Construction of an ROBDD for a PB-Constraint in Band Form and Related Techniques for PB-Solvers

Masahiko SAKAI†, Fellow and Hidetomo NABESHIMA††, Member

1. Introduction

A Pseudo-Boolean (PB) problem is the problem which answers the satisfiability of a given instance, which is a conjunction of linear inequality constraints over Boolean variables. Typical approaches to solve PB-constraints employ Integer Linear Programming (restricted to 0-1 variables), DPLL procedures (regarding PB-constraints as generalized clauses [1]), as well as transformations of PB constraints to propositional formulae [1], [2], [3]. Abio et al. have shown that a conversion to a reduced ordered BDD, Pseudo-Boolean constraint, optimization problem

2. Preliminaries

An interval is a set of consecutive integers. An interval $[β, γ] (β, γ ∈ \mathbb{Z} \cup \{−∞, ∞\})$ represents the set $\{i \in \mathbb{Z} | β ≤ i ≤ γ\}$, where any integer $i$ satisfies $−∞ ≤ i$ and $i ≤ ∞$. We use usual notations like $−∞, i$. The summation $1 + j$ of an interval $I$ and an integer $j$ is defined as $\{i + j | i ∈ I\}$.

A valuation $σ$ is a function that assigns 0 or 1 to variables. An application of $σ$ to a propositional formula $f$, and so on, is naturally extended. A valuation $σ$ satisfies a formula $f$ if $σ(f) = 1$. A formula $f$ is satisfiable if there exists a valuation that satisfies $f$; otherwise it is unsatisfiable. Two propositional formulae $f$ and $g$, which may have different variables, are equivalent if $σ(f) = σ(g)$ for any valuation $σ$. They are equisatisfiable if $f$ is satisfiable whenever $g$ is, and vice versa.

For a set $V$ of variables, we say $σ'$ is a $V$-extension of $σ$ if $σ'(x) = σ(x)$ for any $x ∉ V$. We write $σ_{[v−b]}$ for a $(v)$-extension of $σ$ such that $σ_{[v−b]}(x) = b$. We say a propositional formula $f$ is monotonically increasing (resp. monotonically decreasing) with respect to a variable $x$ if $σ_{[v−0]}(f)$ implies $σ_{[v−1]}(f)$ (resp. $σ_{[v−1]}(f)$ implies $σ_{[v−0]}(f)$) for any valuation $σ$. $f$ is monotonically increasing if it is so with respect to all variables. A propositional formula with variables $x_1, ..., x_n$ is regarded as a Boolean function $f(x_1, ..., x_n)$.

A Binary Decision Diagram (BDD) is a rooted, directed and acyclic graph, which consists of decision nodes and terminal nodes $N_0$ and $N_1$. Each node $N$ represents a Boolean function, denoted by $\text{Fun}(N)$. For terminal nodes, $\text{Fun}(N_0) = 0$ and $\text{Fun}(N_1) = 1$. A decision node has two children, and is labelled with a selector variable $x$, which is simply called a selector. We call the child connected with a solid (resp. dotted) edge true-child (resp. false-child). A decision node $N$ represents the Boolean function $(x \land \text{Fun}(N_i)) \lor (\overline{x} \land \text{Fun}(N_{\overline{i}}))$ determined by its true-child $N_i$ and false-child $N_{\overline{i}}$. A variable order is a total order on variables. This paper assumes that BDDs are ordered (OBDDs), i.e., there exists a variable order $< such that for any path the sequence $x_{1}, ..., x_{n}$ of selectors along the path satisfies $x_{1} < ... < x_{n}$. An ordered BDD is reduced (ROBDD) [5] if Boolean functions represented by the nodes are all different. ROBDDs are a canonical representation for Boolean functions under a given variable order.

A Pseudo-Boolean constraint (PB-constraint) is a linear inequality with integer coefficients, where variables have
Let $\sigma(e) = \{ \sigma(e) < k_l \} \cup \{-\infty\}$, $I = \{ \sigma(e) | k_l \leq \sigma(e) \leq k_h \}$, $I_h = \{ \sigma(e) | k_h < \sigma(e) \} \cup \{\infty\}$.

