# Revisiting Pseudo-Boolean Encodings from an Integer Perspective

Hendrik Bierlee[1,2,3][0000−0001−6766−5435], Jip J. Dekker[2,3][0000−0002−0053−6724], and Peter J. Stuckey[2,3][0000−0003−2186−0459]

[1] DTAI, KU Leuven, Belgium henk.bierlee@kuleuven.be
[2] Monash University, Melbourne, Australia
{jip.dekker,peter.stuckey}@monash.edu
[3] OPTIMA ITTC, Melbourne, Australia

**Abstract.** Traditionally, SAT encodings of complex constraints, such as linear constraints or more specifically PB constraints, are specified in terms of Boolean variables and clauses. However, often sets of related Boolean variables are either encodings of integer variables, or act as if they were. Furthermore, any encoding of linear constraints has to encode partial sums, and these are integers (even if the encoding does not explicitly notice this). By formally specifying the SAT encoding using integer variables and constraints, coupled with a procedure to encode this specification into SAT, we can gain some more insight into the encoding methods, and compose new ones. Experiments using these integer-driven encodings show that they can improve on standard approaches to encoding PB and integer linear constraints to SAT.

**Keywords:** Boolean Satisfaction · Pseudo-Boolean Equations · Encoding.

## 1 Introduction

*Boolean Satisfiability* (SAT) is a powerful approach to solving combinatorial problems, but to use a SAT solver, we need to encode the problem into clauses. A common and important class of constraint to encode are linear constraints over Boolean literals, or *Pseudo-Boolean* (PB) constraints. In its normalized form, a PB constraint is $\sum_{i=1}^{n} q_i b_i \leq k$, where $q_i$ and $k$ are positive integer constants and $b_i$ are Boolean literals.There are many competing methods used to encode a PB constraint, including the *Generalized Totalizer* (GT) [22], *Sequential Weight Counter* (SWC) [20], *Binary Decision Diagram* (BDD) [17], and the *Ripple Carry Adder* (RCA) encoding [26].

Each of these methods implicitly generates auxiliary encodings of some partial sums $\sum_{i \in J} q_i b_i$ where $J \subseteq \{1, \ldots, n\}$. For example, SWCs and BDDs encode a linear *decomposition* of the sum, such that: $y_j \geq \sum_{i=1}^{j} q_i b_i, 2 \leq j \leq n$, using encodings of $q_1 b_1 + q_2 b_2 \leq y_2$, and $y_j + q_{j+1} b_{j+1} \leq y_{j+1}, 2 \leq j < n$, finally adding $y_n \leq k$. Note that it is sufficient to enforce that $y_j$ is greater

than or equal to the partial sum, rather than equal to, since the overall inequality constraint is enforced by $y_n \leq k$. A binary tree decomposition of the sum, used by GTs, encodes the partial sums (assuming $n = 2^k$ for simplicity) as $y_i^1 \geq q_{2i-1}b_{2i-1} + q_{2i}b_{2i}, 1 \leq i \leq 2^{k-1}$ and $y_i^{j+1} \geq y_{2i-1}^j + y_{2i}^j, 1 \leq i \leq 2^{k-j}$, finally adding $y^k \leq k$. RCA encodings are free to decompose the sum in either way, and use binary encodings of partial sums.

In the original specifications of the encoders, apart from RCA, each set of Boolean variables which implicitly represents a partial sum is not explicitly treated as an integer variable $y$. In this paper, we show how to improve these encodings of PB constraints by recognising that all of these methods have one thing in common: they construct a **decomposition of addition constraints over *integers* representing the (lower bound of the) partial sums**. With this viewpoint, several advantages become evident. Importantly, any improvement on the integer decomposition automatically carries over to each of the PB encoders. The contributions of this work are:

- We show how all existing encodings can be generated from a simple integer viewpoint (Section 3). This allows their correctness and propagation consistency to be proved uniformly (Section 4.1).
- Existing PB encoding methods are automatically generalized to support integer linear constraints and side-constraints (Section 4.2).
- We show how to encode PB equations, $\sum_{i=1}^{n} q_i b_i = k$, more efficiently than by decomposing into two inequalities (Section 4.3).
- We show how different encodings can be applied to each individual decision variable *and* the partial sums (Section 4.4).

The remainder of the paper is organized as follows. In Section 2, we define the concepts used within the remainder of this paper. In Section 3, we introduce a simple encoding for the ternary inequality constraint over integers, which can recreate the GT, SWC, BDD and RCA encodings. In Section 4, we take advantage of the integer viewpoint by proposing multiple extensions. In Section 5, we evaluate the effect of these extensions on solver performance. We conclude our findings in Section 6. Note that related work is discussed throughout the paper.

## 2 Preliminaries

### 2.1 Constraint Programming

A *Constraint Satisfaction Problem* (CSP) instance, $P = (\mathcal{X}, D, \mathcal{C})$, consists of a set of variables $\mathcal{X}$, with each $x \in \mathcal{X}$ restricted to taking values from some initial domain $D(x)$. For this paper, we assume domains are ordered sets of integers, and denote by $\text{lb}_D(x)$ and $\text{ub}_D(x)$ the least and greatest values in $D(x)$. We will use interval notation $[l, u]$ to represent the set of integers $\{l, l+1, \ldots, u\}$. A set of constraints $\mathcal{C}$ expresses relationships between the variables. An *assignment* of a CSP instance is a mapping of variables to values, which (if consistent with the domains and constraints) is a *solution* to the instance.

We say a domain $D$ is *domain consistent* for constraint $c$ over variables $V$ if for all $v \in V$ and all $d \in D(v)$ there exist $d_{v'} \in D(v')$ for all $v' \in V \setminus \{v\}$, such that $(v = d) \wedge \bigwedge_{v' \in V \setminus \{v\}} (v' = d_{v'})$ is a solution of $c$. In other words, every value in every domain of $v \in V$ participates in a solution for $c$ in $D$.

We say a domain $D$ is *bounds*($\mathbb{R}$) *consistent* for constraint $c$ over variables $V$ if for all $v \in V$ and for $d \in \{\mathrm{lb}_D(v), \mathrm{ub}_D(v)\}$ there exist **real** numbers $d_{v'}$, where $\mathrm{lb}_D(v') \leq d_{v'} \wedge d_{v'} \leq \mathrm{ub}_D(v')$ for all $v' \in V \setminus \{v\}$ such that $v = d \wedge \bigwedge_{v' \in V \setminus \{v\}} v' = d_{v'}$ is a (real) solution of $c$. That is to say, every lower and upper bound for $v \in V$ participates in a real solution for $c$ within the bounds of $D$.

### 2.2 Satisfiability

A SAT problem can be considered a special case of a CSP, where the domain for all variables $x$ is $D(x) \in \{0, 1\}$, representing the values *false* and *true* . A *literal* is either a Boolean variable $x$ or its negation $\neg x$. We extend the negation operation to operate on literals, i.e. $\neg b = \neg x$ if $b = x$ and $\neg b = x$ if $b = \neg x$. We use the notation $b = v$ where $b$ is a literal and $v \in \{0, 1\}$ to encode the appropriate form of the literal, i.e. if $v = 1$ it is equivalent to $b$ and if $v = 0$ it is equivalent to $\neg b$. The notation $b \neq v$ is defined similarly to encode $\neg(b = v)$. A *clause* is a disjunction of literals. In a SAT problem $P$, the constraints $\mathcal{C}$ are clauses. A partial assignment $\theta$ maps each Boolean literal $b$ to either true $\theta(b) = \{1\}$ or false $\theta(b) = \{0\}$, or unknown $\theta(b) = \{0, 1\}$.

