# *Pueblo*: A Modern Pseudo-Boolean SAT Solver

Hossein M. Sheini and Karem A. Sakallah
*Department of Electrical Engineering and Computer Science*
*University of Michigan*
*Ann Arbor, MI 48109-2122*
*{hsheini, karem}@eecs.umich.edu*

## Abstract

*This paper introduces a new SAT solver that integrates logic-based reasoning and integer programming methods to systems of CNF and PB constraints. Its novel features include an efficient PB literal watching strategy and several PB learning methods that take advantage of the pruning power of PB constraints while minimizing their overhead.*

## 1. Introduction

Modern backtrack search SAT solvers augment the basic DPLL procedure with powerful conflict-based learning [5] and efficient Boolean Constraint Propagation schemes [6].

The closely-related 0-1 integer programming (IP) problem has also been studied extensively. In particular, the extension of SAT techniques to systems of CNF and so-called pseudo Boolean (PB) constraints was addressed in [1]. Algorithms that combine the logic-based reasoning techniques of CNF SAT and the constraint relaxation and polyhedral analysis ("cutting planes") methods of IP were also explored with some success in [2].

In this paper we introduce Pueblo, a new CNF/PB SAT solver that integrates logic-based reasoning and new techniques to handle systems of CNF and PB constraints. Pueblo incorporates a novel watched-literal strategy and features several learning strategies including two that combine conflict-based CNF learning and cutting plane PB learning.

After covering some preliminaries, in Section 3 we describe our PB propagation method. Section 4 details various PB learning strategies. Experimental results are reported in Section 5. Conclusions and future work are presented in Section 6.

## 2. Preliminaries

A linear pseudo-Boolean (PB) constraint is said to be in normal form when expressed as:

$$\sum_{i=1}^{n} a_i \dot{x}_i \geq b \qquad a_i, b \in \mathbb{Z}^+, \dot{x}_i \in \{0,1\} \tag{1}$$

where $\dot{x}_i$ denotes $x_i$ or $x_i'$. A PB constraint in which some coefficients are negative can be transformed to normal form by noting that $\dot{x}_i' = 1 - \dot{x}_i$. In general, a PB constraint is equivalent to a large, potentially exponential, number of CNF clauses [1]. When the right-hand side and all left-hand side coefficients are equal to 1, however, a PB constraint is equivalent to a single CNF clause.

## 3. Pseudo-Boolean Propagation

In [6] it was shown that watching just two literals per clause, regardless of clause size, is sufficient to detect when the clause becomes unit. The watched-literal concept was extended in [2] to handle PB constraints. The basic idea is to watch the fewest number of non-false literals such that when the unassigned watched literal with the largest coefficient is set to false a) the constraint is still guaranteed to be satisfied and b) the constraint can identify the literals that must now be implied to true. Specifically, let $T$ and $U$ denote the sets of true and unassigned literals in the constraint, and let $W \subseteq T \cup U$ denote the set of watched literals. We will refer to $W$ as the *watch list* and to the sum of coefficients of the watched literals as the *watched sum*, i.e., $S_W = \sum_{\dot{x}_i \in W} a_i$ We also introduce $a_{\max}$ defined as: $a_{\max} = \max\{a_i | \dot{x}_i \in U\}$

The invariant that must be maintained to detect when the PB constraint becomes unit can now be succinctly expressed as: $S_W \geq b + a_{\max}$. When a watched literal is set to false, it must be removed from the watch list and replaced by one or more non-false literals to maintain the above invariant. When that is no longer possible, the constraint becomes unit and any unassigned watched literal whose coefficient $a$ satisfies the *unit constraint* condition $a > S_W - b$ must now be implied to true.

Empirical evaluation of this procedure suggests that about two thirds of its run time is spent in updating $a_{\max}$ and corroborates the conclusion in [2] that the "watching scheme is beneficial for clauses and cardinality constraints, but not for LPB constraints; therefore we use counters to implement Boolean constraint propagation on LPB constraints." In other words, PB constraints are processed using a ***Watch All Literals*** strategy similar to that of PBS [1].

