Preprocessing in Pseudo-Boolean Optimization: An Experimental Evaluation

Ruben Martins, Inês Lynce, and Vasco Manquinho

IST/INESC-ID, Technical University of Lisbon, Portugal
{ruben,ines,vmm}@sat.inesc-id.pt

Abstract. Preprocessing in Boolean Satisfiability (SAT) is commonly used since it can substantially decrease the run time of SAT solvers. On the other hand, preprocessing is almost non existent in Pseudo-Boolean Optimization (PBO). PBO is closely related to SAT and therefore can benefit from the advances in SAT preprocessing. In this paper, we study the impact of SAT preprocessors, namely `HyPre` and `extHyPre`, in PBO formulas. We explain how these preprocessors can be adapted to be used in PBO instances. Experimental results show that, even though the preprocessing in PBO is not as effective as in SAT, it can lead to improvements.

1 Introduction

Nowadays, it is acknowledged by the Boolean Satisfiability (SAT) community that using preprocessors prior to solving can substantially decrease the run time of SAT solvers. This is due to the development of several preprocessors for which empirical evidence shows its importance in SAT solving.

`HyPre` [3] was one of the first efficient preprocessors. It uses hyper-binary resolution to deduce new binary clauses. Moreover, `HyPre` is able to detect and substitute equivalent literals incrementally. A weaker schema has been adopted by `NiVER` [11], which eliminates variables by resolution as long as this computation does not increase the number of literals in the CNF formula. `NiVER` has been improved later by a substitution rule, together with the use of clause signatures and touched lists to define the recent `SatELite` [5] preprocessor. More recently, `ReVivAl` [8] was proposed, being based on strengthening, or vivifying, the redundant clauses from the original formulas. Also recently, `extHyPre` [1] has been proposed to extend the hyper-resolution used in `HyPre`, so that it can deduce clauses of a bounded length instead of only binary clauses. `ExtHyPre` applies this inference rule to an extension of the graph representation of binary CNF formulas to the general case. It then adds the deduced clauses to the original formula. This preprocessor, however, does not perform equivalent reduction as in `HyPre`.

Pseudo-Boolean (PB) constraints offer a more expressive and natural way to express constrains than clauses and yet, this formalism remains close enough to the SAT problem to benefit from the recent advances in SAT. Indeed, PBO solvers have been the subject of recent research [7, 10, 6], namely in applying techniques already successful in SAT. However, to the best of our knowledge

no preprocessor has been developed in the past to simplify Pseudo-Boolean Optimization (PBO) problem instances. Moreover, the inclusion of preprocessing techniques in PBO solvers is almost non existent.

In this paper, we propose to benefit from the advances made in SAT preprocessing and to study the impact of SAT preprocessors in PBO formulas. For that we have selected `HyPre` and `extHyPre`, since the inference rules used in both are valid even when applied to a subset of a PB formula. In contrast, the inference rules present in `SatELite`, (such as variable subsumption) cannot be consistently applied to a subset of constraints in a PB formula. This paper presents two preprocessors: one based in `HyPre`, where the SAT preprocessor is used as a black-box to preprocess a PBO formula, and the other one, based in `extHyPre`, where the extended hyper-resolution is applied to a graph representation with restricted information about the PBO formula.

The paper is organized as follows. In the next section basic notations and definitions about PBO are provided. In section 3, the SAT preprocessor `HyPre` is described. We will then show how we can use `HyPre` as a black-box to preprocess a PBO formula. Next, the preprocessor `extHyPre` and the extended hyper-resolution inference rule are presented, followed by an explanation on how to produce a restricted graph representation of a PBO formula, in order to apply the preprocessor `extHyPre`. Experimental results will then show that preprocessing can have a positive impact in PBO. Finally, the paper concludes and suggests future work.

2 Pseudo-Boolean Optimization

A pseudo-Boolean (PB) constraint is defined over a set of n Boolean variables, x_1, $x_2, \ldots, x_n$, which can be assigned truth values 0 (false) or 1 (true). A *literal* l_i is either a variable x_i (i.e., a positive literal) or its complement $\overline{x}_i$ (i.e., a negative literal). A positive literal, x_i, evaluates to 1 if and only if the corresponding variable x_i is assigned value 1, a negative literal, $\overline{x}_i$, evaluates to 1 if and only if the corresponding variable x_i is assigned value 0. A PB constraint is defined as a linear inequality over a set of literals of the following normal form:

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

such that for each $i \in \{1, \ldots, n\}, a_i, b \in \mathbb{Z}^+$. It is well-known that any PB constraint (with negative coefficients, equalities or other inequalities) can be converted into the normal form in linear time [4]. In the sequel, it is assumed that all PB constraints are in normal form.

PB constraints can be divided into three categories: *cardinality constraints*, *clauses* and *general PB constraints*. If a PB constraint has all a_i coefficients with the same value k, it is called a *cardinality constraint*, since it requires $\lceil b/k \rceil$ literals to be true for the constraint to be satisfied.

Example 1. An example of a PB cardinality constraint is $x_1 + x_2 + x_3 + \overline{x}_4 \geq 2$. It is satisfied when at least any two literals evaluate to 1, for example when $x_1 = 1$ and $x_4 = 0$.

If all coefficients a_i and the right hand side b are 1, then the PB constraint is a *clause*. Notice that a clause is a particular case of a cardinality constraint. From this point forward, when we refer to cardinality constraints we will only refer to those which are not clauses.

Example 2. An example of a clause is $x_1 + \overline{x}_2 + x_3 \geq 1$. It is satisfied when at least one literal evaluates to 1, for example when $x_2 = 0$.

If a PB constraint is neither a cardinality constraint nor a clause, then it falls into the category of *general PB constraint*.