### 2.3 Encoding CP to SAT

Given an integer $x$ with (possibly non-contiguous) domain $D(x) = \{d_1, d_2, \ldots, d_m\}$, a variable encoding method maps $x$ to a set of Boolean *encoding* variables. We use semantic brackets to name the Booleans, where the Boolean $[\![f]\!]$ is true iff the formula $f$ holds. Two Booleans with different names $[\![f]\!]$ and $[\![g]\!]$ refer to the same underlying literal iff $[\![f]\!] \equiv [\![g]\!]$.

The *interpretation* of the assignment of a variable encoding returns the assignment of the original encoded integer. This can require a variable encoding to be associated with a *consistency constraint* to ensure that the interpretation correctly maps to a single integer.

The *order* encoding of $x$, denoted $x{:}\mathbb{O}$, introduces $m - 1$ encoding variables $[\![x \geq v]\!], v \in \{d_2, \ldots, d_m\}$. $[\![x \geq v]\!]$ is true iff $x$ is assigned a value greater or equal to $v$. We extend our semantic brackets notation of the order encoding to express other relations on $x$ in terms of these encoding variables:

$$[\![x \geq v]\!] \equiv 1, v \leq d_1 \tag{1a}$$

$$[\![x \geq v]\!] \equiv [\![x \geq d_{i+1}]\!], d_i < v \leq d_{i+1} \tag{1b}$$

$$[\![x \geq v]\!] \equiv 0, v > d_m \tag{1c}$$

$$[\![x > v]\!] \equiv [\![x \geq v + 1]\!] \tag{1d}$$

$$[\![x < v]\!] \equiv \neg[\![x \geq v]\!], [\![x \leq v]\!] \equiv \neg[\![x \geq v + 1]\!] \tag{1e}$$

We can map a partial assignment $\theta$ on the Booleans encoding integer variable $x$ to any value $a$ where $arg\,max_{d \in D(x)}(\theta([\![x \geq d]\!])) \leq a < arg\,min_{d \in D(x) \bigcup \{d_m+1\}}(\theta([\![x < d]\!])$. No value for $a$ may exist, unless the following *Implication Chain* (IC) constraint is enforced:

$$\bigwedge_{i=3}^{m} [\![x \geq d_i]\!] \rightarrow [\![x \geq d_{i-1}]\!] \tag{2}$$

Other important integer variable encodings are the direct, the binary and mixed radix encodings [9]. The *binary* encoding of $x$, denoted $x{:}\mathbb{B}$, introduces $n$ encoding variables $[\![\text{bit}(x,k)]\!], k \in 0..n-1$. Then, $[\![\text{bit}(x,k)]\!]$ is true iff the $k$-th most significant bit in the two's complement binary representation of the value assigned to $x$ is true, i.e, in C notation: $[\![\text{bit}(x,k)]\!] =$`(x >> k) & 1`. For simplicity, we assume $x$ is known to be non-negative, extensions to two's complement representations are well understood. The number of bits required $n = \lceil log(d_m+1) \rceil)$ is given by the upper bound of $x$. Let $\mathcal{B}(x) = 0..n-1$ and let $\mathcal{R}(x)$ be the set of representable integers for the binary encoding of $x$, $\mathcal{R}(x) = 0..2^n-1$.

The *consistency constraint* for the binary encoding enforces that the Boolean representation can only represent values in the initial domain $D_i(x)$. It is the SAT encoding of the constraints (a) $x \geq d_1$ if $d_1 > 0$, (b) $x \leq d_m$ if $d_m < 2^{n-1}$ (c) $x \neq d$ for each $d_1 < d < d_m, d \notin D_i(x)$. Constraints (a) and (b) are encoded using lexicographic decompositions, while (c) can be simply encoded as a clause $\vee_{k \in \mathcal{B}(x)} [\![\text{bit}(x,k)]\!] \neq \text{bit}(d,k)$. For highly sparse domains, more efficient approaches are possible for (c).

### 2.4 Pseudo-Boolean Constraints

A PB constraint has the form $\sum_{i=1}^{n} q_i b_i \ \# \ k$ where $q_i, k$ are integer constants, $b_i$ are Boolean literals and $\# \in \{<, \leq, =, \geq, >\}$. A *normalized* PB constraint requires that $k$ is positive, $1 \leq q_i \leq k, \forall 1 \leq i \leq n$ and the comparator $\#$ is $\leq$. Since a PB constraint can always be normalized to one or two normalized PB inequalities, we only consider normalized PB constraints, until Section 4.3.

## 3 PB encodings as integer-based decompositions

All encodings for the PB constraint $\sum_{i=1}^{n} q_i b_i \leq k$ decompose the constraint into ternary constraints that implicitly or explicitly encode integer partial sums as auxiliary integer variables. The difference between the encodings is the shape of the decomposition, the domains of the auxiliary variables, and their choice of variable encoding (order for GT, SWC and BDD, binary for RCA).

The first step of the integer viewpoint is to introduce *principal* integer variables $x_i$ with domains $D(x_i) = \{0, q_i\}$ to represent the PB terms $x_i = q_i b_i$. This allows us to represent the PB constraint as the integer linear constraint, $\sum_{i=1}^{n} x_i \leq k$. Next, we create an arbitrary binary tree with leaf nodes labelled $x_i$ that sums up the $n$ original terms using $n-1$ internal nodes. Each node would

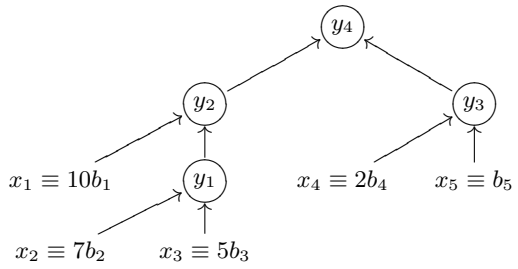


**Fig. 1.** A possible tree decomposition of $10b_1 + 7b_2 + 5b_3 + 2b_4 + b_5 \leq 15$

be labelled by a new auxiliary variable $y_i, 1 \leq i \leq n-1$ that encodes the partial sum of the tree rooted at the internal node $i$. The final variable $y_{n-1}$ is also denoted as $y_{root}$.

Each node $i$ creates a constraint $l(i) + r(i) = y_i$, $l(i)$ is $y_l$ if the left child of node $i$ is an internal node $l$ and $x_l$ if the left child of node $i$ is the leaf node $x_l$. $r(i)$ is defined similarly. Note that we can weaken any of these ternary equalities to an inequality while still encoding the PB constraint. Hence, we can encode $l(i) + r(i) \leq y_i$ for any auxiliary $y_i$, since the PB constraint will hold if and only if all inequalities are satisfied. The overall PB constraint is then enforced by adding a constraint $y_{root} \leq k$ or equivalently $y_{root} = k$ if each path from root to leaf has a weakening inequality.

The introduced variables can be restricted to domain $[0, k]$ since there is no solution where any of them takes a value above $k$. To recreate different encodings, we may restrict the domains further. For example, in the GT encoding for a non-root internal node $i$, we can define $D(y_i) = \{u + v \mid u \in D(l(i)), v \in D(r(i)), u + v \leq k\}$. For the root variable $D(y_{root}) = \{k\}$.

