Learning Techniques for Pseudo-Boolean Solving and Optimization

José Faustino Fragoso Freminin dos Santos

September 29, 2008

Abstract

The extension of conflict-based learning from Propositional Satisfiability (SAT) solvers to Pseudo-Boolean (PB) solvers comprises several different learning schemes. However, it is not commonly agreed among the research community which learning scheme should be used in PB solvers. Hence, this work presents a contribution by providing an exhaustive comparative study between several different learning schemes in a common platform. Results for a large set of benchmarks are presented for the different learning schemes, which were implemented on bsolo, a state of the art PB solver.

1 Introduction

The use of an algorithm based on the Davis Putnam Longemann Loveland (DPLL) Procedure for solving the Pseudo-Boolean Optimization (PBO) problem was proposed by P. Barth in 1995 in its seminal paper “A Davis-Putnam based enumeration algorithm for linear pseudo-Boolean optimization” [1]. The enormous progresses of SAT solvers contributed to the development of more powerful pseudo-Boolean solvers which exhibit the same structure of a typical SAT solver. Moreover, PB solvers also benefit from all the knowledge and experience from Integer Linear Programming research. The contributions of SAT solvers to PB solvers, particularly the introduction of conflict-based learning, are the main topic of this thesis.

2 Problem Definition

A Boolean variable is any symbol to which we can assign one of the truth values 0 and 1, also denoted by FALSE and TRUE, respectively. A literal is a Boolean variable, or the complement of a Boolean variable. An integer pseudo-Boolean function maps n Boolean variables to an integer. A linear Pseudo-Boolean constraint (commonly denoted as PB constraint or LPB constraint) over a set of
Boolean variables $X = \{x_1, \cdots, x_n\}$ is an inequality that has the following form:

$$\sum_{j=1}^{n} a_j \cdot l_j \triangleright b$$

such that for each $j \in \{1, \cdots, n\}$, $a_j$ is an integer coefficient and $l_j$ is a literal, $b$ is an integer coefficient and $\triangleright$ is one of the common relational operators (=, $\geq$, $\leq$, $>$ and $<$). The right side of the constraint is denoted as the degree of the constraint\(^1\). The addition operator and the other relational operators have their usual arithmetic meaning.

A linear PB constraint is said to be in normal form when expressed as:

$$\sum_{i=1}^{n} a_i \cdot l_i \geq b$$

such that for each $i \in \{1, \cdots, n\}$, $a_i \in Z^+$ and $l_i$ is a literal and $b \in Z^+$. Any PB constraint $w$ can be converted into the normal form in linear time [3].

A cardinality constraint is a constraint on the number of literals which are true among a given set of literals. Every cardinality constraint can be expressed in the following way:

$$\sum_i l_i \geq b$$

(1)

A clause is a cardinality constraint such that its right hand side is equal to one.

A Boolean assignment is a mapping $A_X: X' \rightarrow \{0, 1\}$, where $X' \subseteq X$. If $X = X'$ then we say that the assignment $A_X$ is complete, otherwise $A_X$ is said to be a partial assignment.

Given a partial assignment $A_X$ and a constraint $w$, we define $s$ as the slack of $w$ under $A_X$, in the following way:

$$s = \sum_{i \neq 0} a_i - b$$

Given a partial assignment $A_X$ and a constraint $w = \sum_i a_i \cdot l_i \geq b$, $w$ is said to be:

- satisfied if $\sum_{i=1}^{l_i} a_i \geq b$;
- unsatisfied if $\sum_{i \neq 0} a_i < b$, that is, $s < 0$;
- unresolved if $\sum_{i \neq 0} a_i \geq b$, that is, $s \geq 0$;
- unit if $s < a_{\text{max}}^w$.

Where:

$$a_{\text{max}}^w = \max\{a_i : l_i \text{ is an unassigned literal in } w\}$$

\(^1\)It is also commonly referred to as rhs.
Note that, given an assignment $A_X$, a Pseudo-Boolean constraint is said to be \textit{unit} under $A_X$, if it is unresolved under that assignment and at least one of its literals must be assigned to 1 for the constraint to be satisfied.

Given a set of PB constraints $W$ over a set of Boolean variables $X$, we say that $W$ is satisfiable if there is an assignment $A_X$, such that all constraints in $W$ are satisfied under $A_X$.