Since $k_l \leq e \leq k_h$ is satisfiable, $I$ is not an empty set. From the definition of consistency, the interval $(L, H)$ is uniquely determined as $[\max(I) + 1, \min(I)], [\max(I), \min(I_h) - 1)$.

Remark that an unsatisfiable PB-constraint may have two consistent pairs of intervals. For example, $5 \leq x + y \leq -5$ has two consistent pairs $((-\infty, \infty), (-\infty, -1])$ and $[3, \infty), (-\infty, -1])$. Also consistent pairs of $5 \leq 0 \leq -5$ are $((-\infty, \infty), (-\infty, -1])$ and $[1, \infty), (-\infty, -1])$

In the rest of this section, we assume that a given PB-constraint is $C : k_l \leq a_1 \ell_1 + \cdots + a_n \ell_n \leq k_h$, and construct an ROBDD, which is equivalent to $C$, under a fixed variable order $x_0 < x_1 < \cdots < x_1$. We use $e_i$ to denote the sub-expression $a_1 \ell_1 + \cdots + a_n \ell_n$.

An important observation in the ROBDD construction is that the pair of intervals for a node is directly calculated from those of its children as shown in the following lemma.

**Lemma 5:** Let pairs $(L_t, H_t)$ and $(L_f, H_f)$ intervals be consistent to $e_i$. If both of intervals $L = (L_t + a_i \ell_t) \cap L_f$ and $H = (H_t + a_i \ell_t) \cap H_f$ are non-empty, $(L, H)$ is consistent to $e_{i+1}$.

**Proof** Assume that $(*)$ in Definition 3 does not hold for $e_{i+1}$ and $(L, H)$. Then, there exists $l, l' \in L, h, h' \in H$, and a valuation $\sigma$ that satisfies $l \leq e_{i+1} \leq h$ but does not satisfy $l' \leq e_{i+1} \leq h'$. In the case that $\sigma(e_{i+1}) = 0$, we have $\sigma(e_{i+1}) = \sigma(a_i + e_i) = \sigma(e_i)$. Thus, $\sigma$ satisfies $l \leq e_i \leq h$, but does not satisfy $l' \leq e_i \leq h'$. Since $l, l' \in L_f$ and $h, h' \in H_f$ from the construction of $L$ and $H$, this is a contradiction to the consistency of $(L_f, H_f)$ to $e_i$. In the case that $\sigma(e_{i+1}) = 1$, we have $\sigma(e_{i+1}) = \sigma(a_i + e_i) = \sigma(e_i)$.

For example, $[1, 1], [1, 2]$ is the unique pair of intervals that is consistent to $l \leq e \leq h \leq 2$.

**Proposition 4:** Let $k_l \leq e \leq k_h$ be a satisfiable PB-constraint. Then its consistent pair $(L, H)$ of intervals is unique.

**Proof** Let $(L, H)$ be a pair of intervals that is consistent to $k_l \leq e \leq k_h$. From the definition, $k_l \in L$ and $k_h \in H$ follow. Let

\[ I_l = \{ \sigma(e) | \sigma(e) < k_l \} \cup \{-\infty\}, \]
\[ I = \{ \sigma(e) | k_l \leq \sigma(e) \leq k_h \}, \]
\[ I_h = \{ \sigma(e) | k_h < \sigma(e) \} \cup \{\infty\}. \]

We use $L \leq e \leq H$ to denote the expected Boolean function. Moreover, for $k_l \in L$ and $k_h \in H$, we say that a pair $(L, H)$ of intervals is consistent to $k_l \leq e \leq k_h$.
that $\gamma = \gamma_f$ and $\gamma = \gamma_i + a_i + 1$.