*Example 1.* The tree decomposition shown in Fig. 1 decomposes the PB constraint as $x_2 + x_3 = y_1$, $x_1 + y_1 = y_2$, $x_4 + x_5 = y_3$, $y_2 + y_3 = y_4$, and $y_4 \leq 15$. The GT decomposition makes further changes: it weakens the equalities to inequalities, omits the final constraint, and has domains $D(y_1) = \{0, 5, 7, 12\}$, $D(y_2) = \{0, 5, 7, 10, 12, 15\}$, $D(y_3) = \{0, 1, 2, 3\}$, and $D(y_4) = \{15\}$. □

Once a tree decomposition has been constructed, the encoding of the PB constraint can be reduced to the repeated encoding of *ternary (in)equality* constraints. The RCA encoding of $x + y = z$ assumes the variables $x$, $y$ and $z$ are binary encoded, then encodes the following simple constraints:

$$\begin{aligned} c_1 &\equiv [\![\text{bit}(x,k)]\!] \wedge [\![\text{bit}(y,k)]\!] \\ c_{k+1} &\equiv ([\![\text{bit}(x,k)]\!] + [\![\text{bit}(y,k)]\!] + c_k \geq 2), \qquad k \in 1..n-1 \\ [\![\text{bit}(z,0)]\!] &\equiv [\![\text{bit}(x,k)]\!] \oplus [\![\text{bit}(y,k)]\!] \\ [\![\text{bit}(z,k)]\!] &\equiv [\![\text{bit}(x,k)]\!] \oplus [\![\text{bit}(y,k)]\!] \oplus c_{k-1}, \qquad k \in 1..n-1 \end{aligned} \tag{3}$$

Here, $c_k$ is the carry bit from the $k^{th}$ addition, and $\oplus$ is exclusive-or.

The following equation defines the ternary inequality constraint encoding, for $x + y \leq z$ assuming each of $x$, $y$ and $z$ is an order encoded integer variables with appropriate domains:

$$\bigwedge_{v \in D(x)} \bigwedge_{w \in D(y)} ([\![x \geq v]\!] \wedge [\![y \geq w]\!]) \rightarrow [\![z \geq v + w]\!] \tag{4}$$

Although other encodings of the ternary inequality constraint can be devised, we will show how Eq. (4) is the shared structure between the GT, SWC, and BDD encoding. In the following subsections, we first describe the existing encoding method from the literature. Then, we show a particular decomposition of a PB constraint into ternary inequalities and the domains of the auxiliary integer variables. Finally, we prove how an order encoding of the integer variables and encoding of the ternary inequality constraints (by Eq. (4)) produces an equivalent set of clauses as the original encoding methods given the same PB constraint. The only difference is that some trivially assignable literals (e.g. by unit propagation) of the original encoding may already be removed in our method.

### 3.1 Generalized Totalizer

The GT encoding [22] constructs a complete binary tree bottom-up. Each node $Z$ is associated with a set of auxiliary variables $z_v$, where $v$ corresponds to the lower bound of the partial sum of the subtree. Each leaf node represents a PB term $q_i b_i$ with singleton set $\{z_{q_i}\}$, where $z_{q_i} \equiv b_i$. Every internal node $Z$ with child nodes $L$ and $R$ contains an auxiliary Boolean variable $z_v$ for every possible sum of the values of the auxiliary variables of child nodes $L$ and $R$, where those greater than $k$ are all represented by $k + 1$. That is, $Z = \{z_{min(v+w,k+1)} \mid l_v \in L, r_w \in R\}$. The GT is encoded by adding a unary clause $\neg a_{k+1}$ for the root node, and for every internal node $Z$:

$$l_v \rightarrow z_v \quad l_v \in L \tag{5a}$$

$$r_w \rightarrow z_w \quad r_w \in R \tag{5b}$$

$$(l_v \wedge r_w) \rightarrow z_{\min(v+w,k+1)} \quad l_v \in L, r_w \in R \tag{5c}$$

Clearly, the nodes of the GT encoding imply a higher-level structure, and has been formalized before as an equivalent PB-based decomposition [25]. We will now define the integer-based GT encoding. The auxiliary Boolean variables of every node $Z$ are represented by a single auxiliary integer variable $y_i$ of a binary tree decomposition of the following ternary inequality constraints:

$$\bigwedge_{i=1}^{n-1} l(i) + r(i) \leq y_i \tag{6}$$

where $D(y_i) = \{v + w : v \in D(l(i)), w \in D(r(i)), v + w \leq k\}$, except $D(y_{root}) = \{k\}$. We can show this equivalent to the GT encoding.

**Theorem 1.** *Encoding Eq.* (6) *is, after unit propagation, equivalent to GT.*

### 3.2 Sequential Weight Counter

In the SWC encoding [20], a series of $n$ weight counters is introduced. The $i$-th counter enforces $\sum_{i'=1}^{i} q_{i'} b_{i'} \leq k$ by introducing auxiliary variables $z_{i-1,j}$ for $1 \leq j \leq k$, where $z_{i,j}$ represents $\sum_{i'=1}^{i} q_{i'} b_{i'} \geq j$. We consider $z_{0,j} \equiv 0, 0 \leq j \leq k$. We encode the $i$-th counter with the following clauses:

$$z_{i-1,j} \rightarrow z_{i,j} \quad 1 \leq j \leq k \tag{7a}$$
$$\neg \left( z_{i-1,k+1-q_i} \wedge b_i \right) \tag{7b}$$
$$\left( z_{i-1,j} \wedge b_i \right) \rightarrow z_{i,j+q_i} \quad 1 \leq j \leq k - q_i \tag{7c}$$
$$b_i \rightarrow z_{i,j} \quad 1 \leq j \leq q_i \tag{7d}$$

From the integer perspective, the same encoding can be described using a linear tree that introduces $n + 1$ auxiliary integer variables $w_i \in [-k, 0]$ except $w_0 = 0$ and $w_n = -k$, and the following ternary inequalities.

$$\bigwedge_{i=1}^{n} x_i + w_i \leq w_{i-1} \tag{8}$$

These inequalities are derived from $x_i + y_{i-1} \leq y_i$ where $y_i \in [0, k]$, which is equivalent from an integer perspective to $x_i - y_i \leq -y_{i-1}$ assuming $w_i = -y_i$. However, encoding $x_i + y_{i-1} \leq y_i$ would lead to a subtle difference in clauses compared to the original SWC encoding.

**Theorem 2.** *Encoding Eq.* (8) *is, after unit propagation, equivalent to SWC.*

### 3.3 Binary Decision Diagram

A BDD [14] is a binary, rooted, directed, acyclic graph which models a Boolean function. Every node is associated with a selector variable which determines whether a path follows either the node's zero or one edge. When the path reaches a terminal node $z_\top$ or $z_\bot$, the function returns zero or one, respectively. In ordered BDDs, selector variables appear in the same order on all paths from the root, organizing the BDD into one layer per selector variable. A quasi-reduced QOBDDs has no isomorphic sub-BDDs. Fully reduced ROBDDs are quasi-reduced and have no *identity* nodes, where all edges target the same child.

