40/72/168**

Fig. 2. Partial results of the PB06 evaluation plus additional results in the same conditions. See `http://www.cril.univ-artois.fr/PB06/results/` for detailed results.

	MiniSat+	SAT4J C.P.	Pueblo 1.4	PBS4-v2	Bsolo	SAT4J Res.
UNSAT/SAT	170/**146**	88/56	**202/139**	197/130	137/141	167/110
MEDIUM	35/208/510	**47**/147/544	31/180/**569**	23/141/557	33/**237**/562	35/156/532
BIG	38/36/103	38/60/146	-	-	29/117	**40/78/190**

Fig. 3. Partial results of the PB07 evaluation. See `http://www.cril.univ-artois.fr/PB07/results/` for detailed results.

at solving decision problems. Our own solver provided many non optimal answers. However, the overall results were well behind the other solvers for optimal answers. During the second PB evaluation (Fig. 2), on a bigger set of benchmarks, MiniSat+ provided again the best results in the optimal SMALL category. Pueblo was again the best PB solver on decision problems. Our own solver performed admittedly poorly in general, but was able to solve a fair amount of UNSAT benchmarks in the OPTSMALLINT category. We added to the official results of the PB evaluation an additional column with the score of SAT4J with resolution instead of cutting planes under the same conditions. The results are usually better, especially in the BIG category where it outperforms the other solvers. As a consequence, the extended watched literal approach is not used for handling PB constraints in our solver SAT4JPseudo. As a consequence, the extended watched literal approach is not used for handling PB constraints in our solver SAT4JPseudo. Those results were confirmed during the third PB evaluation (Fig. 3) in which three flavors of SAT4J pseudo were submitted, in particular a version with full cutting planes and a version with resolution. We are detailing in the next section the results of those solvers on specific kind of benchmarks.

3.3 Speed vs power in PB06

In Fig. 4, we depict for some families of benchmarks the total number of benchmarks solved by each solver (i.e. UNSAT+OPT or UNSAT+SAT). We focussed on families with a particular form (e.g. pigeon hole or Ardal) or for which the size was large enough to draw some conclusions.

The pigeon hole problems are used to discriminate resolution-based against more powerful reasoning engines. In that case, SAT4J and PBS outperform the other solvers. Pueblo, despite using cutting planes during conflict analysis, cannot solve all of them. Note that a pure linear programming approach such a glpPB[15] can also solve all of them. The mps-reduced problems contain mainly pseudo-boolean constraints. On those benchmarks, our solver performed better than the other solvers, and more generally, solvers using cutting planes in some ways did perform better than the other solvers. The good result of SAT4J on those benchmarks contributes 34 of the 54 UNSAT benchmarks solved in the OPTSMALLINT category. On the other hand, the minprime problems contain only clauses and an optimization function. For such benchmarks, the translation into full CNF is the best option, but Pueblo is not far away.

At the light of those first results, we could expect that cutting plane based solvers should perform better on benchmarks containing mostly pseudo-boolean constraints.

Some puzzling results can be found in the Ardal problems contributed by Vasco Manquinho. They express instances of the *subset sum* problem with a single equality constraint with few variables (between 60 to 100). That constraint is expressed by two inequalities in the PB06 format.

[15] `http://www.eecs.umich.edu/~hsheini/pueblo/`

	MiniSat+	SAT4J	Pueblo	PBS	Bsolo	glpPB
SAT/UNSAT						
pigeon hole /20	2	**20**	13	**20**	2	20
queens /100	**100**	18	99	**100**	**100**	**100**
tsp / 100	91	20	**100**	85	40	42
fpga /57	35	43	**57**	47	9	26
uclid /50	**47**	30	42	44	38	10
OPT SMALLINT						
minprime /156	**124**	104	118	103	106	52
reduced-mps /273	46	**70**	63	27	54	58
Ardal problems (one eq. constraint)						
Ardal_1 /12	**10**	2	0	3	2	0

Fig. 4. Partial results of the PB06 evaluation. See `http://www.cril.univ-artois.fr/PB06/results/` for detailed results.