Example 3. An example of a general PB constraints is $7x_1 + 4\overline{x}_2 + 5x_3 \geq 7$. It is satisfied when $x_1 = 1$ or when $x_2 = 0$ and $x_3 = 1$.

Given a formula φ, which is a conjunction of PB constraints [1], and a linear objective function f, the *Pseudo-Boolean Optimization (PBO) problem* is defined as the problem of finding an assignment to problem variables such that all constraints are satisfied and the value of the objective function f is optimized.

Definition 1 (Pseudo-Boolean Optimization (PBO) Problem). *A general definition of the PBO problem is:*

$$\begin{array}{l} \textit{minimize} : \sum_j c_j x_j, \quad \forall_j, c_j \in \mathbb{Z} \\ \textit{subject to} : \bigwedge_i \sum_j a_{i,j} x_j \geq b_i \end{array}$$

Example 4. An example of a PBO problem instance is:

$$\begin{array}{ll} \text{minimize} & : x_1 - 2x_3 + x_4 \\ \text{subject to} & : x_1 + x_2 + x_3 + \overline{x}_4 \geq 2 \\ & \quad x_1 + \overline{x}_2 + x_3 \geq 1 \\ & \quad 3x_1 + 4\overline{x}_2 + 5x_3 \geq 7 \end{array}$$

It has an optimal solution of -1 when $x_1 = 1$, $x_2 = 1$, $x_3 = 1$ and $x_4 = 0$.

3 Hyper-Binary Resolution

A CNF formula is represented using a finite set of Boolean variables x_i. A *clause* ω is a disjunction of literals and a *CNF formula* φ is a conjunction of clauses.

Example 5. An example of a CNF formula is $\varphi = (\omega_1 \wedge \omega_2 \wedge \omega_3)$ (equivalent to $\varphi = \{\omega_1, \omega_2, \omega_3\}$ using a set notation) where $\omega_1 = (x_1 \vee \overline{x_2})$, $\omega_2 = (x_2 \vee x_3)$ and $\omega_1 = (\overline{x_2} \vee \overline{x_3})$. It is satisfied when, for example, $x_1 = 1, x_2 = 1$ and $x_3 = 0$.

A CNF formula is satisfied if and only if there is an assignment to the variables such that all of its clauses are satisfied. The *SAT problem* is to decide whether there exists a truth assignment to the variables such that the formula becomes satisfied.

[1] The conjunction of cardinality and PB constraints is denoted *pseudo-Boolean part*, whereas the conjunction of clauses is denoted *clausal part*.

Definition 2 (Binary Implication Graph). *Given a CNF formula φ with a set of binary clauses γ, a Binary Implication Graph is a directed graph $G = (V, E)$ such that:*

- $l_i, \overline{l_i} \in V$ *if and only if* l_i *is a literal in* φ,
- $e = (l_i, l_j) \in E[G]$ *if and only if* γ *contains a clause* $(\overline{l_i} \vee l_j)$.

Definition 3 (Unit Propagation (UP)). *The unit clause rule applies whenever a unit clause $\omega_i = (l_j)$ is identified. In this case, all clauses containing literal l_j are declared satisfied, and the literal $\overline{l_j}$ is removed from all clauses containing it. This simplification may originate new unit clauses in which case the unit clause rule should be applied again until no unit clauses remain or the empty clause is derived. The process of iteratively applying the unit clause rule is called unit propagation (UP).*

Unit propagation is widely used in SAT as a simplification rule that can be performed in polynomial time.

Example 6. Consider two clauses $\omega_1 = (x_1 \vee \overline{x_2} \vee x_3)$ and $\omega_2 = (x_2 \vee x_4)$. Given the partial assignment $\{x_1 = 0, x_3 = 0\}$, literal $\overline{x_2}$ in ω_1 must have value 1 and therefore we must assign $x_2 = 0$. Consequently, literal x_4 in clause ω_2 must also have value 1 and therefore we must assign $x_4 = 1$.

Bacchus and Winter proposed `HyPre` [3], a preprocessor for CNF formulas based on hyper-binary resolution and equality reduction. Their experiments show that in most cases there is a benefit in using this preprocessor prior to invoking a state-of-the-art SAT solver. `HyPre` preprocesses a CNF instance in three interconnected ways:

1. Adding binary clauses using hyper binary resolution;
2. Identifying equivalent variables by traversing the binary implication graph and performing substitutions accordingly;
3. Finding unit clauses and propagating the corresponding unassigned literals.

Definition 4 (Resolution [9]). *The resolution rule is defined as follows:*

$$\frac{(l_{i_1} \vee \ldots \vee l_{i_n} \vee x) \quad (l_{j_1} \vee \ldots \vee l_{j_n} \vee \overline{x})}{(l_{i_1} \vee \ldots \vee l_{i_n} \vee l_{j_1} \vee \ldots \vee l_{j_n})}$$

Example 7. Consider the following clauses: $\omega_1 = (x_1 \vee x_2 \vee x_5)$ and $\omega_2 = (x_3 \vee \overline{x_4} \vee \overline{x_5})$. Applying resolution to ω_1 and ω_2 produces the resolvent $(x_1 \vee x_2 \vee x_3 \vee \overline{x_4})$.

Hyper Binary Resolution was proposed by Bacchus and Winter [2] as a hyper-resolution step (i.e., a resolution step that involves more than two input clauses) that attempts to infer new binary clauses.