Construction algorithms exist which produce ROBDDs that model satisfaction of PB constraints [4]. The layer $i$ models the decision variable $b_i$ using multiple nodes $(i, j)$, each associated with an auxiliary variable $z_{i,j}$. The set of edges $E$ contains $((i, j), (i', j'), p)$ with $p$ equal to zero (one) iff there is a zero (one) edge from node $(i, j)$ to node $(i', j')$. The ROBDD is encoded as follows:

$$z_{1,1} \wedge \neg z_\bot \wedge z_\top \tag{9a}$$
$$z_{i,j} \rightarrow z_{i',j'} \quad (i, j), (i', j'), 0) \in E \tag{9b}$$
$$\left( b_i \wedge z_{i,j} \right) \rightarrow z_{i'',j''} \quad (i, j), (i', j'), 1) \in E \tag{9c}$$

Before moving to the integer perspective, we establish two general properties of BDDs. First, an ROBDD can be converted to a QOBDD by replacing any *long* edges $((i,j),(i',j'),p)$ where $i'-i>1$ by chains of $i'-i-1$ identity nodes, so that $i'=i+1$ for all edges. This requires no additional variables or clauses, as the new identity node variables are all equivalent to $z_{i,j}$.

Second, we adapt the PB intervals of the above construction algorithm. A path from root node $(1,1)$ to node $(i,j)$ is associated with a partial assignment of $b_1, b_2, \ldots, b_{i-1}$, which yields a partial sum. Then, $W_{i,j}$ is the set of all partial sums given by all possible paths from the root node to node $(i,j)$. In a QOBDD, the sub-QOBDD, rooted at the node $(i,j)$, covers all PB constraints $w+\sum_{i'=i}^{n} q_i b_i \leq k$ in the interval $\min W_{i,j} \leq w \leq \max W_{i,j}$. Hence, intervals on the same layers do not overlap, as that would produce identical sub-ROBDD, contradicting the isomorphism property.

In the integer perspective, a single integer variable $y_i$ represents each layer $i$ of the QOBDD as a partial sum $\sum_{j=1}^{i} x_j \leq y_i$. This is enforced by the following linear tree of ternary inequalities:

$$\bigwedge_{i=1}^{n} x_i + y_{i-1} \leq y_i \tag{10}$$

The domain values 0 and $q_i$ of $x_i$ are represented by the zero and one edges at layer $i$, respectively. The $j$-th domain value $d_{i,j}$ of $y_i$ is represented by node $(i,j)$ and is equal to $\max W_{i,j}$, except for $y_0=0$ and $y_n=k$. The domain values can also be recursively calculated given QOBDD edges $E$, bottom-up:

$$d_{i,j} = \max\{d_{i-1,j} + pq_i : ((i-1,j'),(i,j),p) \in E\} \tag{11}$$

Two interesting notes on this specification are that

1. some non-reduced BDDs cannot be represented by our encoding of the ternary inequality constraints, and
2. other encodings of BDDs exist, which would require different constraint encodings of the ternary inequalities.

**Theorem 3.** *Encoding Eq.* (10) *is, after unit propagation, equivalent to BDD.*

## 4 Taking Advantage of the Integer Viewpoint

There are various advantages of specifying the PB encodings in terms of integer ternary inequalities. First, we will show that by proving the propagation consistency of the ternary inequality encoding, we can also prove the propagation consistency of the overall encoding of the PB constraint. Second, since the input of the specification is a integer linear constraint representing a PB constraint, we can support integer linear variables and constraints directly. This also applies to integer variables, which arise from detecting sets of literals that act like integers

due to existing side-constraints. Third, we can encode equality constraints more efficiently by replacing each ternary inequality in the decomposition for a ternary equality. Fourth, we can create encodings where the choice of each variable representation is mixed: using order encoding for small domain sizes to benefit from strong propagation, and binary encodings for those with large domain sizes to keep the encoding small.

### 4.1 Proving Consistency of all Encodings

Instead of proving consistency on every encoding individually, we can prove the propagation consistency on the ternary inequality, and use this to prove domain consistency of every decomposition. Surprisingly, the encoding of $x + y \leq z$, given by Eq. (4), does not maintain bounds($\mathbb{R}$) consistency:

*Example 2.* Consider the domains $D(x) = \{0, 4, 8\}$, $D(y) = \{0, 7, 10\}$, and $D(z) = \{0, 4, 7, 8, 10, 11\}$ then suppose we have $[\![y \geq 7]\!]$ and $\neg[\![z \geq 10]\!]$ ($z \leq 9$). We would hope to propagate $x \leq 2$, or equivalently $\neg[\![x \geq 4]\!]$, through the clause $\neg[\![x \geq 4]\!] \vee \neg[\![y \geq 7]\!] \vee [\![z \geq 11]\!]$. However, we do not have $\neg[\![z \geq 11]\!]$. The issue is that there is no IC to propagate $[\![z \geq 11]\!] \rightarrow [\![z \geq 10]\!]$. □

If we assume the upper bound of $z$ satisfies an IC, we can prove the following:

**Lemma 1.** *Encoding $x + y \leq z$ with Eq. (4) ensures that lower bounds on $x$ and $y$ are propagated correctly to $z$ (or cause failure), and when an upper bound $u$ on $z$ satisfies an IC (i.e. $[\![z \leq u]\!] \rightarrow [\![z \leq u + 1]\!]$), then propagation of upper bounds on $x$ and $y$ are correctly propagated, and these upper bounds also satisfy ICs.*

A consequence of the above theorem is that the decompositions of a PB constraint into a tree of ternary integer inequalities enforces bounds($\mathbb{R}$) consistency (or, equivalently for this constraint, domain consistency) by satisfying an implicit IC on upper bounds without explicitly encoding the clauses in Eq. (2):

**Theorem 4.** *The decomposition of a PB constraint $\sum_{i=1}^{n} q_i b_i \leq k$ into ternary inequalities encoded using Eq. (4) enforces domain consistency of the constraint.*

Note that the counterexample of Example 2 does not conflict with Theorem 4, since it cannot occur, unless we branch on intermediate literals, in this case by setting $\neg[\![z \geq 10]\!]$. Note, the proof given for this result for GT by Joshi *et al.* [22] seems to never mention the need for enforcing ICs, which suggests it may actually be incomplete.

### 4.2 Using Integer Decision Variables

An integer linear constraint $\sum_{i}^{n} q_i x'_i \leq k$ over order encoded integer variables $x'_i$ and constants $q_i, k$ can be converted to a PB constraint by splitting out every integer linear term $q_i x'_i$ into multiple PB terms using $\sum_{v \in D(x'_i)} q_i [\![x'_i \geq v]\!]$. However, encoding the resulting PB constraint leads to a high degree of redundancy,

since it introduces a GT leaf node, SWC counter, or BDD layer per domain value, for every integer linear term.

Since our ternary inequalities are defined over integers, the PB encoding methods are easily extended to integer linear encodings. As in Section 3, we use views $x_i \equiv q_i x'_i$, where $D(x_i) = \{q_i v : v \in D(x'_i)\}$ to establish a linear constraint with unit coefficients $\sum_{i=1}^{n} x_i \leq k$. Encoding the ternary inequalities results in just a single leaf node, counter, or layer per term. In particular, BDDs are thus generalized to *Multi-valued Decision Diagrams* (MDDs), as every node on layer $i$ will have one edge per domain value of $x_i$ (see e.g. [5]).