Consider the former case. Since $\langle L_f, H_f \rangle$ is consistent to $e_i$, there exist $l \in L_f$, $h, h' \in H_f$, and a valuation $\sigma$ that satisfies either $l \leq e_i \leq h$ or $\gamma_f + 1 \leq e_i \leq h'$, exclusively. Let $\sigma'$ satisfies $\sigma'(t_{i+1}) = 0$ and $\sigma'(x) = \sigma(x)$ for any $x \neq t_{i+1}$. Then $\sigma$ satisfies $l \leq e_i \leq h$ if and only if $\sigma'$ satisfies $l \leq e_{i+1} \leq h$ and also $\sigma$ satisfies $\gamma_f + 1 \leq e_i \leq h'$ if and only if $\sigma'$ satisfies $\gamma_f + 1 \leq e_{i+1} \leq h'$. Since $l, \gamma_f + 1 \in L'$ and $h, h' \in H'$, the pair $\langle L', H' \rangle$ is not consistent to $e_{i+1}$. The latter case is similar.

Thanks to Lemma 5, consistent intervals $\langle L, H \rangle$ for a node of a BDD for a PB-constraint can be immediately calculated from its children.

Example 6: An ROBDD for $3 \leq 6x + 5y + 3z \leq 7$ with an order $x < y < z$ is shown in Fig. 2.

The algorithm for the ROBDD construction shown in Fig. 3 works in depth first way with memorizing the interval information, which is a natural extension of the algorithm proposed in Sect. 5 of the Ref. [4] for constraints in standard form. The essential difference from [4] is on memorizing a pair of intervals instead of an interval for each BDD node.

We use a set $D$ each of whose elements is a tuple of a natural number $i$, intervals $L$ and $H$, and a node $N$ of BDD in constructing, where $i$ is used to identify the selector variables $x_i$.

**Definition 7:** We say $D$ is consistent, if $D_0 \subseteq D$ and for every $\langle i, L, H, N \rangle \in D$

(i) $\langle L, H \rangle$ is consistent to $e_i$, and

(ii) $\text{Fun}(N)$ is equivalent to $L \leq e_i \leq H$.

**Lemma 8:** $D_0$ is consistent.

**Proof** Consider the element $(0, (-\infty, -\infty], (-\infty, -1], N_0)$. Since $e_0 = 0$, $l \leq e_0 \leq h$ is not satisfied for any $l \in (-\infty, \infty)$ and $h \in (-\infty, -1]$, which is equivalent to $\text{Fun}(N_0) = 0$. The maximality is trivial. The other elements are similarly shown.

The function CreateBDD eventually goes into Step a) if $i = 0$, because the intervals in $D_0$ cover all pairs of integers. Thus the termination of the algorithm is easily derived.

**Lemma 9:** Let $\langle L, H, N, D' \rangle := \text{CreateBDD}(i, L, h, D)$ for a consistent $D$ and $0 \leq i \leq n$,

(1) $l \in L$, $h \in H$, $\langle i, L, H, N \rangle \in D'$, and

(2) $D'$ is consistent.

**Proof** By induction on $i$. In the case that $i = 0$, Step a) succeeds to find $\langle 0, L, H, N \rangle \in D_0 \subseteq D$. Thus (1) and (2) follow directly from the consistency of $D$.

Consider the case that $i > 0$. If $\langle i, L, H, N \rangle$ such that $l \in L$ and $h \in H$ is found in $D$ at the Step a), then $D = D'$. Thus (1) is trivial and (2) follows directly from the consistency of $D$. Otherwise $\langle L_i, H_i, N_i, D_i \rangle := \text{CreateBDD}(i - 1, l-a_i, h-a_i, D)$ and $\langle L_f, H_f, N_f, D_f \rangle := \text{CreateBDD}(i - 1, l, h, D_i)$ are invoked in the algorithm. Here $D_1$ and $D_2$ are consistent by induction hypothesis (2), and $l-a_i \in L_1$, $h-a_i \in H_1$, $\langle i-1, L_i, H_i, N_i \rangle \in D_1$, $l \in L_f$, $h \in H_f$, and $\langle i-1, L_f, H_f, N_f \rangle \in D_2$ by induction hypothesis (1). Since $L = (L_i+a_i) \cap L_f$, we obtain $l \in L$ and hence of $N_i$ is not empty. Similarly $h \in H$ and $H$ is non-empty.