Definition 5 (Hyper Binary Resolution (HypBinRes)). *The HypBinRes inference rule can be defined as follows:*

$$\frac{(l_1 \vee \ldots \vee l_n) \quad (\overline{l_1} \vee y) \ldots (\overline{l_{n-1}} \vee y)}{(y \vee l_n)} \quad \textit{for } n \geq 2$$

Input: Formula φ
Output: Simplified formula φ
while φ *can be simplified* **do**
  φ = HypBinRes(φ);
  φ = EqReduce(φ);
  φ = UP(φ);
end

Algorithm 1: General idea of the `HyPre` preprocessor

Example 8. Applying HypBinRes to the clauses $(x_1 \vee x_2 \vee x_3 \vee x_4)$, $(\overline{x_1} \vee x_5)$, $(\overline{x_2} \vee x_5)$ and $(\overline{x_3} \vee x_5)$, produces the new binary clause $(x_4 \vee x_5)$.

HypBinRes is used to infer new binary clauses. Once binary clauses are available, equality reduction can be performed. If the formula φ contains two binary clauses $(\overline{x_i} \vee x_j)$ and $(x_i \vee \overline{x_j})$ or $(x_i \vee x_j)$ and $(\overline{x_i} \vee \overline{x_j})$ (i.e., the equivalences $x_i \Leftrightarrow x_j$ or $x_j \Leftrightarrow \overline{x_i}$, respectively), then we can generate a new formula *EqReduce(φ)* by *equality reduction*.

Definition 6 (Equality Reduction). *If two variables are equivalent (i.e., $x_i \Leftrightarrow x_j$) then a new formula EqReduce(φ) is obtained by:*

- *Replacing all occurrences of x_j in φ by x_i or $\overline{x_i}$;*
- *Removing all clauses which now contain both x_i and $\overline{x_i}$;*
- *Removing all duplicate occurrences of x_i (or $\overline{x_i}$) from all clauses.*

We should note that equality reduction may generate new binary clauses, which may lead to identifying additional equivalent variables.

Example 9. Consider the following CNF formula: $\varphi = \{\omega_1, \omega_2, \omega_3, \omega_4, \omega_5\}$ where $\omega_1 = (x_1 \vee \overline{x_2})$, $\omega_2 = (\overline{x_1} \vee x_2)$, $\omega_3 = (x_1 \vee \overline{x_2} \vee x_3)$, $\omega_4 = (x_2 \vee \overline{x_4})$, $\omega_5 = (x_1 \vee x_2 \vee x_4)$. *EqReduce*$(\varphi) = \{(x_1 \vee \overline{x_4}), (x_1 \vee x_4)\}$, since $x_1 \Leftrightarrow x_2$. Clearly EqReduce(φ) has a satisfying truth assignment if and only if φ does. Furthermore, any truth assignment for EqReduce(φ) can be extended to one for φ by assigning x_2 the same value as x_1.

Algorithm 1 summarizes the `HyPre` preprocessor. `HyPre` applies HypBinRes, equality reduction and unit propagation to a CNF formula φ until no more new inferences can be made with these rules.

3.1 HyPre in PBO

PBO formulas can be split into a clausal part and a PB part. `HyPre` can be applied to the clausal part to produce a simplified clausal part. If there are equivalent variables and/or unit clauses, we also have to make the corresponding simplifications in the PB part. The inference rules used by `HyPre` are sound even when applied to a subset of the constraints, since the derived implications remain true for all the constraints.

Figure 1 illustrates how `HyPre` can be applied to a PBO formula. We start by splitting the original PBO formula into two parts: the clausal part (1) and the

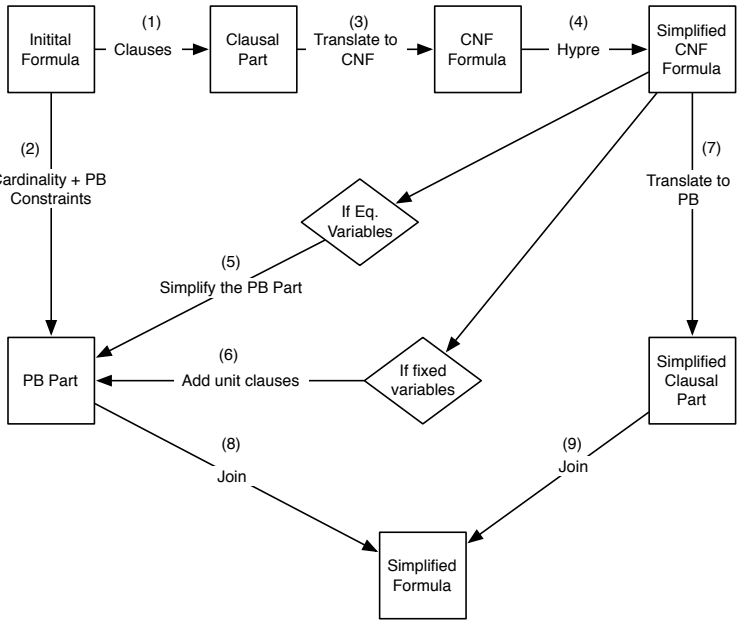

Fig. 1. Applying `HyPre` to a PBO formula.

PB part (2). Afterwards, the clausal part is transformed into a CNF formula (3). `HyPre` is then applied as a black-box to the CNF formula to obtain a simplified CNF formula (4). If there are any equivalent variables, then the pseudo-Boolean part is simplified accordingly (5), taking into account the objective function.

Example 10. Suppose that the PB part of the PBO problem is the following:

$$\begin{aligned} &min : x_1 - x_2 + x_3 \\ &x_1 + \overline{x_2} + 2x_3 \geq 4 \\ &x_1 + x_2 + x_3 + x_4 \geq 2 \end{aligned}$$

and that `HyPre` is applied to the clausal part, such that $x_1 \Leftrightarrow \overline{x_2}$ (i.e., $x_2 = 1 - x_1$) is inferred. After simplifying the PB part the preprocessed formula becomes:

$$\begin{aligned} &min : 2x_1 + x_3 - 1 \\ &2x_1 + 2x_3 \geq 4 \\ &x_3 + x_4 \geq 1 \end{aligned}$$

Notice the introduction of a constant term in the objective function that results from replacing x_2 by $1 - x_1$.