Even if constraints are not explicitly modelled using integer variables, we can use the knowledge of other constraints to treat sets of literals as such. In particular, we look for IC and *At-Most-One* (AMO) constraints, where a set of literals follow similar behaviour to order and direct encoded integer. Bofill *et al.* first explored how the presence of AMO constraints can lead to improved PB encodings [13]. We find that using our encodings the IC constraints can be similarly exploited. This results in novel encodings for GT and SWC, and, for BDD, this results in the same encoding as [2].

For some PB constraints, it might be efficient to introduce additional integer variables. Given a PB constraint $\sum_{i=1}^{n} q_i b_i \leq k$, suppose that there are shared coefficients $q_1 = q_i, 2 \leq i \leq l$ (w.l.o.g. we assume the first $l$ coefficients are the same). We can replace the term $q_1 b_1 + \cdots + q_1 b_l$ by an integer term $q_1 x$ where $D(x) = \{0, 1, \ldots, l\}$ and encode the constraint $b_1 + \cdots + b_l \leq x$ using a cardinality network (e.g. using [3]). This performs the addition in a more compact way, and generates new auxiliary variables for $x$ that recognize symmetric situations, and has been shown to improve encodings [5]. Note that the integer term $q_1 x$ is simply a view on the $x$ variable so that we do not need to introduce additional Booleans to encode it [2].

Another use case of integer decision variables is to re-use a (non-constant) integer variable at the root of one constraint's decomposition as a node for another, as has been done for GT [21].

### 4.3 Encoding Equality Constraints

Rather than split an equality constraint $\sum_{i=1}^{n} q_i b_i = k$ into two inequalities $\sum_{i=1}^{n} q_i b_i \leq k$ and $\sum_{i=1}^{n} q_i b_i \geq k$, it would be preferable to encode the equality directly. We can do so straightforwardly by decomposing the equality into integer equalities of the form $x + y = z$, and then generating an encoding of these directly. In fact, the unweighted totalizer encoding for cardinality constraints (where all $q_i = 1$) supports directly encoding both lower- and upper bound, and consequently supports equality as well [6]. Since domain consistency is NP hard (by reduction to subset sum [15]) we will only enforce bounds($\mathbb{R}$) consistency, which is equivalent to the consistency enforced by splitting into two inequalities.

The encoding simply reuses the PB encoding methods for $\sum_{i=1}^{n} q_i b_i \leq k$, with the following changes. Instead of decomposing to ternary inequalities $x + y \leq z$, we decompose to ternary equalities $x + y = z$. The domain computation for each introduced integer variable remains the same. The given ROBDD should model

the equality constraint, which can be achieved by changing the base case of the construction algorithm [4].

The encoding of the ternary equality constraint $x+y=z$ is given by the usual encoding of $x+y \leq z$ from Eq. (4), with the following encoding on $x+y \geq z$:

$$\bigwedge_{v \in D(x)} \bigwedge_{w \in D(y)} (\llbracket x \leq v \rrbracket \wedge \llbracket y \leq w \rrbracket) \rightarrow \llbracket z \leq v+w \rrbracket \tag{12}$$

The advantage of the equality encoding over splitting into two inequalities is that it reuses the same variables for the intermediate sums, rather than building two copies of the intermediate sums, after we normalize the second inequality.

*Example 3.* Consider encoding the equation $10b_1+7b_2+5b_3+2b_4+b_5 = 15$, using the same tree decomposition as in Example 1. The constraints are $x_2+x_3=y_1$, $x_1+y_2=y_2$, $x_4+x_5=y_3$, $y_2+y_3=y_4$, $y_4=15$, with the same domains as shown in Example 1. We encode each ternary equation as defined above, e.g. $y_2+y_3=y_4$ generates (trivial) clauses such as $\neg\llbracket y_2 \geq 5 \rrbracket \vee \neg\llbracket y_3 \geq 2 \rrbracket \vee \llbracket y_4 \geq 7 \rrbracket \equiv \mathit{true}$ from $(\llbracket y_2 \geq 5 \rrbracket \wedge \llbracket y_2 \geq 2 \rrbracket) \rightarrow \llbracket y_4 \geq 7 \rrbracket$ and more interesting clauses such as $\llbracket y_2 \geq 7 \rrbracket \vee \llbracket y_3 \geq 3 \rrbracket \vee \neg\llbracket y_4 \geq 8 \rrbracket$ from $(\llbracket y_2 \leq 5 \rrbracket \wedge \llbracket y_3 \leq 2 \rrbracket) \rightarrow \llbracket y_4 \leq 7 \rrbracket$, which is equivalent to just $\llbracket y_2 \geq 7 \rrbracket \vee \llbracket y_3 \geq 3 \rrbracket$. □

We can prove results analogous to Lemma 1 and Theorem 4 about the encoding of $x+y \geq z$ generated by Eq. (12).

**Theorem 5.** *The decomposition of a pseudo-Boolean constraint* $\sum_{i=1}^{n} q_i b_i \geq k$ *into ternary inequalities encoded using Eq.* (12) *enforces domain consistency of the constraint.*

### 4.4 Mixed Integer Variable Encodings of the Decompositions

By Theorem 4, order encoding the integer variables in a decomposition enforces domain consistency on the inequality constraint. However, the number of clauses in Eq. (4) grow cubically with the domains of the variables. This makes the encoding less effective as domain sizes grow. Alternatively, a binary encoding of integer variables creates fewer clauses, but trades in the consistency guarantees.

Depending on the chosen decomposition, the domains of the different auxiliary variables vary in size. In the SWC, the domains of (non-constant) auxiliary variables are of the same size. In the GT, the size of auxiliary domains grows at each subsequent layer. In the BDD decomposition, the auxiliary domain values directly correspond with the nodes at each layer of a given QOBDD. They start with one node at the root layer, grow in number until the middle layer, and then shrink again until the terminal layer. Since the decomposition is agnostic as to which variable encoding is used for which variable, we can choose the order encoding for domains up to size $c$, and binary for larger domains. We will denote this simple heuristic by $\mathbb{O} \leq c$.

Consider the equality constraint $x+y=z$ in a decomposition of an inequality constraint. Suppose we are given a pre-determined choice of order or binary encoding for each variable indicated by $:\mathbb{O}$ or $:\mathbb{B}$, respectively. So far, these choices

have always been $x{:}\mathbb{O} + y{:}\mathbb{O} = z{:}\mathbb{O}$, which we have encoded using Eq. (4) after relaxing to an inequality constraint. If instead the choices are $x{:}\mathbb{B}+y{:}\mathbb{B} = z{:}\mathbb{B}$, we can directly apply the RCA encoding of Eq. (3) for ternary equalities. Unfortunately, for mixed encodings such as $x{:}\mathbb{O}+y{:}\mathbb{B} = z{:}\mathbb{B}$, often there are no (efficient) encodings. However, there are efficient encodings for $x{:}\mathbb{O} \ \# \ x{:}\mathbb{B}, \# \in \{\leq, \geq, =\}$ [12]. We can introduce $x{:}\mathbb{B}$ as an additional integer variable and decompose the constraint to $x{:}\mathbb{O} \leq x{:}\mathbb{B} \wedge x{:}\mathbb{B} + y{:}\mathbb{B} = y{:}\mathbb{B}$. If the decomposition contains a ternary inequality constraint $x{:}\mathbb{B} + y{:}\mathbb{B} \leq z{:}\mathbb{B}$, then to apply the RCA encoding, we rewrite it to an equality constraint. To avoid removing solutions, we change the auxiliary variable's domain $D(z) = \min(\mathrm{lb}_D(z), \mathrm{lb}_D(x) + \mathrm{lb}_D(y))..\mathrm{ub}_D(z)$.