We show (i) and (ii) in Definition 7 for $\langle i, L, H, N \rangle$. By Lemma 5, (i) follows. Since $l - a_i \leq e_{i-1} \leq h - a_i$ (resp. $l \leq e_{i-1} \leq h$) is equivalent to $\text{Fun}(N_i)$ (resp. $\text{Fun}(N_f)$) from the consistency of $D_1$ (resp. $D_2$). From the construction of $N$ and Lemma 2, we obtain $\text{Fun}(N)$ is equivalent to $l \leq a_i + e_{i-1} \leq h$. Combining this with $l \leq e_i \leq e_{i+1}$, we obtain (ii). Moreover, if $N_i = N_f$ then $\text{Fun}(N_i)$, $\text{Fun}(N_f)$, and $l \leq a_i + e_{i+1} \leq h$ are equivalent.

**Lemma 10:** The algorithm computes an ROBDD, whose root represents the given PB-constraint $k_i \leq a_i \leq k_h$. Since
Definition 11: Given a BDD rooted by PB-constraint in band form.

All nodes in a resulted ordered BDD appears as different elements in the consistent database $D'$ from the following reasons. Nodes $N$ and $N'$ with the same selector variable $x_i$ represent different functions from Proposition 4. Moreover, if $N_i = N_f$ the algorithm do not create a new node. Thus it is reduced.

This algorithm runs in $O(nm \log(m))$ where $m$ is the size of the ROBDD, which is shown in the same way as [4]; the cost of search and insertion on $D$ is $O(\log m)$, and the number of calls of CreateBDD is bounded by $O(mn)$. Note that $n$ is necessary because Step b) does not always create a BDD node.

4. Monotonic CNF-Coding of BDD

In this section, we present a simple way to detect nodes that represent monotonic functions in an ROBDD produced from PB-constraint in band form.

Definition 11: Given a BDD rooted by $R$, we define a CNF-coding, denoted by Cnf($R$), as the conjunction of the following formulae, provided a fresh variable $p_N$ for each node $N$.

(1) “1” for a terminal node $N_1$,

(2) “$\overline{p_N}$“ for a terminal node $N_0$, and

(3) “$p_N \implies \text{if } x \text{ then } p_{N_1} \text{ else } p_{N_f}$” for a decision node $N$ labelled with $x$, where $N_i$ (resp. $N_f$) is the true-child $N_i$ (resp. false-child $N_f$) of $N$.

Note that if-then-else has the meaning as expected. We refer the set of newly introduced variables for the coding of a BDD rooted by $R$ by PVar($R$).

For coding (3) as clauses, three-clause coding is used in MiniSAT+ [2], which produces

(Three) $x \lor p_{N_f} \lor \overline{p_N}$, $\overline{x} \lor p_{N_1} \lor \overline{p_N}$, $p_{N_f} \lor p_{N_1} \lor \overline{p_N}$.

In the Ref. [4], it is proposed two-clause coding, which is more efficient than three-clause one but only applicable to BDDs that represent monotonic functions. Note that PB-constraints in standard form are monotonically decreasing, but not for those in band form. We show that two-clause coding is possible for some nodes of ROBDDs created from PB-constraints in band form.

A node $N$ is monotonically increasing (resp. decreasing) if Fun($N$) is monotonically increasing (resp. decreasing) with respect to its selector variable. Such nodes in an ROBDD constructed in Sect. 3 are easily found from the following proposition.