Further analysis of the data, however, suggests a potentially more efficient *hybrid* watching strategy that differentiates between the literals with unit- and non-unit coefficients in the same PB constraint. Specifically, let $L$ denote the set of literals whose coefficients are greater than 1 (the *large* literals) and let $C$ denote those literals whose coefficients are equal to 1 (the *cheap* literals). Much of propagation time can be eliminated by applying the above procedure only to the literals in the (relatively small) $L$ set. To achieve this, the computation of $a_{\max}$ is modified to $a_{\max} = \max\{a_i | \dot{x}_i \in L \cap U\}$ and $a_{\max} = 1$ if $L \cap U = \varnothing$.

Furthermore, the watch list $W$ is modified to always include all of the literals in $C$. This modification requires a slight change to the manner in which the watched sum is calculated since the watched literals are no longer guaranteed to be true or unassigned. Specifically, the watched sum must now be decremented by 1 when a $C$ literal is set to false, and incremented by 1 when a $C$ literal is unassigned from false.

## 4. Pseudo-Boolean Learning

Generating and recording a so-called conflict-induced clause [5], enables the solver to prune away a portion of the search

1530-1591/05 $20.00 © 2005 IEEE

space thus avoiding a recurrence of the same conflict. We will refer to this style of learning as *CNF learning*. An alternative approach, referred to as *PB learning* in this paper, based on cutting plane methods [3], can be used to create a conflict-induced PB constraint instead. For details of this method, the readers are referred to [2].

Since PB constraints are generally more expressive than CNF clauses, a learned PB constraint has the potential of pruning more of the search space than a CNF clause. However, the steep overhead of manipulating PB constraints can more than offset their pruning benefits. (the high cost of unlimited PB learning and propagation is shown in Figure 1). Thus, an adaptive approach that combines CNF and PB learning, and introduces PB constraints selectively, might be superior to either approach alone. We describe next two variations on this theme.

**Scheme 1: Learn Strong PB Constraints.** In this scheme, after detecting a conflict, both CNF and PB Learning procedures are followed simultaneously. A PB constraint is learned and recorded instead of a CNF clause if and only if all the following three conditions hold:

1. it is unit; it could be the case that the cutting plane generated from two constraints (conflicting and implying constraints) is not in conflict anymore due to over-satisfaction of the implying constraint.
2. it corresponds to more than just a single CNF clause;
3. the number of its large literals (those in the L set) is less than a given threshold;

**Scheme 2: Convert Learned PB Constraint to CNF.** In this scheme, the learned PB constraint is recorded but not used in BCP. Rather, it is viewed as a compact representation of a set of CNF clauses, subsets of which can be extracted as needed during the search. Extraction of suitable CNF clauses from the PB constraint is carried out using a simple knapsack algorithm. The rationale for this scheme is to capitalize on the pruning power of the learned PB constraint without incurring its high propagation overhead. For instance, learning $6x_1 + 5x_2 + 4x_3 + 3x_4 + 2x_5 + 2x_6 \geq 12$ where $x_2$ and $x_4$ are false at the backtrack level, will result in learning the following clauses: $(x_1 + x_2)$,$(x_2 + x_3 + x_4)$,$(x_2 + x_4 + x_5 + x_6)$ The first two clauses are unit and would result in the same implications as the original PB constraint. The PB constraint, on the other hand, is kept aside and retouched later during the search to extract other CNF clauses which could be more useful based on the variable assignments in effect at that time.