*Example 4.* Consider the PB constraint $8b_1 + 6b_2 + 3b_3 + 3b_4 + 2b_5 \leq 10$. After constructing a BDD, we decompose using Eqs. (10) and (11) and order encode all variables with domain size up to 3 (i.e. $\mathbb{O} \leq 3$):

$$
\begin{aligned}
x_1{:}\mathbb{O} \leq y_1{:}\mathbb{O} \quad & \text{where } D(x_1{:}\mathbb{O}) = \{0, 8\}, D(y_1{:}\mathbb{O}) = \{1, 8\} \\
x_2{:}\mathbb{O} + y_1{:}\mathbb{O} \leq y_2{:}\mathbb{O} \quad & \text{where } D(x_2{:}\mathbb{O}) = \{0, 6\}, D(y_2{:}\mathbb{O}) = \{2, 7, 8\} \\
x_3{:}\mathbb{O} + y_2{:}\mathbb{O} \leq y_3{:}\mathbb{B} \quad & \text{where } D(x_3{:}\mathbb{O}) = \{0, 3\}, D(y_3{:}\mathbb{B}) = \{5, 7, 8, 10\} \\
x_4{:}\mathbb{O} + y_3{:}\mathbb{B} \leq y_4{:}\mathbb{O} \quad & \text{where } D(x_4{:}\mathbb{O}) = \{0, 3\}, D(y_4{:}\mathbb{O}) = \{8, 10\} \\
x_5{:}\mathbb{O} + y_4{:}\mathbb{O} \leq 10 \quad & \text{where } D(x_5{:}\mathbb{O}) = \{0, 2\}
\end{aligned}
$$

This requires further decomposition of the two mixed encoding constraints:

$$
\begin{aligned}
x_3{:}\mathbb{O} \leq x_3{:}\mathbb{B} \quad & \text{where } D(x_3{:}\mathbb{O}) = D(x_3{:}\mathbb{B}) = \{0, 3\} \\
y_2{:}\mathbb{O} \leq y_2{:}\mathbb{B} \quad & \text{where } D(y_2{:}\mathbb{O}) = \{2, 7, 8\}, D(y_2{:}\mathbb{B}) = \{2, 7, 8\} \\
x_3{:}\mathbb{B} + y_2{:}\mathbb{B} = y_3{:}\mathbb{B} \quad & \text{where } D(y_3{:}\mathbb{B}) = 2..10 \\
x_4{:}\mathbb{O} \leq x_4{:}\mathbb{B} \quad & \text{where } D(x_4{:}\mathbb{O}) = D(x_4{:}\mathbb{B}) = \{0, 3\} \\
x_4{:}\mathbb{B} + y_3{:}\mathbb{B} = y_4{:}\mathbb{B} \quad & \text{where } D(y_4{:}\mathbb{B}) = 2..10 \\
y_4{:}\mathbb{B} \geq y_4{:}\mathbb{O} \quad & \text{where } D(y_4{:}\mathbb{O}) = \{8, 10\}
\end{aligned}
$$

□

To support binary encoded integer decision variables (see Section 4.2), representing a term $x_i = q_i x_i'{:}\mathbb{B}$ is not without overhead (unlike $x_i = q_i x_i'{:}\mathbb{O}$). We use efficient encodings which are pre-computed for a given coefficient and domain [11]. By using common sub-expression elimination of terms, we avoid introducing the same encoding twice [16]. To encode decompositions of equality constraints (see Section 4.3), we apply $x{:}\mathbb{O} = x{:}\mathbb{B}$ [12].

## 5 Experimental Evaluation

In this section, we evaluate the GT and BDD decomposition, and apply each practical extension. First, we test the PB encoding in its base form by converting each integer decision variable into PB terms using the order encoding. The next configuration (+$\mathbb{O}$) supports the integer linear constraint directly (Section 4.2). Next, we add direct support for equality constraints (+`eq`), rather than splitting

each in two inequalities (Section 4.3). Finally, we mix in the binary encoding for (auxiliary) integer variables ($+\mathbb{O} \leq c$) using two cut-offs (Section 4.4), as well as a uniform binary approach ($+\mathbb{B}$). We omit the SWC decomposition because its constant size auxiliary variable domains are not as interesting for the mixed encoding, and due to space limitations.

As baseline, we include various configurations of three established SAT encoders which can encode integer linear constraints. Savile Row (version 1.10.1) implements the fundamental PB encodings that we study in this paper, as well as binary encoding approaches such as *Generalized n-Level Modulo Totalizer* (GMTO) and *Global Polynomial Watchdog* (GPW) [7]. These PB encodings are generalized for AMO side-constraints, allowing for principal integer variables, the GT encoding is enhanced by a BDD-like reduction algorithm [13]. Equality constraints are separately encoded as two inequalities. Picat-SAT (version 3.5) primarily uses the binary encoding with RCA and ad-hoc optimization [27, 28]. Fun-sCOP (version `20230601-13h09m`) hybridizes order and binary encodings of the principal integer variables [24]. Yet, its binary encoding employs PB encodings such as BDD, which still order encode the auxiliary variables uniformly. Consequently, there was no difference between their binary and mixed approach. We have omitted PBLib [23], another common PB encoding library, as it does not support side constraints or integer variables, and we have found that it is not competitive in an integer setting.

All encodings are solved using the same executable of CaDiCaL (v2.1.0) [8] with a 3 GB memory limit and a time limit of 60 seconds. For two different integer linear problems, we generate three instance sets. We compare the number of solved instances, followed by their average solve time, in parentheses. The encoding size is shown as the number of thousands of variables and clauses, with the number of memory- and timeouts (if any) indicated by a superscript, or by — if all failed to encode. Our implementation of the encoding methods, the generated benchmark instances, and the benchmark scripts are available [10].

### 5.1 Multidimensional Bounded Knapsack Problem

In the *Multidimensional Bounded Knapsack Problem* (MBKP) we decide for $N$ item types how many $x_i$ to pack (up to $B$) such that a minimum profit $\sum_{i=1}^{N} x_i p_i \geq P$ is reached. Each item is restricted by $M$ dimensions of weight with $\bigwedge_{i=1}^{M} \sum_{i=j}^{N} x_j w_{i,j} \leq W_i$. Adapted from [18], instances can be generated by first generating $C$ coefficient sets for $w_{i,j}$ and $p_i$, sampling uniformly from $[1, Q]$. Then, for each coefficient set, we choose $S$ capacity sets for $1 \leq s \leq S$ with a capacity factor $f = \frac{s}{S}$: generating $W_i = f \sum_{j=1}^{N} B w_{i,j}, 1 \leq i \leq M$, and $P = (1 - f) \sum_{j=1}^{N} B p_j$. As $f$ increases, the constraints become less strict, and instances turn from unsatisfiable to satisfiable. Since the low and high ends of $f$ are trivial, we normalize $f$ to be within $0.4 \leq f \leq 0.6$. We generate one PB instance set (where B=1), and two integer linear instance sets with different Q.