Proposition 12: Let $N$ be a node with selector variable $x_i$ such that Fun($N$) is equivalent to $L \leq a_i \ell_i + \cdots + a_1 \ell_1 \leq H$ and

- $a_i + \cdots + a_1 \in H$ and $\ell_i = x_i$ (resp. $\ell_i = \overline{x_i}$), or
- $-1 \in L$ and $\ell_i = \overline{x_i}$ (resp. and $\ell_i = x_i$),

then $N$ is monotonically increasing (resp. decreasing).

Proof We consider only the case that $a_i + \cdots + a_1 \in H$ and $\ell_i = x_i$. The formula is equivalent to $l \leq e_i$ for any $l \in L$, where $e_i$ denotes $a_i \ell_i + \cdots + a_1 \ell_1$. Suppose $\sigma_{[x_i=0]}$ satisfies $l \leq e_i$ for the selector variable $x_i$. Since $a_i > 0$, $\sigma_{[x_i=1]}$ satisfies $l \leq e_i$. The other cases are shown similarly. □

We encode the formula in (3) as follows.

(Dec) For monotonically decreasing nodes $N$,

$\overline{x} \lor p_{N_1} \lor \overline{p_N}$, $p_{N_f} \lor \overline{p_N}$.

(Inc) For monotonically increasing nodes $N$,

$x \lor p_{N_f} \lor \overline{p_N}$, $p_{N_1} \lor \overline{p_N}$.

(Three) For the other nodes $N$, three-clause coding (Three).

In the rest of the section, we show the correctness of this partial two-clause coding.

Lemma 13: Fun($N$) is satisfied by a valuation $\sigma$ that satisfies $p_N \land \text{Cnf}($N$)$

Proof By induction on the structure of the BDD.

If $N$ is a terminal node $N_1$, it is trivial because Fun($N_1$) = 1. If $N$ is a terminal node $N_0$, we have no valuation $\sigma$ that satisfies $p_{N_0} \land \overline{p_{N_0}}$. Otherwise, $N$ is a decision node with a selector variable $x$, true-child $N_i$, and false-child $N_f$. From the assumption, $\sigma$ satisfies $p_{N_i}$.• Consider the case that $N$ is encoded by (Three). If $\sigma(x) = 1$, $\sigma(p_{N_f})$ must be 1 from the second clause. Considering the sub-BDD rooted $N_i$, Cnf($N_i$) is included in Cnf($N$). Combining these, $\sigma$ satisfies Fun($N_i$) by induction hypothesis. From the definition of Fun, $\sigma$ satisfies $\text{Cnf}($N$)$, $\sigma$ satisfies $\text{Fun}($N$)$.

If $\sigma(x) = 0$, $\sigma(p_{N_f})$ must be 1 from the first clause. Considering the sub-BDD rooted $N_f$, Cnf($N_f$) is included in Cnf($N$). Combining these, $\sigma$ satisfies $\text{Fun}($N_f$)$ by induction hypothesis. From the definition of Fun, $\sigma$ satisfies $\text{Cnf}($N$)$.

- Consider the case that $N$ is encoded by (Dec). If $\sigma(x) = 1$, $\sigma(p_{N_f})$ must be 1 from the first clause. Considering the sub-BDD rooted $N_i$, Cnf($N_i$) is included in Cnf($N$). Combining these, $\sigma$ satisfies $\text{Fun}($N_i$)$ by induction hypothesis. From the definition of Fun, $\sigma$ satisfies Fun($N_i$).

If $\sigma(x) = 0$, $\sigma(p_{N_f})$ must be 1 from the second clause. Considering the sub-BDD rooted $N_f$, Cnf($N_f$) is included in Cnf($N$). Combining these, $\sigma$ satisfies $\text{Fun}($N_f$)$ by induction hypothesis. From the definition of Fun, $\sigma$ satisfies $\text{Cnf}($N$)$.