The pseudo-Boolean Satisfiability (PB-SAT) problem asks whether a given set of PB constraints is satisfiable. In formal language:

$$\text{PB-SAT} = \{ \langle W \rangle : W \text{ is a satisfiable set of PB constraints} \}$$

Where $\langle W \rangle$ denotes the encoding of $W$.

The pseudo-Boolean Optimization problem (PBO) consists in finding a satisfying assignment to a set of PB constraints that minimizes a given pseudo-Boolean objective function.

Since the main goal of this work is the study of the conflict analysis procedure, we will only consider the PB-SAT problem.

### 2.1 Inference Rules

One of the most important features of modern PB-SAT solvers is the ability to derive a new PB constraint from a set of PB constraints [3, 8]. In this section we will only state the more important inference rules.

- **Cutting Planes:**

  $$\sum_i a_i \cdot l_i \geq b$$
  $$\sum_i c_i \cdot l_i \geq d$$
  $$\alpha > 0$$
  $$\beta > 0$$
  $$\sum_i (\alpha \cdot a_i + \beta \cdot c_i) \cdot l_i \geq (\alpha \cdot b + \beta \cdot d)$$

  Generally, when applying this rule, $\alpha$ and $\beta$ are chosen in order to eliminate one variable which occurs simultaneously in both constraints (in one of them it must occur its corresponding positive literal and in the other its negative one).

- **Coefficient Reduction:**

  $$\sum_i a_i \cdot l_i \geq b$$
  $$a_j > 0$$
  $$\sum_{i \neq j} a_i \cdot l_i \geq (b - a_j)$$

- **Cardinality Constraint Reduction**

  $$\sum_{i=1}^{n} a_i \cdot l_i \geq b$$
  $$\sum_{i=1}^{\beta-1} a_i < b \leq \sum_{i=1}^{\beta} a_i$$
  $$\sum_{i=1}^{\beta} l_i \geq \beta$$
3 A Generic PB-SAT Algorithm

Given a PB-SAT formula $W$, an algorithm for the pseudo-Boolean satisfiabilityproblem must decide if there exists an assignment $A_X$ that satisfies $W$, denoted as a *satisfying assignment*. Starting from an empty truth assignment, a backtrack algorithm examines the space of truth assignments in order to find a satisfying assignment. It organizes the search for a satisfying assignment by implicitly maintaining a *decision tree*. Each node in the decision tree specifies an elective assignment for an unassigned variable called a *decision assignment*. A *decision level* is associated with each decision assignment to denote its depth in the decision tree.

A generic PB-SAT algorithm has the same structure of a generic SAT algorithm. As such, in each iteration of the search process, the algorithm performs the following steps [6]:

1. Extend the current partial assignment by making a *decision assignment*.

2. Extend the current partial assignment performing BCP (Boolean Constraint Propagation [4]). The assignments made in this step are referred to as *implied assignments*.

   The deduction process may lead to the identification of one or more unsatisfied constraints, which imply that the current assignment is not a satisfying one. Such occurrence is called a *conflict* and the associated unsatisfying assignment is called a *conflicting assignment*, the constraints which are unsatisfied under the current assignment are referred to as *conflicting constraints*.

3. If the current assignment is a conflicting assignment another one must be tried. Therefore, the current one must be undone. The backtracking mechanism enables the algorithm to retreat from regions of the search space that do not correspond to satisfying assignments.

4 Conflict Analysis

Let the assignment of a variable $x$ be implied due to a constraint $w$, which is referred to as the *antecedent constraint* [6] of $x$. The *antecedent assignment* of $x$, denoted as $A^w(x)$, is defined as the set of assignments to variables corresponding to false literals in $w$. Similarly, when a PB constraint $w$ becomes unsatisfied, the antecedent assignment of its corresponding conflict, $A^w(k)$, will be the set of all assignments corresponding to false literals in $w$. The implication relationships of variable assignments during the PB-SAT solving process can be expressed as an *implication graph* [6].

4.1 Clause Learning

When a logical conflict arises, the implication sequence leading to the conflict is analysed to determine the variable assignments that are responsible for the
conflict. The conjunction of these assignments represents a sufficient condition for the conflict to arise and, as such, its negation must be consistent [6] with the PB formula. This new constraint, called the conflict constraint, is then added to the PB formula in order to help pruning the search space.