In the results shown in Table 1, the effects of using integer decision variables ($+\mathbb{O}$) is difficult to compare due to encoding memory- and timeouts. Using

| $(N, B, M, Q)$ | $(50, 1, 25, 50)$ | | $(10, 10, 200, 50)$ | | $(10, 10, 200, 250)$ | |
|---|---|---|---|---|---|---|
| | solved | vars./cl. | solved | vars./cl. | solved | vars./cl. |
| GT + $\mathbb{B}$ | 84 (6.0) | 15/76 | 85 (11.5) | 36/224 | 74 (16.5) | 52/332 |
| GT + $\mathbb{O} \leq 25$ | 88 (5.7) | 14/54 | **94 (10.9)** | 39/238 | 87 (14.9) | 58/337 |
| GT + $\mathbb{O} \leq 75$ | **89 (5.7)** | 14/54 | 93 (11.8) | 43/252 | **91 (16.6)** | 58/340 |
| GT + $\mathbb{O}$ | 67 (3.5) | 84/7852 | 0 | 701/159425 | — | — |
| GT | 68 (3.9) | 84/7852 | 0 | 985/164102 | — | — |
| BDD + $\mathbb{B}$ | 78 (3.3) | 23/99 | 78 (10.1) | 40/234 | 68 (12.0) | 56/343 |
| BDD + $\mathbb{O} \leq 25$ | 75 (8.9) | 26/113 | 93 (13.4) | 42/244 | 86 (15.1) | 58/334 |
| BDD + $\mathbb{O} \leq 75$ | 76 (10.2) | 29/114 | 90 (13.8) | 43/252 | 88 (16.1) | 58/335 |
| BDD + $\mathbb{O}$ | 75 (6.4) | 351/694 | 61 (17.4) | 775/7983 | 0 | 2723/28503 |
| BDD | 74 (5.2) | 351/694 | 0 | 8313/16588 | 0 | 30312/60561 |
| Savile Row (GT) | 82 (2.3) | 61/5946 | 0 | $610/108981^{56}$ | — | — |
| Savile Row (SWC) | 74 (13.9) | 795/1561 | 9 (13.9) | 2244/22459 | 0 | $10566/105809^{30}$ |
| Savile Row (BDD) | 86 (0.8) | 230/446 | 44 (2.8) | 582/5535 | 40 (2.2) | 2019/19390 |
| Savile Row (GPW) | 87 (3.5) | 22/117 | 74 (12.1) | 83/288 | 72 (13.1) | 111/394 |
| Savile Row (GMTO) | 87 (5.1) | 20/43 | 89 (8.0) | 50/179 | 84 (10.9) | 63/234 |
| Fun-sCOP | 83 (0.2) | 232/667 | 58 (8.5) | 1252/3512 | 44 (9.0) | 5405/15092 |
| Picat-SAT (RCA) | 81 (6.5) | 10/62 | 70 (16.7) | 27/211 | 47 (19.6) | 42/359 |

**Table 1.** Results for 3 sets of 100 MBKP instances ($C = 4, S = 25, 0.4 \leq f \leq 0.6$).

the uniform binary encoding (+$\mathbb{B}$) improves results in every case, presumably because the auxiliary domains are considerable. A relatively small cut-off value, with only a minority of all domain values order encoded, pushes the number of solved instances for both GT and BDD on all instance set, with the former outperforming the control encoders. In experiments not shown here, a larger cut-off value of 150 or 300 was less effective.

### 5.2 Multidimensional Bounded Subset Sum Problem

The *Multidimensional Bounded Subset Sum Problem* (MBSSP) is similar to MBKP, but requires packed items to sum up to an exact capacity using equality constraints. We decide for item types $1 \leq i \leq N$ how many $x_i$ items to select (up to bound $B$) such that an exact subset sum $k_i$ is reached for each $1 \leq j \leq M$ dimension given the weight $q_{i,j} \in [1, Q]$ of the item $j$ in the dimension $i$. That is, $\bigwedge_{i=1}^{M} \sum_{j=1}^{N} q_{i,j} x_j = k_i$. We generate instances by uniformly sampling all $q_{i,j}$. Then, to guarantee a solution exists, we uniformly sample an assignment $a_j$ for every $x_j$, yielding feasible values for $k_i = \sum_{j=1}^{N} q_{i,j} a_j$ for $1 \leq i \leq M$. Again, we generate one PB and two integer linear instance sets.

The results in Table 2 show an improvement when using integer variables (+$\mathbb{O}$) on non-PB instances. The support for equality constraints predictably halves the number of auxiliary variables, and makes a positive effect on solver performance overall. Again, the binary encoding approach is better than the order-based ones, and equals the best control for two out of three instance sets. Mixing in the order encoding does not improve performance for MBSSP, as it perhaps suits the equality constraints less.

| $(N, B, M, Q)$ | $(40, 1, 15, 50)$ | | $(12, 10, 8, 50)$ | | $(12, 10, 8, 250)$ | |
|---|---|---|---|---|---|---|
| | solved | vars./cl. | solved | vars./cl. | solved | vars./cl. |
| GT + eq + $\mathbb{B}$ | 100 (2.5) | 7/34 | 100 (1.2) | 2/14 | **100 (1.4)** | 3/19 |
| GT + eq + $\mathbb{O} \leq 25$ | 100 (2.4) | 6/55 | 100 (7.0) | 2/58 | 99 (10.3) | 3/253 |
| GT + eq + $\mathbb{O} \leq 75$ | 100 (2.3) | 6/55 | 100 (9.4) | 3/59 | 100 (11.7) | 3/253 |
| GT + eq + $\mathbb{O}$ | 6 (36.8) | 32/5794 | 0 | 38/22574 | 0 | $103/177961^{96}$ |
| GT + $\mathbb{O}$ | 4 (46.4) | 68/6446 | 0 | 81/23596 | 0 | $251/207563^{99}$ |
| GT | 2 (37.9) | 68/6446 | 0 | 101/25768 | 0 | $291/196827^{97}$ |
| BDD + eq + $\mathbb{B}$ | 99 (9.8) | 10/44 | **100 (1.1)** | 2/14 | **100 (1.4)** | 3/20 |
| BDD + eq + $\mathbb{O} \leq 25$ | — | — | 100 (6.1) | 3/59 | 100 (10.2) | 3/254 |
| BDD + eq + $\mathbb{O} \leq 75$ | — | — | 100 (8.1) | 3/60 | 100 (10.1) | 3/254 |
| BDD + eq + $\mathbb{O}$ | 35 (35.7) | 116/458 | 7 (42.6) | 49/1024 | 0 | 192/4045 |
| BDD + $\mathbb{O}$ | 29 (35.3) | 232/458 | 5 (34.3) | 99/1024 | 1 (17.3) | 385/4048 |
| BDD | 32 (36.3) | 232/458 | 0 | 1037/2068 | 0 | 4117/8225 |
| Savile Row (GT) | **100 (0.2)** | 47/3190 | 0 | 67/15607 | — | — |
| Savile Row (SWC) | 33 (48.3) | 597/1166 | 1 (24.7) | 269/2728 | 0 | 1322/13412 |
| Savile Row (BDD) | 34 (28.0) | 153/294 | 0 | 75/725 | 0 | 289/2816 |
| Savile Row (GPW) | 98 (12.4) | 19/89 | 15 (25.6) | 9/31 | 9 (25.2) | 11/42 |
| Savile Row (GMTO) | 94 (13.7) | 18/39 | 11 (31.7) | 5/18 | 8 (32.1) | 6/23 |
| Fun-sCOP | 26 (38.3) | 156/442 | 5 (25.6) | 155/435 | 1 (46.7) | 688/1930 |
| Picat-SAT (RCA) | 81 (16.1) | 5/29 | 100 (1.6) | 2/13 | 100 (1.8) | 2/19 |

**Table 2.** Results for 3 sets of 100 MBSSP instances.

## 6 Conclusion and Future Work

In this paper, we have shown that by thinking of encodings of PB constraints as computing partial sums, we can reframe the fundamental GT, SWC, and BDD encodings as integer decompositions. But once we have the *integer viewpoint*, any improvement on the decomposition automatically improves each encoding method. We extend the decomposition by supporting linear constraints, equality constraints, and mixed encodings for each individual integer variable. We show how each extension practically improves each method across two types of benchmarks and compared to three strong baseline encoders.