If there are any fixed variables, i.e. variables for which a necessary assignment is identified, then the corresponding unit clauses are added (6). The formula could alternatively be simplified simply propagating the necessary assignments

but since PBO solvers already do that for unit clauses we opted for just adding them to the PB part. Meanwhile, the simplified CNF produced by `HyPre` is translated to pseudo-Boolean constraints (7). Finally, the simplified clausal part is joined with the simplified PB part (8,9) and the final simplified PBO formula is obtained.

4 Extended Hyper-Resolution

Audemard et al. [1] recently proposed a new graph based preprocessing of CNF formulas. The idea behind their graph representation is to extend the binary implication graph for clauses of any size. For that they propose the use of *contexts*.

Definition 7 (Context). *Given a clause ω with at least two literals, i.e. $|\omega| \geq 2$, a context η_ω associated to ω is a conjunction of literals so that $\overline{\eta}_\omega \subset \omega$ and $|\eta_\omega| = |\omega| - 2$. Clearly, an assignment that satisfies η_ω makes the clause w to become a binary clause.*

Example 11. Consider the following clause: $w = (x_1 \vee \overline{x_2} \vee x_3 \vee x_4)$. One possible context associated with ω is $\eta_\omega = (x_2 \wedge \overline{x_3})$. The clause w can be rewritten as $((\overline{x_1} \wedge \eta_\omega) \Rightarrow x_4)$.

The context associated to a binary clause is empty, whereas for ternary clauses the context contains only one literal. A unit clause $\omega = (x_i)$ is represented by the implication $(\overline{x_i} \Rightarrow x_i)$.

Definition 8 (Graph SAT Representation). *Given a CNF formula φ, a graph SAT representation is an edge-labeled directed graph $G = (V, E)$, where each label corresponds to a context, such that:*

- *$l_i, \overline{l_i} \in V$ if and only if l_i is a literal in φ*
- *$e = (\overline{l_i}, l_j) \in E[G]$ with $label(e) = \eta$ if and only if $\exists \omega \in \varphi, \omega = (l_i \vee \overline{\eta} \vee l_j)$ where $\overline{\eta}$ is a sub-clause of ω. For the sake of simplicity, we will often denote e as $(\overline{l_i}, l_j, \eta)$ or $\overline{l_i} \Rightarrow_\eta l_j$.*

For a clause of size k there are $\frac{k(k-1)}{2}$ possible contexts. Assuming that CNF formulas might be very large, it is necessary to use a restricted graph representation for having an efficient preprocessing. With this in mind, it was proposed in [1] a restricted graph representation that imposes an order on the possible contexts.

Definition 9 (Ordered SAT Graph). *Given a CNF formula φ, a ordered SAT graph is an edge-labeled directed graph $G = (V, E)$, where each label corresponds to a context, such that:*

- *$l_i, \overline{l_i} \in V$ if and only if l_i is a literal in φ*
- *For a given clause $\omega = (l_1 \vee l_2 \vee \ldots \vee l_k)$ in φ, the set of edges resulting from ω is $e(\omega) = \bigcup_{1 \leq i \leq k} a_i$ where a_i is defined as follows:*
  - *$(1 \leq i < k) : a_i = (\overline{l_i}, l_{i+1}, \eta(a_i))$ and $\eta(a_i) = \{\overline{l_j} \mid 1 \leq j \leq k$ and $j \neq i$ and $j \neq i + 1\}$*

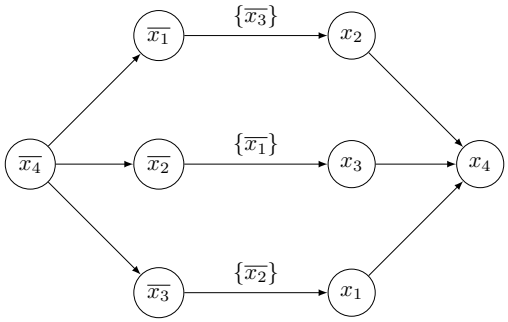

Fig. 2. Ordered SAT graph representation of the CNF formula presented in example 12

- $(i = k) : a_k = (\overline{l_k}, l_1, \eta(a_k))$ *and* $\eta(a_k) = \{\overline{l_2}, \ldots, \overline{l_{k-1}}\}$

Example 12. Consider the following CNF formula $\varphi = \{\omega_1, \omega_2, \omega_3, \omega_4\}$, where $\omega_1 = (x_1 \vee x_2 \vee x_3)$, $\omega_2 = (\overline{x_1} \vee x_4)$, $\omega_3 = (\overline{x_2} \vee x_4)$ and $\omega_3 = (\overline{x_3} \vee x_4)$. The ordered SAT graph representation of φ is presented in figure 2. Note that binary clauses are represented by two edges in both graphs.

Definition 10 (Path). *A path $p(x, y)$ between two vertices x, y in the graph $G = (V, E)$, is defined as follows: $p(x, y) = [l_1, l_2, \ldots, l_k]$ such that $l_1 = x$ and $l_k = y$ and $(l_{i-1}, l_i) \in E[G]$ for $1 < i \leq k$. We define $\eta_p = \bigcup_{1<i\leq k}(l_{i-1}, l_i)$ as the context associated to $p(x, y)$ and $tr(p(x, y)) = tr(\ldots(tr((l_1, l_2), (l_2, l_3))$ $\ldots(l_{k-1}, l_k))\ldots)$ as the transitive closure associated to $p(x, y)$.*

Definition 11 (Fundamental Path). *Let $p(x, y) = [x = l_1, l_2, \ldots, l_k = y]$ be a path between x and y. $p(x, y)$ is called fundamental if it satisfies the following two conditions:*

- *η_p does not contain a literal and its complement;*
- *$\overline{x} \notin \eta_p$ and $y \notin \eta_p$.*

The transitive closure on any path in the graph is used to generate new resolvents. The aim of the preprocessor is then to generate fundamental resolvents of bounded length that can be added to the original formula.