This kind of implication graph analysis can be viewed as a sequence of resolution steps guided by the original implication sequence in reverse order. If we denote the violated constraint by \( V \) and the unit implying constraints leading to the conflict by \( U_1, \ldots, U_k \), the conflict analysis procedure can be expressed as follows:

\[
R_1 = \text{resolve} (V; U_1; x_1) \\
R_2 = \text{resolve} (R_1; U_2; x_2) \\
\vdots \\
R_k = \text{resolve} (R_{k-1}; U_k; x_k)
\]

Where \( R_i \) denotes the resolvent at each resolution step (also denoted as Accumulator Constraint) and \( x_i \) the variable to be eliminated. Note that, before each resolution step, the coefficient reduction rule must be used to eliminate all positive and unassigned literals except for the implied literal. It must be stressed that the implied variables \( x_1, \ldots, x_k \) are considered in reverse order and as such after each resolution step the corresponding eliminated variable can be removed from the current assignment.

### 4.2 Generalizing Conflict Analysis

#### 4.2.1 General PB Learning

The operation on PB constraints which corresponds to clause resolution is the cutting plane operation. As such, to learn a general PB constraint, the algorithm must perform a sequence of cutting plane steps instead of a sequence of resolution steps. Again, in each cutting plane step one implied variable is eliminated. The implied variables are considered in reverse order, i.e., we start at the last assignment and finish at the first UIP (Unique Implication Point) [6]. A UIP cut is used since it is considered the best one [9].

When a conflict occurs the algorithm uses the learned constraints to:

- Determine to which level it must backtrack.
- Imply a new assignment after backtracking.

Therefore each learned constraint must exhibit two properties:

- It must be unsatisfied under the current assignment.
- It must be an assertive constraint (that is, it must become unit after backtracking).

However, when performing cutting planes, the resulting constraint may not be unsatisfied under the current assignment [3]. In this situation, the learned
constraint will not be able to flip any variables after erasing all the assignments made at the current decision level, which is essential for driving the search forward.

Consider the application of a cutting plane step to two arbitrary constraints \(w_1\) with slack \(s_1\) and \(w_2\) with slack \(s_2\) and suppose \(\alpha\) and \(\beta\) are used as the multiplying factors. In this situation, the slack of the resulting constraint, here denoted by \(s_r\), is given by linearly combining the slacks of \(w_1\) and \(w_2\):

\[
s_r = (\alpha \cdot s_1) + (\beta \cdot s_2)
\]

As such, before the application of each cutting plane step, the learning algorithm verifies if the resulting constraint is still unsatisfied under the current assignment. If it is not, the implied constraint must be reduced to lower its slack \([3, 8]\). This process is guaranteed to work since the repeated reduction of constraints will eventually lead to a simple clause with slack 0.

It is not known any efficient incremental way to do detect assertive constraints when using a general PB learning scheme \([2]\).

### 4.2.2 A Hybrid Learning Scheme

Sheini and Sakallah \([8]\) noted that any solver which performs PB learning can be modified to additionally perform clause learning with no significant extra overhead. Moreover, despite the greater pruning power of PB learning, clause learning has its own advantages: it always produces an assertive constraint and it does not compromise as heavily the propagation procedure as general PB learning. As such, in their solver \(Pueblo\), they implement a hybrid learning method.

### 4.2.3 Cardinality Constraint Learning

Learning general PB constraints slows down the deduction procedure because the watch literal strategy is not as efficient with general PB constraints as it is with clauses or cardinality constraints \([3, 7]\). Note that in a clause, as well as in a cardinality constraint, it is only necessary to watch a fixed number of literals, whereas in a general PB constraint the number of watched literals varies during the execution of the algorithm.

In \(Galena\), Chai and Kuehlmann choose to learn cardinality constraints instead of general PB constraints. The method used to learn cardinality constraints in \(Galena\) is similar to the method used to learn general PB constraints. Additionally, it is introduced a post-reduction procedure, which converts the learned constraint into a weaker cardinality constraint.