- The case encoded by (Inc) is similar to (Dec). □

For proving the reverse, we introduce a notion of active nodes to construct an extended valuation.
Definition 14: For a given valuation \( \sigma \) and a BDD rooted by \( R \), the set \( A_\sigma \) of active nodes is defined as a minimal set satisfying the following conditions:

- \( R \in A_\sigma \).
- \( N_f \in A_\sigma \) for the true-child \( N_f \) of an \( N \in A_\sigma \), if \( \sigma(x) = 1 \) or \( N \) is monotonically increasing where \( x \) is the selector variable of \( N \).
- \( N_f \in A_\sigma \) for the false-child \( N_f \) of an \( N \in A_\sigma \), if \( \sigma(x) = 0 \) or \( N \) is monotonically decreasing where \( x \) is the selector variable of \( N \).

If \( \sigma \) satisfies Fun(\( R \)) for a BDD rooted by \( R \), terminal node \( N_0 \) is not active from the following lemma.

Lemma 15: For a BDD rooted by \( R \), \( \sigma \) satisfies Fun(\( N \)) for any active nodes \( N \) if \( \sigma \) satisfies Fun(\( R \)).

Proof By induction on the definition of the active nodes. In the case that \( N = R \in A_\sigma \), it is trivial.

Consider the case that \( N_f \in A_\sigma \), \( \sigma(x) = 1 \), and the selector variable of \( N \) is \( x \). From the definition of Fun(\( N \)), \( \sigma \) satisfies Fun(\( N_f \)).

Consider the case that \( N_f \in A_\sigma \), which is the true-child of a monotonically increasing node \( N \in A_\sigma \) with selector variable \( x \). We can assume \( \sigma(x) = 0 \), otherwise it is subsumed by the previous case. From the definition of Fun(\( N \)), \( \sigma \) satisfies Fun(\( N_f \)). Since \( N \) is monotonically increasing, \( \sigma|_{\{x=1\}} \) also satisfies Fun(\( N \)). Thus \( \sigma|_{\{x=1\}} \) satisfies Fun(\( N_f \)). Since Fun(\( N_f \)) does not contain \( x \), \( \sigma \) satisfies Fun(\( N_f \)). We can prove for the last case similarly.

Lemma 16: For a BDD rooted by \( R \), let \( \sigma \) be a valuation that satisfies Fun(\( R \)). Let \( \sigma' \) be a \( \text{PVar}(R) \)-extension of \( \sigma \) such that \( \sigma'(p_N) = 1 \) for active nodes \( N \) and \( \sigma'(p_N) = 0 \) for the other nodes \( N \). Then \( \sigma' \) satisfies \( p_R \land \text{Cnf}(R) \).

Proof Since \( R \) is active, \( \sigma'(p_R) = 1 \).

If \( N \) is a terminal node \( N_1 \), then the clause is 1. If \( N \) is a terminal node \( N_0 \), then the clause is \( \overline{p_R} \). Since \( N_0 \) is not active by Lemma 15, \( \sigma'(p_N_0) = 0 \).

If \( N \) is a decision node with a selector variable \( x \), a true-child \( N_f \), and a false-child \( N_f \).

- Consider the case that \( N \) is active and encoded by (Three). In the case that \( \sigma(x) = 1 \), \( N_f \) is also active. Thus, \( \sigma'(p_N) = 1 \). In the case that \( \sigma(x) = 0 \), \( N_f \) is also active. Thus \( \sigma'(p_N) = 1 \).
- Consider the case that \( N \) is active and encoded by (Dec). Since \( N_f \) is active, \( \sigma'(p_N_f) = 1 \). If \( \sigma(x) = 1 \), \( N_f \) is also active and hence \( \sigma'(p_N_f) = 1 \). In either of the cases, \( \sigma' \) satisfies the clauses (Dec).
- The case that \( N \) is active and encoded by (Inc) is similar to (Dec).
- If \( N \) is not active, then \( \sigma'(p_N) = 0 \). Thus, \( \sigma' \) satisfies all the clauses (Three), (Dec) and (Inc).

From Lemmas 13 and 16, the following theorem is shown.

Theorem 17: For a BDD rooted by \( R \), \( p_R \land \text{Cnf}(R) \) and Fun(\( R \)) is equisatisfiable.