Definition 12 (Extended Hyper Resolution (ExtHypRes)). *The ExtHypRes inference rule can be described as follows:*

$$\frac{(l_1 \vee l_2 \vee \ldots \vee l_n) \quad (\overline{l_1} \vee \alpha_1), \ldots, (\overline{l_n} \vee \alpha_n)}{(\bigcup_{1\leq i\leq n}(\alpha_i))}$$

where α_i for $1 \leq i \leq n$ are sub-clauses.

Input: ordered $G = (V, E)$ and $\omega \in \varphi$
Output: ω_{res} a resolvent clause
$contexts = \emptyset$;
$nodes = \emptyset$;
foreach $l_i \in \omega$ **do**
  if $\exists e = (l_i, l_j, \eta(e)) \in E[G]$ *such that* $l_j \notin contexts$ *and* $\eta(e) \cap nodes = \emptyset$ *and* $\overline{l_j} \notin \omega$ *and* $\overline{\eta(e)} \cap \omega = \emptyset$ **then**
    $contexts = contexts \cup \eta(e)$, $nodes = nodes \cup \{l_j\}$
  else
    if $l_i \notin contexts$ *and* $\overline{l_i} \notin nodes$ **then**
      $nodes = nodes \cup \{l_i\}$
    else
      break
    end
  end
  $\omega_{res} = nodes \cup \{\overline{l_i} : l_i \in contexts\}$
end

Algorithm 2: Extended Hyper-Resolution Preprocessor (`extHyPre`).

Example 13. Consider the following clause $\omega = (x_1 \vee x_2 \vee x_3)$ and a set of binary clauses $\gamma = \{(\overline{x_1} \vee y), (\overline{x_2} \vee y), (\overline{x_3} \vee y)\}$. $ExtHypRes(\omega, \gamma) = y$. In this case, the application of the ExtHypRes rule corresponds exactly to the application of HypBinRes. If we consider the set of clauses $\varphi = \{(\overline{x_1} \vee \overline{\eta}_1 \vee y_1), (\overline{x_2} \vee \overline{\eta}_2 \vee y_2), (\overline{x_3} \vee \overline{\eta}_3 \vee y_3)\}$, then $ExtHypRes(\omega, \varphi) = (\overline{\eta}_1 \vee \overline{\eta}_2 \vee \overline{\eta}_3 \vee y_1 \vee y_2 \vee y_3)$. Clearly, a step of ExtHypRes can be seen as an extension to the HypBinRes rule presented in the previous section.

Algorithm 2 is applied to each clause ω in the formula. It is then iterated using the resolvent generated in the previous step, until either the size of the resolvent exceeds a given limit size or no fundamental resolvent can be generated.

At each step of the algorithm, a new clause ω of a given formula φ is selected and processed. For each $l_i \in \omega$, an edge $e = (l_i, l_j, \eta(e)) \in E[G]$ is chosen so that $p(l_i, l_j)$ is fundamental. The new resolvent is generated following the edges in the ordered SAT representation $G = (V, E)$. Such a resolvent is composed of the accumulated set of literals (nodes) and contexts. This resolvent is obtained using an extension of hyper-resolution. If no fundamental resolvent can be generated the process terminates.

4.1 Extended Hyper-Resolution in PBO

The preprocessor `extHyPre` is applied to a restricted graph representation of a SAT formula. To apply the preprocessor to PBO we should build a restricted graph representation of a PBO formula. As mentioned before, a PBO formula can have three types of constraints: clauses, cardinality and general PB constraints. The representation of clauses is straightforward. Cardinality constraints are encoded using the following graph representation:

Definition 13 (Graph Representation of Cardinality Constraints). *Given a cardinality constraint $\omega = l_1 + \ldots + l_n \geq k$, its corresponding graph repre-*

sentation is an edge-labeled directed graph $G = (V, E)$ with a context associated to each edge such that:

- $l_i, \overline{l_i} \in V$ *if and only if* l_i *is a literal in* ω.
- $e = (\overline{l_i}, l_j, \eta) \in E[G]$ *if and only if* $l_i \in \omega$ *and* $|\omega| - k - 1 \geq 0$ *and* $\eta = \bigcup_k \{\overline{l_k}\}$ *with* $(i + 1 \ mod \ |\omega|) \leq k \leq (|\omega| - k - 1 \ mod \ |\omega|)$ *for each* j *such as* $(i + j + |\omega| - k - 1 \ mod \ |\omega|) \leq j \leq i$.

Example 14. Consider the following cardinality constraint $x_1 + x_2 + x_3 + x_4 \geq 2$. Using the graph representation of a cardinality constraint implies adding the following implications to the PBO graph representation:

$$\overline{x_1} \Rightarrow_{\{\overline{x_2}\}} x_3 \wedge \overline{x_1} \Rightarrow_{\{\overline{x_2}\}} x_4$$
$$\overline{x_2} \Rightarrow_{\{\overline{x_3}\}} x_4 \wedge \overline{x_2} \Rightarrow_{\{\overline{x_3}\}} x_1$$
$$\overline{x_3} \Rightarrow_{\{\overline{x_4}\}} x_1 \wedge \overline{x_3} \Rightarrow_{\{\overline{x_4}\}} x_2$$
$$\overline{x_4} \Rightarrow_{\{\overline{x_1}\}} x_2 \wedge \overline{x_4} \Rightarrow_{\{\overline{x_1}\}} x_3$$

For the general PB constraints something similar to the graph representation of the cardinality constraint could be done but that would lead to an exponential computation. Therefore we propose to do a probing on each variable of the general PB constraints and add to the graph the binary implications found. Note that these implications have empty contexts. Although this captures less information than the other approach, it can be done in an efficient way.