MiniSat+ outperforms the other solvers on those benchmarks. PBS was able to solve 3 of them, SAT4J with cutting planes 2, SAT4J with resolution 3 (the benchmarks solved by PBS and SAT4J are not the same). We can illustrate the difference of efficiency between cutting plane reasoning and resolution based one in SAT4J on one benchmark of that family (prob5): the speed of the resolution based solver is 1832 decisions per second against 22 decisions per second for the cutting plane version, for a timeout of 1800s! Note that a pure linear programming approach (solver glpPB) was unable to conquer any of those benchmarks either.

On FPGA problems, SAT4J performs poorly on SAT benchmarks, but well on UNSAT ones (solvers having cutting planes capabilities – glpPB, PBS4.1L, Pueblo, SAT4J – were the only ones able to solve all unsat benchmarks in that category). Uclid benchmarks are all unsat but SAT4J was not very robust on them, while MiniSat+ provides the best results.

There are a few cases on which our solver performed really badly compared to the other solvers, namely on TSP and Weighted Queens problems[16] contributed by Gayathri Namasivayam. Those benchmarks have much more clauses than cardinality constraints or PB constraints.

Just to illustrate the root of the problem, we can take an example of a queens problem that can be solved by the resolution based SAT4J in 35 seconds after 16 restarts, 95829 conflicts at 3320 decisions per seconds. The full cutting plane version times out at 1800 seconds after only 7 restarts for 2338 conflicts at 7 decisions/second! So full cutting plane based reasoning is powerful (see e.g. pigeon hole problems, reduced-mps problems, etc.) but we have currently no "efficient" way to implement it. The next section summarizes the theoretical/algorithmic problems met when using cutting plane based inference.

3.4 The limits of our cutting plane based CDCL solver SAT4JPseudo

In [18], we focussed on the problem of representing pseudo boolean constraints in a CDCL solver, using the watched literal data structure as an example of property that cannot be easily adapted to PB constraint [6, 8]. That property is useful for efficient unit propagation. In the version of SAT4JPseudo we submitted to the PB05 evaluation, all the constraints were translated into PB constraints in order to apply cutting planes. As a result, the solver was much slower than the other ones on benchmarks with many clauses. In the version submitted to the PB06 evaluation, each kind of constraint was represented in the simplest way possible: clause, cardinality constraint or PB constraint. As a consequence, the cost of unit propagation is now kept as low as possible in SAT4JPseudo, and the speed of the new solver is usually greater than the previous one.

However, after the analysis of the PB06 evaluation, we detected that the speed of our cutting plane based solver, in number of decisions taken per second, was several order of magnitude smaller than our resolution based solver.

[16] `http://www.csr.uky.edu/~gayathri/pbsmodels.htm#Benchmarks`

We summarize here the main reasons that could explain that difference:

literals vs weighted literals weight manipulation makes cutting planes operations much more costly than resolution. This is even worse when weights are arbitrary precision integers! Note that we are the only one CDCL solver to apply cutting planes with arbitrary precision integer arithmetic. The other solvers reduce the constraints to cardinality constraints when facing overflow.

maintaining a conflictual constraint When using cutting planes in conflict analysis, additional processing is needed to ensure that the resulting constraint is conflictual [6]. However, in our current solver, we limit that extra processing thanks to the properties defined in [8].

detecting efficiently assertive constraints Conflict analysis ends when the derived constraint is assertive. It is easy to detect that a clause is assertive using decision levels (UIP [22]). For PB constraints, we do not have for the moment an efficient incremental way to do it. As a consequence, our solver spends 50% of its running time to detect whether the constraint derived during conflict analysis is assertive or not!

After additional analysis of the results of SAT4JPseudo behavior during the third PB evaluation, we found a fourth and most likely the main reason of our solver defficiency.

Applying cutting planes means learning new PB constraints. On examples for which SAT4Jpseudo CP did perform poorly, we noticed that the speed of the solver was decreasing when learning new constraints. This is in contrast with SAT4Jpseudo resolution and traditional SAT solvers since the cost of learning became negligible with the introduction of the watched literal data structure. We are currently investigating two ways to solve that problem:

- is learning really necessary for a PB solver? preliminary results of a version of SAT4JPseudo without learning (constraints resulting from conflict analysis are just kept as a reason for propagating some truth values) show that learning does not play a central role in Cutting Planes based solving.
- a tradeoff would be to learn clauses derived from the PB constraint found during conflict analysis. We haven't experimented that scheme yet.