### 4.2.4 A Generic Conflict Analysis Procedure

Algorithm 1 presents the pseudo-code for computing the conflict-induced pseudo-Boolean constraint \(w(k)\). This algorithm performs a sequence of cutting plane steps, starting from the unsatisfied constraint \(w(c)\).
Algorithm 1 Generic Pseudo-Boolean Learning Algorithm

// W corresponds to the set of constraints in the PB formula
// w(c) corresponds to the initial conflicting constraint
V ← \{x_i \mid x_i \text{ corresponds to a false literal in } w(k)\};
w(k) ← reduce1(w(c))

while TRUE do
    x_i ← removeNext(V);
    w_i ← implyingConstraint(x_i);
    if (w_i = NULL || V = 1) then
        w(k) ← reduce3(w(k));
        Add w(k) to W;
        btLevel ← assertingLevel(w(k));
        if btLevel < 0 then
            return CONFLICT;
        else
            backtrack(btLevel);
            return NOCONFLICT;
        end if
    else
        w_i ← reduce2(w_i, w(k));
        w(k) ← cutResolve(w(k), w_i', x_i);
        V ← V \ \{x_i\} \cup \{x_k \mid x_k \text{ corresponds to a false literal in } w_i'\}
    end if
end while
First of all, it is important to note that the guard of the external if forces the backward traversal of the implication graph to stop at the first UIP.

Before, each cutting plane step a coefficient reduction is applied to the implying constraint. In order to implement a clause learning scheme, functions reduce1 and reduce2 must eliminate all non-negative literals in \( w_c \) and \( w_i \), respectively, except for the implied literal and then trivially reduce the obtained contraint to a clause. In order to implement a general PB learning scheme, function reduce2 must eliminate only enough non-negative literals in \( w_i \) to guarantee that after the cutting plane step, the resulting constraint is unsatisfied (\( w_k \) is given as input so that this function can determine the extent of the reduction). If function reduce3 implements a cardinality constraint reduction than algorithm 1 corresponds to a cardinality constraint learning scheme.

4.2.5 Backtracking

Suppose the conflict analysis procedure is always able to learn an assertive constraint. After learning such a constraint, the algorithm can use it to determine to which decision level it must backtrack. It must backtrack to a decision level at which the learned constraint becomes unit [3, 6]. However, a general PB constraint may become unit in multiple decision levels. The majority of the solvers which perform general PB learning\(^2\) implement the largest possible backtrack.

5 Experimental Results

This section presents the experimental results of applying different learning schemes to the small integer\(^3\) non-optimization benchmarks from PB’07 evaluation [5]. All these learning schemes were implemented on top of bsolo, a state of the art PB solver. All versions of the solver were run on a Intel Xeon 5160 server (3.0GHZ, 1333Mhz, 4GB) running Red Hat Enterprise Linux WS 4. The CPU time limit for each instance was set to 1800 seconds. The results in each cell denote the number of benchmarks SAT/UNSAT/UNKNOWN for each class of instances.

In table 1 the results of all different learning schemes are presented. Version CL1 implements a clause learning scheme. Versions PB1.2 and PB2 perform general PB learning. Additionally, version PB2 performs a final sequence of coefficient reductions in order to eliminate all non-false literals. On top of versions PB1.2 we implemented a final cardinality constraint reduction corresponding to version CARD1.2. In version COMB it is applied an initial classification step in order to select the best fitting learning scheme.

In table 2 the results of the best known solvers can be checked and compared with our best version.

\(^2\)Clauses and cardinality constraints can only become unit in one decision level.

\(^3\)All coefficients are smaller than \(2^{30}\).
Table 1: An overview of the results of all the different learning schemes

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>CL1</th>
<th>CARD1.2</th>
<th>PB1.2</th>
<th>PB2</th>
<th>COMB</th>
</tr>
</thead>
<tbody>
<tr>
<td>armies</td>
<td>5/0/7</td>
<td>6/0/6</td>
<td>5/0/7</td>
<td>7/0/5</td>
<td></td>
</tr>
<tr>
<td>dbst</td>
<td>15/0/0</td>
<td>15/0/0</td>
<td>15/0/0</td>
<td>15/0/0</td>
<td>15/0/0</td>
</tr>
<tr>
<td>FPGA</td>
<td>36/2/7</td>
<td>36/2/1</td>
<td>35/1/3</td>
<td>35/1/21</td>
<td>36/21/0</td>
</tr>
<tr>
<td>pigeon</td>
<td>0/2/7</td>
<td>0/19/7</td>
<td>0/18/2</td>
<td>0/17/19</td>
<td>0/19/7</td>
</tr>
<tr>
<td>progressive party</td>
<td>4/0/2</td>
<td>4/0/2</td>
<td>4/0/2</td>
<td>4/0/2</td>
<td>4/0/2</td>
</tr>
<tr>
<td>reduced</td>
<td>14/0/0</td>
<td>14/0/0</td>
<td>14/0/0</td>
<td>14/0/0</td>
<td>14/0/0</td>
</tr>
<tr>
<td>robin</td>
<td>3/0/3</td>
<td>2/0/4</td>
<td>3/0/3</td>
<td>4/0/2</td>
<td>4/0/2</td>
</tr>
<tr>
<td>tsp</td>
<td>40/37/21</td>
<td>40/44/16</td>
<td>40/55/55</td>
<td>40/83/17</td>
<td>40/89/11</td>
</tr>
<tr>
<td>uclid</td>
<td>1/39/10</td>
<td>1/43/6</td>
<td>1/42/7</td>
<td>1/43/6</td>
<td>1/41/8</td>
</tr>
<tr>
<td>vdw</td>
<td>1/0/4</td>
<td>1/0/4</td>
<td>1/0/4</td>
<td>1/0/4</td>
<td>1/0/4</td>
</tr>
<tr>
<td>wnqueen</td>
<td>32/68/0</td>
<td>32/68/0</td>
<td>32/68/0</td>
<td>32/68/0</td>
<td>32/68/0</td>
</tr>
<tr>
<td>Total</td>
<td>151/144/90</td>
<td>151/195/39</td>
<td>151/153/81</td>
<td>151/156/78</td>
<td>153/198/34</td>
</tr>
</tbody>
</table>