Definition 14 (Graph Representation of General PB Constraints). *Given a general PB constraint* $\omega = a_1 l_1 + \ldots a_n l_n \geq k$ *and* $m = \sum_{i=1}^{n} a_i$, *its corresponding graph representation is an edge-labeled directed* $G = (V, E)$ *such that:*

- $l_i, \overline{l_i} \in V$ *if and only if* l_i *is a literal in* ω.
- $e = (\overline{l_i}, l_j, \{\})$ *if and only if* $l_i \in \omega$ *and one of the following two cases occurs:*
  1. *if* $m - a_i = k$ *then there is an edge for each* j *with* $j \neq i$; *else*
  2. *if* $\exists_j m - a_i - a_j < k$ *then there is an edge between* $\overline{l_i}$ *and* l_j.

Example 15. Consider the following PB constraints: $\omega_1 = 2x_1 + x_2 + x_3 \geq 2$ and $\omega_2 = 2x_1 + 2x_2 + x_3 \geq 2$. If we apply the graph representation to ω_1 the implications $\{(\overline{x_1} \Rightarrow x_2), (\overline{x_1} \Rightarrow x_3)\}$ will be added to the PBO graph representation. If it is applied to ω_2, then the implications $\{(\overline{x_1} \Rightarrow x_2), (\overline{x_2} \Rightarrow x_1)\}$ will be added to the PBO graph representation.

Based on these types of constraints we formulate the following restricted PBO graph:

Definition 15 (Restricted PBO Graph). *Given a PBO formula* φ, *a restricted PBO graph is an edge-labeled directed graph* $G = (V, E)$ *with a context associated to each edge such that:*

- $l_i, \overline{l_i} \in V$ *if and only if* l_i *is a literal in* φ
- *For each constraint* β *the set of edges formed by this constraint are defined according to the type of constraint. For that we use the respective graph representation for clauses, cardinality and general PB constraints described previously.*

ExtHyPre is applied to the restricted PBO graph. With this graph representation of the PBO formula we are able to capture information about the whole formula, and therefore the preprocessor will be acting over an enriched graph which is more powerful than if we would just consider the clausal part.

4.2 Clause Selection Heuristics

ExtHyPre uses two fixed parameters: the maximum size of the resolvent and the percentage of clauses to be considered. The authors proposed the maximum size of the generated resolvent to be 75 and considered only 10% of the clauses for preprocessing, which are selected using a *random* heuristic. In what follows, this approach will be denoted by `extHyPre(75,10,r)`. Note that the 10% are relative to the whole formula and not just the PB constraints corresponding to clauses.

We have performed a study which suggests that the clause selection heuristic can have a relevant impact in the effectiveness of the preprocessor. Therefore, two additional clause selection heuristics are proposed: the *clause* heuristic and the *degree* heuristic.

The *clause heuristic* selects the clauses with larger size to be preprocessed. This approach is denoted by `extHyPre(75,10,c)`. The *degree heuristic* analyzes the graph representation and for each literal and each edge starting at the literal's node, the literal weight is incremented by one plus the number of literals in the edge label. It then runs through the clauses and sums the weight of each of its literals, so that each clause has an associated weight. Afterwards it preprocesses the clauses with the larger weights. This approach is denoted by `extHyPre(75,10,r)`.

Example 16. Consider the graph showed in figure 2. Each literal has the following weights: $x_1 = 1, \overline{x_1} = 2$, $x_2 = 1, \overline{x_2} = 2$, $x_3 = 1, \overline{x_3} = 2$, $x_4 = 0, \overline{x_4} = 3$. Therefore the clauses $\omega_1 = (x_1 \vee x_2 \vee x_3)$, $\omega_2 = (\overline{x_1} \vee x_4)$, $\omega_3 = (\overline{x_2} \vee x_4)$, $\omega_4 = (\overline{x_3} \vee x_4)$ have the following weight (*degree*): $d(\omega_1) = 3$, $d(\omega_2) = 2$, $d(\omega_3) = 2$ and $d(\omega_4) = 2$.

5 Experimental Results

This section evaluates the impact of the two preprocessors proposed in the previous sections. For the experiments reported we used 807 instances from the PB competition of 2007 (available from `http://www.cril.univ-artois.fr/PB07/`) from the category optimization using small integers with linear constraints. The impact of the preprocessors was studied over three state-of-the-art PB solvers: `bsolo 3.1` [7], `Pueblo 1.5` [10] and `minisat+ 1.14` [6]. The results were obtained on an Intel Xeon 5160 server (3.0 GHz with 4GB of RAM) running Red Hat Enterprise Linux WS 4 with a timeout of 1,800 seconds.

5.1 HyPre based preprocessing

This section only considers the benchmarks that HyPre was able to simplify, i.e., for which some preprocessing occurred. This selection has reduced the initial set

Table 1. Average size of the 309 instances to be preprocessed by `HyPre`.

Benchmark	#	Vars	Cons	Constraints		
				Cls	Card	PB
aksoy	79	15,259.90	53,156.70	53,156.57	0.13	0.00
logic-synthesis	74	1,767.91	1,605.35	1,605.35	0.00	0.00
primes-dimacs-cnf	130	1,688.45	11,537.59	11,537.59	0.00	0.00
routing	10	679.20	2,047.00	2,023.00	24.00	0.00
synthesis-ptl-cmos	8	375.75	879.26	879.26	0.00	0.00
haplotype	8	19,926.75	1,189,801.75	1,186,541.00	3,260.75	0.00

Table 2. Average reduction in the clausal part and average time `HyPre` preprocessing.