5. Binary/Alternative Searches for Optimization

The PB-optimization answers the minimal value of a given goal expression among valuations that satisfy the PB-constraints, where goal expressions are in forms of \( b_1 \ell_1 + \cdots + b_n \ell_n \) for integers \( b_1, \ldots, b_n \).

Most of PB-solvers such as sat4j and MiniSat+ employ the sequential search, in which the satisfiability is firstly checked without considering the goal expression \( g \). Once a satisfiable valuation is obtained, the best known goal value \( k \) is calculated. By augmenting clauses that represents \( g < k \), invoke SAT-solver. Repeating this until resulting unsatisfiable, the best goal value is fixed as the last \( k \). This can be implemented without restarting SAT-solvers, because recent solvers allow incremental solving; adding clauses incrementally, and continuing solving.

This sequential search is not so bad, because in most cases an improvement of the known goal value is more than one. However, a lot of SAT-solver executions are sometimes necessary. Thus it is natural to consider binary search strategies.

This section describes on binary search strategies for a goal minimization. Avoiding a SAT-solver restart, we use a SAT-solver function that solves under tentatively assuming the given literal instead of adding it. Let \( k \) be the best known goal value, and \( l \) be the greatest known lower bound, which is initially the sum of negative coefficients \( b_i \)'s. By introducing a fresh variable \( p \), we add a constraint for \( p \implies g < \lfloor (k - l)/2 \rfloor \), and solve them under assumption that \( p = 1 \). If it is satisfiable, the known best value is renewed from the obtained valuation and proceeds. Otherwise the greatest known lower bound is renewed to \( \lfloor (k - l)/2 \rfloor \) and proceeds.

Binary search strategies sometimes take unnecessary executions. For example, we suppose a goal \( g = 1000x + 4000y \) and its best goal value is 1000. By the sequential search, after we got 1000 it is enough one more SAT-solver execution with \( g < 1000 \). By the above binary search strategy, however, considerable times of SAT-solving are necessary with \( g < 500, g < 750 \), and so on. For this, a pre-process of goal expressions is effective; dividing coefficients by their greatest common divisor (GCD). The example is transformed to \( x + 4y \).

The simplification by GCD takes no effect for the goal \( g' = x + 1000y \) if the best goal value is 1000. For this, we propose an alternating strategy that adopts the sequential (resp. binary) search after UNSAT (resp. SAT) answer obtained. Suppose the best known goal value is 1000 for the goal, two-time executions with \( g' < 500 \) (due to binary one) and \( g' < 1000 \) (due to sequential one) are enough.

6. Evaluation

We implemented our findings into our tool, named GPW,
which is constructed based on Minisat+ [2] version 1.0. The major extensions are summarized as follows:

- Minisat+ has a function to generate clauses via BDDs constructed from each PB-constraint in band form. Thus we implemented the construction of ROBDDs.
- Minisat+ employs the three-clause coding. Thus we implemented two-clause coding for nodes having monotonic property.
- Minisat+ has the incremental SAT-solving for the sequential search in the goal minimization. We implemented the incremental SAT-solving for the binary search and the alternative search.

We performed experiments on a machine equipped with dual Xeon W5590 (3.33GHz, 4core 8thread, L2cache4*256KB, and L3cache 8MB) processors and 48GB memory. MiniSat version 1.13 is used as an underlying SAT-solver, which is included in Minisat+, as an underlying solver. The benchmarks are 1683 instances in total; 452, 532, and 699 instances used in DEC-SMALLINT-LIN, OPT-BIGINT-LIN, and OPT-SMALLINT-LIN divisions of Pseudo-Boolean Competition 2010, respectively. The GPW detects at least one band form in 26, 417, and 100 instances for the respective divisions.