4 Conclusion

We summarized the results of CDCL like PB solvers during the PB 05 and PB 06 evaluations. We detailed the behavior of our solver SAT4JPseudo on some families of benchmarks from the PB06 evaluation, and tried to make a quick comparative approach based on the features implemented in the selected solvers. We hope that such a competition could continue in the next years with an increasing number of benchmarks, allowing to better understand the ways to deal with such specific constraints.

Our own solver did not generally give good results. But we identified one main bottleneck in our approach: learning PB constraints during the search slows down the solver. We are currently unsure that it is a good idea to build a pure PB solver based on an extension of the CDCL framework. Indeed, preliminary results show that disabling learning in our CP based solver does not change significantly the behavior of the solver.

The current benchmarks used in the PB evaluations are not pure PB problems: they are hybrid problems with clauses, cardinality constraints and pseudo boolean constraints. We believe that clause learning is necessary to deal with the clausal part of those benchmarks. As a consequence, we need to find a way to mix cutting planes based reasoning and clause learning in order to build a good performer in the PB evaluations. Our future work will follow that direction.

Ackowledgments

We would like to thank Olivier Roussel and Vasco Manquinho to have allowed us to use the pseudo-boolean evaluation framework for additional tests, and thank again Olivier Roussel for the fruitful discussions we had with him. We would also like to thank the anonymous reviewers for their valuable comments which helped us to improve the paper.

This work has been partially supported by the Région Nord-Pas-de-Calais within the Cocoa project.

References

1. F. Aloul, A. Ramani, I. Markov, and K. Sakallah. Generic ILP versus Specialized 0-1 ILP: an update. In *Proceedings of ICCAD'02*, pages 450–457, 2002.
2. Peter Barth. A Davis-Putnam based enumeration algorithm for linear pseudo-Boolean optimization. Technical Report MPI–I–95–2–003, Saarbrücken, 1995.
3. Don S. Batory. Feature models, grammars, and propositional formulas. In J. Henk Obbink and Klaus Pohl, editors, *SPLC*, volume 3714 of *Lecture Notes in Computer Science*, pages 7–20. Springer, 2005.
4. B. Benhamou, L. Sais, and P. Siegel. Two proof procedures for a cardinality based language in propositional calculus. In *11th Annual Symposium on Theoretical Aspects of Computer Science (STACS), volume 775 LNCS*, pages 71–82, Caen, France, 1994.
5. Alexander Bockmayr and Friedrich Eisenbrand. Combining logic and optimization in cutting plane theory. In Hélène Kirchner and Christophe Ringeissen, editors, *FroCos*, volume 1794 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 2000.
6. Donald Chai and Andreas Kuehlmann. A fast pseudo-boolean constraint solver. In *ACM/IEEE Design Automation Conference (DAC'03)*, pages 830–835, Anaheim, CA, 2003.
7. Michael Codish, Peter Schneider-Kamp, Vitaly Lagoon, René Thiemann, and Jürgen Giesl. Sat solving for argument filterings. In *International Conference on Logic for Programming, Artificial Intelligence and Reasoning*, volume 4246 of *Lecture Notes in Artificial Intelligence*, pages 30–44. Springer-Verlag, November 2006.
8. Heidi Dixon. *Automated Pseudo-Boolean Inference within the DPLL framework.* PhD thesis, University of Oregon, 2004.
9. Bruno Dutertre and Leonardo Mendonça de Moura. A fast linear-arithmetic solver for dpll(t). In Thomas Ball and Robert B. Jones, editors, *CAV*, volume 4144 of *Lecture Notes in Computer Science*, pages 81–94. Springer, 2006.
10. Niklas Eén Niklas Sörensson. An extensible sat-solver. In *Proceedings of the Sixth International Conference on Theory and Applications of Satisfiability Testing, LNCS 2919*, pages 502–518, 2003.
11. J. Endrullis. Jambox. http://joerg.endrullis.de/.
12. J. Giesl, R. Thiemann, P. Schneider-Kamp, and S. Falke. Automated termination proofs with aprove. In *Proceedings of the 15th International Conference on Rewriting Techniques and Applications (RTA-04), Lecture Notes in Computer Science 3091*, pages 210–220, Aachen, Germany, 2004.
13. Heidi E. Dixon Matthew L. Ginsberg. Inference methods for a pseudo-boolean satisfiability solver. In *Proceedings of The Eighteenth National Conference on Artificial Intelligence (AAAI-2002)*, pages 635–640, 2002.
14. R. Gomory. Outline of an algorithm for integer solutions to linar programs. 64:275–278, 1958.
15. Dieter Hofbauer and Johannes Waldmann. Termination of string rewriting with matrix interpretations. In Frank Pfenning, editor, *RTA*, volume 4098 of *Lecture Notes in Computer Science*, pages 328–342. Springer, 2006.
16. J. N. Hooker. Generalized resolution and cutting planes. *Ann. Oper. Res.*, 12(1-4):217–239, 1988.
17. Daniel Jackson and Mandana Vaziri. Finding bugs with a constraint solver. In *ISSTA*, pages 14–25, 2000.
18. Daniel Le Berre, Anne Parrain, and Olivier Roussel. The long way from conflict driven clause learning to conflict driven constraint learning. In *Proceedings of Guangzhou Symposium on SATisfiability and its Applications*, September 2004.
19. Fabio Mancinelli, Jaap Boender, Roberto di Cosmo, Jérôme Vouillon, Berke Durak, Xavier Leroy, and Ralf Treinen. Managing the complexity of large free and open source package-based software distributions. In *Proceedings of the 21st IEEE/ACM International Conference on Automated Software Engineering (ASE06)*, pages 199–208, Tokyo, JAPAN, september 2006. IEEE Computer Society Press.