Benchmark	#	HyPre						
		Original			Preprocessed			
		Vars	Cls	Bin Cls	Vars	Cls	Bin Cls	time
aksoy	79	15,259.90	17,635.05	35,521.52	14,754.35	17,380.80	34,126.67	0.21
logic-synthesis	74	1,767.91	1,400.97	204.38	1,760.27	1,396.66	81.26	0.01
primes-dimacs-cnf	130	1,688.45	1,420.28	10,117.31	1,063.58	1,051.60	9,333.98	0.04
routing	10	679.20	1,623.80	399.20	566.00	1,623.80	176.20	0.01
synthesis-ptl-cmos	8	375.75	629.38	249.88	364.50	620.00	170.25	0.01
haplotype	8	19926.75	1,185,738.00	803.00	19,450.00	1,178,724.25	0.25	1.30

of 807 instances to 309 instances. For the remaining instances, the solving time would be almost the same, since when the preprocessor is unable to simplify the formula it runs in negligible time. Table 1 presents the 309 selected instances, giving the average number of variables (Vars) and constraints (Cons), as well as the number of clauses (Cls), cardinality constraints (Card) and general PB constraints (PB). Observe that the formulas are dominated by their clausal part. In fact, the set of benchmarks *logic-synthesis*, *primes-dimacs-cnf* and *synthesis-ptl-cmos* are composed solely by clauses.

Table 2 presents the average time required by `HyPre` to preprocess each instance for each benchmark as well as the size of the clausal part before and after preprocessing. The table contains the original number of variables, n-ary clauses and binary clauses before and after preprocessing. Clearly, the impact of preprocessing in the clausal part of the PBO formula can be significant. For example, in benchmark *primes-dimacs-cnf* there is a reduction on the number of variables (from 1,688.45 to 1,063.58) and the number of n-ary clauses (from 1,420.28 to 1,051.60). Although in a smaller scale, there is also a reduction on the number of binary clauses (from 10,117.31 to 9,333.98). Finally, observe that the preprocessing time rather small. Even though `HyPre` can be too time consuming when applied to certain instances, it was not the case with the tested benchmarks.

Table 3 presents results for the impact of `HyPre` on the three selected PBO solvers. For each solver we present the number of instances where optimality was proved with and without the used of `HyPre` at preprocessing. The improvements achieved by our approach on the number of optimal solutions found is highlighted in bold, whereas a downgrade in performance is in italic font.

Preprocessing had a positive impact on `Pueblo` and `minisat+`. In the case of `Pueblo`, 4 more instances have been solved. `Pueblo` has gains on *logic-synthesis* and *primes-dimacs-cnf* benchmarks while only losing 1 instance on the *routing*

Table 3. Impact of HyPre preprocessing on bsolo, Pueblo and minisat+.

Benchmark	#	bsolo		Pueblo		minisat+	
		w/o pre	HyPre	w/o pre	HyPre	w/o pre	HyPre
aksoy	79	26	*24*	15	15	24	24
logic-synthesis	74	51	*50*	31	**32**	31	31
primes-dimacs-cnf	130	69	69	75	**79**	79	**83**
routing	10	9	*8*	10	*9*	10	10
synthesis-ptl-cmos	8	6	**7**	1	1	1	1
haplotype	8	0	0	0	0	8	8
Total	309	161	*158*	132	**136**	153	**157**

Table 4. Average size of the 327 instances to be preprocessed by extHyPre.

Benchmark	#	Vars	Cons	Constraints		
				Cls	Card	PB
aksoy	79	15,259.90	53,156.70	53,156.57	0.13	0.00
logic-synthesis	74	1,767.91	1,605.35	1,605.35	0.00	0.00
primes-dimacs-cnf	130	1,688.45	11,537.59	11,537.59	0.00	0.00
radar	12	4,188.25	4,741.58	3,981.75	759.83	0.00
routing	10	679.20	2,047.00	2,023.00	24.00	0.00
synthesis-ptl-cmos	8	375.75	879.26	879.26	0.00	0.00
testset	6	139.50	163.00	147.33	15.67	0.00
ttp	8	1,182.00	16,209.00	8,091.00	414.00	7,704.00

benchmark. Preprocessing was most effective on minisat+, solving more 4 instances of the *primes-dimacs-cnf* benchmark while solving the same number of instances as before of the remaining benchmarks.

5.2 extHyPre based preprocessing

This section evaluates the benchmarks for which extHyPre was able to produce fundamental resolvents with a timeout of 180 seconds and a memory limit of 4GB. For some instances this did not happen due to one of the following reasons: (1) nonexistence of clauses in the formula, (2) no fundamental resolvents could be generated although there are clauses in the formula, (3) the preprocessor times out before fundamental resolvents have been generated. For the instances not considered due to reasons (1) or (2), the required time for preprocessing is usually less than one second. We ended up with 327 instances from the initial set of 807 instances.

Table 4 gives details for the 327 instances to be preprocessed by extHyPre. There is a wide variety of benchmarks to be tested. Although most benchmarks have only clauses, there are some benchmarks that have cardinality constraints as well, like for example *radar* and *ttp*, whereas *ttp* is the only benchmark that has general PB constraints. The benchmark *aksoy* is the most challenging for our preprocessor since the size of some instances forces the preprocessor to terminate due to time or memory exhaustion.

Table 5 presents the average time needed to preprocess each instance of each benchmark set using the different clause selection heuristics described in section 4.2. In general, the preprocessing time is very low. The only exception is for the benchmark *aksoy*, which is larger than the remaining ones. Also, note that the *degree* heuristic is a bit slower than the other ones due to its overhead.

Table 5. Average time in seconds of `extHyPre` preprocessing using the different clause selection heuristics.