Table 1 shows the number of instances that different methods could solve within 1000 seconds timeout. The rows correspond to coding methods in GPW where 2cl/3cl present two-clause/three-clause coding, std/band present BDD construction from standard-form/band-form, seq/bin/alt present sequential/binary/alternative strategies, and MiniSat+ mode adopts three clause, band form, sequential strategy, and (non-RO) BDD construction. The first seven columns correspond to the divisions and total numbers, where Sat/UnSat/Opt present solved instances with satisfiable/unsatisfiable/optimized results. The rest of columns correspond to the total numbers of constraints processed as standard/band form, BDD nodes (coded by 2-clause, and by 3-clause), and Sat/UnSat solver calls for solved instances.

Figure 4 shows cactus plots of the results, which indicate the number of solved instances within the time. In the figures, lines located in lower and more right side show better results. Note that the “best” solver is a virtual one which runs all solvers by the different methods in parallel and takes the best result.
Generally, processing in standard form with the two-clause coding and with the binary or alternative search cores the best. Let’s see more detailed analysis.

The binary search results better than the sequential search, and the alternative search seems the best. This is reasoned from the following facts:

- The sequential search requires more solver-calls than others,
- UNSAT-calls are generally much heavier than SAT-calls, and
- once obtained the minimal goal value, the sequential (resp. binary, alternative) search requires one (resp. many, two) UNSAT-calls.

It is easily shown that the number of solver-calls with the alternative search is not more than double of those with the binary search.

The effect of the band form varies according to coding/search methods and divisions of the instances. Generally, the band form has advantages of smaller memory use and shorter BDD-construction time than the standard form. The band form has an advantage of the number of solved instances in the following cases:

- Instances are processed by the three-clause coding,
- Instances are solved by the sequential search strategy.
- Instances in OPT-SMALL-LIN division are processed.

The band form has a disadvantage for instances in OPT-BIG-LIN division processed by the two-clause coding.

From the other views, ROBDD construction takes effect, because processing 3cl/band/seq with ROBDD scores better than MiniSat+ mode (3cl/band/seq with non-ROBDD). Two-clause coding method takes effect together with standard form, but also with band form, if it is combined with the alternative or binary strategy.

7. Concluding Remarks

This paper proposed the following methods:

- an efficient ROBDD construction algorithm for PB-constraints in band form by modifying the algorithm in the Ref. [4] designed for standard form,
- a partial two-clause coding of BDD, which produces two clauses for a monotonic node and three clauses for a non-monotonic node, and
- a binary search and an alternative search for goal minimization, which allow incremental SAT-solving.

It appears that the best combination of methods is the alternative (or binary search) strategy and the ROBDD construction from the standard form with the two-clause coding. The construction from the band form is effective under the three-clause coding, or the sequential strategy.

We can choose the band or standard form in ROBDD construction for each constraint in an instance. Thus it is interesting to find good choice strategies.

Acknowledgments

We thank Harald Zankl for discussing on band form. Partially supported by the Austrian Science Fund (FWF) project 1963 and the Japan Society for the Promotion of Science.

References

[1] C. Herde, “Extending DPLL for Pseudo-Boolean constraints,” in Efficient Solving of Large Arithmetic Constraint Systems with Complex Boolean Structure, pp.37–58, Vieweg+Teubner, 2011.
[2] N. Eén and N. Sörensson, “Translating Pseudo-Boolean constraints into SAT,” J. Satisfiability, Boolean Modeling and Computation (JSAT), vol.2, no.1-4, pp.1–26, 2006.
[3] O. Bailleux, Y. Boufkhad, and O. Roussel, “A translation of Pseudo Boolean constraints to SAT,” J. satisfiability, Boolean modeling and computation (JSAT), vol.2, no.1-4, pp.191–200, 2006.
[4] I. Abío, R. Nieuwenhuis, A. Oliveras, E. Rodríguez-Carbonell, and V. Mayer-Eichberger, “A new look at BDDs for Pseudo-Boolean constraints,” J. Artificial Intelligence Research (JAIR), vol.45, pp.443–480, 2012.
[5] R.E. Bryant, “Graph-based algorithms for boolean function manipulation,” IEEE Trans. Comput., vol.35, no.8, pp.677–691, Aug. 1986.