20. V. M. Manquinho and J. Marques-Silva. On using cutting planes in pseudo-boolean optimization. *Journal on Satisfiability, Boolean Modeling and Computation (JSAT)*, 2:209–219, 2006.
21. C. Marché and H. Zantema. The termination competition 2006. http://www.lri.fr/ marche/termination-competition/2006/reportCompetition2006.pdf.
22. Joao P. Marques-Silva and Karem A. Sakallah. GRASP - A New Search Algorithm for Satisfiability. In *Proceedings of IEEE/ACM International Conference on Computer-Aided Design*, pages 220–227, November 1996.
23. Matthew W. Moskewicz, Conor F. Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. Chaff: Engineering an Efficient SAT Solver. In *Proceedings of the 38th Design Automation Conference (DAC'01)*, pages 530–535, 2001.
24. Robert Nieuwenhuis and Albert Oliveras. Decision procedures for sat, sat modulo theories and beyond. the barcelogictools. In Geoff Sutcliffe and Andrei Voronkov, editors, *LPAR*, volume 3835 of *Lecture Notes in Computer Science*, pages 23–46. Springer, 2005.
25. Hossein M. Sheini and Karem A. Sakallah. Pueblo: A Hybrid Pseudo-Boolean SAT Solver. *Journal on Satisfiability, Boolean Modeling and Computation (JSAT)*, 2:165–182, 2006.
26. Niklas Eén Niklas Sörensson. Translating pseudo-boolean constraints into sat. *Journal on Satisfiability, Boolean Modeling and Computation (JSAT)*, 2:1–26, 2006.
27. Manquinho V. and Roussel O. The first evaluation of pseudo-boolean solvers (pb'05). *Journal on Satisfiability, Boolean Modeling and Computation (JSAT)*, 2:103–143, 2006.
28. Johannes Waldmann. Matchbox: A tool for match-bounded string rewriting. In *Proceedings of the 15th International Conference on Rewriting Techniques and Applications (RTA-04), Lecture Notes in Computer Science 3091*, pages 85–94, Aachen, Germany, 2004.
29. Yichen Xie and Alexander Aiken. Saturn: A sat-based tool for bug detection. In Kousha Etessami and Sriram K. Rajamani, editors, *CAV*, volume 3576 of *Lecture Notes in Computer Science*, pages 139–143. Springer, 2005.
30. Yichen Xie and Alexander Aiken. Scalable error detection using boolean satisfiability. In Jens Palsberg and Martín Abadi, editors, *POPL*, pages 351–363. ACM, 2005.