Benchmark	#	`extHyPre (75,10,c)`	`extHyPre (75,10,d)`	`extHyPre (75,10,r)`
aksoy	79	78.51	95.83	80.61
logic-synthesis	74	3.48	3.51	3.43
primes-dimacs-cnf	130	0.34	0.39	0.62
radar	12	0.12	0.18	0.12
routing	10	0.13	0.13	0.11
synthesis-ptl-cmos	8	0.04	0.05	0.04
testset	6	0.01	0.01	0.01
ttp	8	4.54	3.08	3.99

Table 6. Impact of `extHyPre` preprocessing on `bsolo` and `Pueblo`.

Benchmark	#	`bsolo`				`Pueblo`			
		w/o pre	`extHypre` (75,10,c)	`extHypre` (75,10,d)	`extHypre` (75,10,r)	w/o pre	`extHypre` (75,10,c)	`extHypre` (75,10,d)	`extHypre` (75,10,r)
aksoy	79	26	26	*25*	*23*	15	15	15	15
logic-synthesis	74	51	51	51	51	31	**32**	**32**	31
primes-dimacs-cnf	130	69	**71**	**70**	**70**	75	**78**	**77**	**78**
radar	12	6	6	6	6	0	0	0	0
routing	10	9	*8*	**10**	9	10	10	10	10
synthesis-ptl-cmos	8	6	6	6	6	1	1	1	1
testset	6	6	6	6	6	6	6	6	6
ttp	8	2	2	2	2	2	2	2	2
Total	327	175	**176**	**176**	*173*	140	**144**	**143**	**143**

Tables 6 and 7 show the impact of `extHyPre` preprocessing on `bsolo, Pueblo` and `minisat+`, respectively. For each variant of the preprocessor and each benchmark set is given the number of instances for which optimality was proved. `bsolo` was able to solve more 1 instance overall using the *clause* and *degree* heuristics. With the *random* heuristic less 2 instances were solved. Overall, preprocessing with `extHyPre` has a minor impact on `bsolo`. On the other hand, preprocessing seems to have a significant effect on `Pueblo` since it is able to solve more 4 instances with the *clause* heuristic and 3 more instances using the other two heuristics. Apart from that, average solving times are more or less the same. Finally for `minisat+`, preprocessing seems to have better results with the *degree* heuristic, solving 3 more instances than before. Notice that the impact of the preprocessor was particularly effective for the *aksoy* benchmark, which shows that the same preprocessing techniques may have a different impact on different solvers.

6 Conclusions and Future Work

Preprocessing in SAT is currently acknowledged as an important step in SAT solving. On the other hand, in PBO there is a lack of preprocessing tools. Due to the close relation between both fields, we have studied two ways of preprocessing PBO formulas that are based on two SAT preprocessors.

Even though preprocessing in PBO is not as effective as in SAT, it can still have a positive impact and lead to better results. Note that proving that a

Table 7. Impact of `extHyPre` preprocessing on `minisat+`.

Benchmark	#	`minisat+`			
		w/o pre	`extHypre` (75,10,c)	`extHypre` (75,10,d)	`extHypre` (75,10,r)
aksoy	79	24	24	**27**	**27**
logic-synthesis	74	31	*30*	*30*	*30*
primes-dimacs-cnf	130	79	**80**	**80**	79
radar	12	0	0	0	0
routing	10	10	10	10	10
synthesis-ptl-cmos	8	1	1	1	1
testset	6	4	4	4	4
ttp	8	2	2	2	2
Total	327	151	151	**154**	**153**

problem has an optimal solution is harder than proving that a problem has a solution, which may explain why preprocessing is not as beneficial to PBO as it is to SAT. However, our results show that preprocessing in PBO can lead to a few improvements, and therefore should be the target of further research work. From our side, we plan to improve the efficiency of our PBO version of `extHyPre` and study different heuristics for clause selection.

Acknowledgments This work is partially supported by Fundação para a Ciência e Tecnologia under research project PTDC/EIA/76572/2006.

References

1. G. Audemard, S. Jabbour, and L. Sais. SAT graph-based representation: A new perspective. *Journal of Algorithms*, 63(1-3):17–33, 2008.
2. F. Bacchus and J. Winter. Enhancing Davis Putnam with extended binary clause reasoning. In *AAAI'02*, pages 613–619, 2002.
3. F. Bacchus and J. Winter. Effective Preprocessing with Hyper-Resolution and Equality Reduction. In *SAT'03*, pages 341–355, 2003.
4. P. Barth. *Logic-Based 0-1 Constraint Programming*. Kluwer Academic Publishers, 1996.
5. N. Eén and A. Biere. Effective Preprocessing in SAT Through Variable and Clause Elimination. In *SAT'05*, pages 61–75, 2005.
6. N. Eén and N. Sörensson. Translating Pseudo-Boolean Constraints into SAT. *JSAT*, 2(1-4):1–26, 2006.
7. V. M. Manquinho and J. Marques-Silva. Effective Lower Bounding Techniques for Pseudo-Boolean Optimization. In *DATE'05*, pages 660–665, 2005.
8. C. Piette, Y. Hamadi, and S. Lakhdar. Vivifying Propositional Clausal Formulae. In *ECAI'08*, pages 525–529, 2008.
9. J. A. Robinson. A Machine-Oriented Logic Based on the Resolution Principle. *Journal of ACM*, 12(1):23–41, 1965.
10. H. M. Sheini and K. A. Sakallah. Pueblo: A Hybrid Pseudo-Boolean SAT Solver. *JSAT*, 2(1-4):165–189, 2006.
11. S. Subbarayan and D. K. Pradhan. NiVER: Non Increasing Variable Elimination Resolution for Preprocessing SAT instances. In *SAT'04*, pages 276–291, 2004.