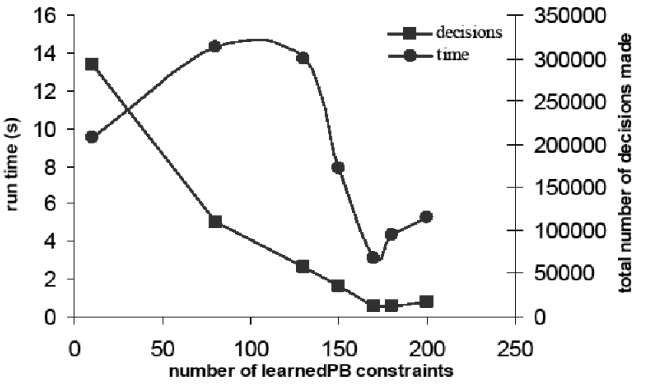


**Figure 1: Effect of PB learning on number of decisions and run-time with a restriction on the maximum allowable number of learned PB constraints (X-axis) in s4-3-2pb [1].**

## 5. Experimental Results

We conducted several experiments to evaluate the strategies described above using Pueblo. Pueblo is built on top of MiniSAT [4] and extends its VSIDS decision heuristic and its clause removal mechanism to PB constraints. It also adopts hybrid propagation strategy as explained earlier. All experiments were conducted on Pentium-IV 2.8GHz - 1GB of RAM.

Table 1 depicts a comparison of Pueblo and galena [2] PB learning strategies on a set of benchmarks including the Routing instances of [1]. Overall, scheme 1 performs robustly and compares favorably to the cardinality strategy of galena. All these methods perform much better than CNF learning in these types of problems. In these benchmarks, due to large number of unit-coefficient variables, the positive effects of the hybrid propagation strategy is more evident.

**Table 1: Different PB learning strategies**

| Benchmark | Time (sec) | | | |
|---|---|---|---|---|
| | Pueblo | | Galena | |
| | CNF | Sch. 1 | Card. | LPB |
| s4-3-2pb (SAT) | 4.78 | 0.57 | 0.50 | 0.70 |
| s4-3-3pb (SAT) | 3.35 | 0.64 | 0.50 | 0.70 |
| s4-3-4pb (SAT) | 2.54 | 0.05 | 0.60 | 1.70 |
| s4-3-5pb (SAT) | 3.12 | 0.73 | 0.80 | 0.15 |
| fpga11_9 (SAT) | 41.19 | 0.09 | 43.39 | >1000 |
| fpga12_14 (UNS) | >1000 | 0 | 0.10 | 0 |
| fpga15_10 (SAT) | >1000 | 7.45 | >1000 | 85.40 |
| fpga15_14 (SAT) | >1000 | 0.02 | 0.37 | 1.14 |

## 6. Conclusions and Future Work

As we learn more about the trade-offs involved in combining logic-based reasoning and IP methods, we will be able to develop effective integration strategies that outperform individual techniques. The concepts described in this paper do not exhaust all the possibilities for taking advantage of the pruning power of PB constraints while minimizing their computational overhead. Other ways of generating cutting planes, for example, that are provably superior to current approaches should be investigated.

## Acknowledgment

This work was funded in part by the National Science Foundation under ITR grant No. 0205288.

## References

[1] F.A. Aloul, A. Ramani, I.L. Markov, K.A. Sakallah, "Generic ILP versus specialized 0-1 ILP: An Update", ICCAD'02 pp. 450-457

[2] D. Chai, A. Kuehlmann, "A Fast Pseudo-Boolean Constraint Solver", DAC'03 pp. 830-835

[3] V. Chvátal, "Edmonds Polytopes and a Hierarchy of Combinatorial Problems", Discr. Math., vol. 4, pp. 305-307, 1973.

[4] N. Eén, and N. Sörensson, "An Extensible SAT-solver," SAT'03 pp 502-508.

[5] J.P. Marques-Silva and K.A. Sakallah, "GRASP: a search algorithm for propositional satisfiability", IEEE Transaction on Computers, vol. 48/5, pp. 506-521, 1999.

[6] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Chaff: Engineering an Efficient SAT Solver", DAC'01 pp. 530-535