Table 2: The Results of other solvers

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Isolo</th>
<th>Pueblo</th>
<th>minisat+</th>
<th>PBS4</th>
</tr>
</thead>
<tbody>
<tr>
<td>armies</td>
<td>7/0/5</td>
<td>6/0/6</td>
<td>8/0/4</td>
<td>9/0/3</td>
</tr>
<tr>
<td>dbst</td>
<td>15/0/0</td>
<td>15/0/0</td>
<td>7/0/8</td>
<td>15/0/0</td>
</tr>
<tr>
<td>FPGA</td>
<td>36/21/0</td>
<td>36/21/0</td>
<td>33/21/0</td>
<td>26/21/0</td>
</tr>
<tr>
<td>pigeon</td>
<td>0/19/4</td>
<td>0/13/7</td>
<td>0/2/18</td>
<td>0/2/0</td>
</tr>
<tr>
<td>progressive party</td>
<td>3/0/3</td>
<td>6/0/0</td>
<td>5/0/1</td>
<td>3/0/5</td>
</tr>
<tr>
<td>reduced</td>
<td>14/0/0</td>
<td>18/0/0</td>
<td>13/0/0</td>
<td>13/0/0</td>
</tr>
<tr>
<td>robin</td>
<td>4/0/2</td>
<td>3/0/3</td>
<td>4/0/2</td>
<td>3/0/3</td>
</tr>
<tr>
<td>tsp</td>
<td>40/49/11</td>
<td>40/60/0</td>
<td>39/46/15</td>
<td>40/52/8</td>
</tr>
<tr>
<td>uclid</td>
<td>1/41/8</td>
<td>1/42/7</td>
<td>1/46/3</td>
<td>1/44/5</td>
</tr>
<tr>
<td>vdw</td>
<td>1/0/4</td>
<td>1/0/4</td>
<td>1/0/4</td>
<td>1/0/4</td>
</tr>
<tr>
<td>wnqueen</td>
<td>32/68/0</td>
<td>32/68/0</td>
<td>32/68/0</td>
<td>32/68/0</td>
</tr>
<tr>
<td>Total</td>
<td>153/198/34</td>
<td>154/205/28</td>
<td>144/165/76</td>
<td>144/205/36</td>
</tr>
</tbody>
</table>

6 Conclusion and Future Work

It is commonly known that, general PB constraints are more expressive than clauses and cardinality constraints. However, the additional pruning power obtained by learning general PB constraints may not compensate the additional effort concerning the propagation procedure [3]. Note that general PB constraints are much harder to propagate than clauses, or even cardinality constraints.

Considering the disparity between the results concerning the application of learning techniques in several state of the art PB solvers [5], the main goal of this work is to help clarifying which is the best learning scheme.

Our results show that cardinality constraint learning is the most effective and robust learning scheme. Moreover, it obtained much better results than the original clause learning scheme both on the small integer non-optimization and optimization benchmarks from PB’07 evaluation. Cardinality constraints are easier to propagate than general PB constraints and are also more expressive than clauses. Therefore, this learning scheme seems a reasonable compromise between general PB learning and clause learning. As such, these results were not unexpected and confirm the results obtained by Chai and Kuehlmann [3] using their PB solver, galena.
References