In future work, additional PB encodings may be understood from the same viewpoint. Some methods (e.g. Sorting Networks [17]) may also use an implicit order encoding for their auxiliary variables, while others may use the binary or direct encoding (e.g. alternative BDD encodings [1]). Further extensions of the decomposition (e.g. by improving its encoding) could also be of interest.

**Acknowledgments.** This research was partially funded by the Australian Government through the Australian Research Council Industrial Transformation Training Centre in Optimisation Technologies, Integrated Methodologies, and Applications (OPTIMA), Project ID IC200100009, and H. Bierlee was also partially funded by the European Research Council (ERC) under the EU Horizon 2020 research and innovation programme (Grant No 101002802, CHAT-Opt).

**Disclosure of Interests.** The authors have no competing interests to declare that are relevant to the content of this article.

## References


1. Abio, I., Gange, G., Mayer-Eichberger, V., Stuckey, P.J.: On CNF encodings for decision diagrams. In: Quimper, C.G. (ed.) Thirteenth International Conference on Integration of Artificial Intelligence and Operations Research techniques in Constraint Programming. pp. 1–17. No. 9676 in LNCS (2016)
2. Abío, I., Mayer-Eichberger, V., Stuckey, P.J.: Encoding linear constraints with implication chains to CNF. In: Pesant, G. (ed.) Principles and Practice of Constraint Programming - 21st International Conference, CP 2015, Cork, Ireland, August 31 - September 4, 2015, Proceedings. Lecture Notes in Computer Science, vol. 9255, pp. 3–11. Springer (2015). https://doi.org/10.1007/978-3-319-23219-5_1, https://doi.org/10.1007/978-3-319-23219-5_1
3. Abío, I., Nieuwenhuis, R., Oliveras, A., Rodríguez-Carbonell, E.: A parametric approach for smaller and better encodings of cardinality constraints. In: Schulte, C. (ed.) Principles and Practice of Constraint Programming - 19th International Conference, CP 2013, Uppsala, Sweden, September 16-20, 2013. Proceedings. Lecture Notes in Computer Science, vol. 8124, pp. 80–96. Springer (2013). https://doi.org/10.1007/978-3-642-40627-0_9, https://doi.org/10.1007/978-3-642-40627-0_9
4. Abío, I., Nieuwenhuis, R., Oliveras, A., Rodríguez-Carbonell, E., Mayer-Eichberger, V.: A new look at BDDs for pseudo-boolean constraints. J. Artif. Intell. Res. **45**, 443–480 (2012). https://doi.org/10.1613/jair.3653, https://doi.org/10.1613/jair.3653
5. Abio, I., Stuckey, P.J.: Encoding linear constraints into SAT. In: O'Sullivan, B. (ed.) Proceedings of the 20th International Conference on Principles and Practice of Constraint Programming. LNCS, vol. 8656, pp. 75–91. Springer (2014). https://doi.org/http://dx.doi.org/10.1007/978-3-319-10428-7_9
6. Bailleux, O., Boufkhad, Y.: Efficient CNF encoding of boolean cardinality constraints. In: Rossi, F. (ed.) Principles and Practice of Constraint Programming - CP 2003, 9th International Conference, CP 2003, Kinsale, Ireland, September 29 - October 3, 2003, Proceedings. Lecture Notes in Computer Science, vol. 2833, pp. 108–122. Springer (2003). https://doi.org/10.1007/978-3-540-45193-8_8, https://doi.org/10.1007/978-3-540-45193-8_8
7. Bailleux, O., Boufkhad, Y., Roussel, O.: New encodings of pseudo-boolean constraints into CNF. In: Kullmann, O. (ed.) Theory and Applications of Satisfiability Testing - SAT 2009, 12th International Conference, SAT 2009, Swansea, UK, June 30 - July 3, 2009. Proceedings. Lecture Notes in Computer Science, vol. 5584, pp. 181–194. Springer (2009). https://doi.org/10.1007/978-3-642-02777-2_19, https://doi.org/10.1007/978-3-642-02777-2_19
8. Biere, A., Fazekas, K., Fleury, M., Heisinger, M.: CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling entering the SAT Competition 2020. In: Balyo, T., Froleyks, N., Heule, M., Iser, M., Järvisalo, M., Suda, M. (eds.) Proc. of SAT Competition 2020 – Solver and Benchmark Descriptions. Department of Computer Science Report Series B, vol. B-2020-1, pp. 51–53. University of Helsinki (2020)
9. Biere, A., Heule, M., van Maaren, H., Walsh, T. (eds.): Handbook of Satisfiability. IOS Press, 2 edn. (May 2021), google-Books-ID: dUAvEAAAQBAJ

10. Bierlee, H., Dekker, J.J.: Pindakaas: CPAIOR-25 (submission) (Dec 2024). https://doi.org/10.5281/zenodo.14500064, https://doi.org/10.5281/zenodo.14500064
11. Bierlee, H., Dekker, J.J., Lagoon, V., Stuckey, P.J., Tack, G.: Single constant multiplication for SAT. In: Dilkina, B. (ed.) Integration of Constraint Programming, Artificial Intelligence, and Operations Research - 21st International Conference, CPAIOR 2024, Uppsala, Sweden, May 28-31, 2024, Proceedings, Part I. Lecture Notes in Computer Science, vol. 14742, pp. 84–98. Springer (2024). https://doi.org/10.1007/978-3-031-60597-0_6, https://doi.org/10.1007/978-3-031-60597-0_6
12. Bierlee, H., Gange, G., Tack, G., Dekker, J.J., Stuckey, P.J.: Coupling different integer encodings for SAT. In: Schaus, P. (ed.) Proceedings of the 19th International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR 2022). pp. 44–63. No. 13292 in LNCS, Springer (2022)
13. Bofill, M., Coll, J., Nightingale, P., Suy, J., Ulrich-Oltean, F., Villaret, M.: SAT encodings for pseudo-boolean constraints together with at-most-one constraints. Artif. Intell. **302**, 103604 (2022). https://doi.org/10.1016/j.artint.2021.103604, https://doi.org/10.1016/j.artint.2021.103604
14. Bryant, R.: Graph-based algorithms for boolean function manipulation. IEEE Transactions on Computers **C-35**(8), 677–691 (1986). https://doi.org/10.1109/TC.1986.1676819
15. Choi, C., Harvey, W., Lee, J., Stuckey, P.: Finite domain bounds consistency revisited. In: Proceedings of the Australian Conference on Artificial Intelligence 2006. LNCS, vol. 4304, pp. 49–58. Springer-Verlag (2006)
16. Cocke, J.: Global common subexpression elimination. In: Northcote, R.S. (ed.) Proceedings of a Symposium on Compiler Optimization, Urbana-Champaign, Illinois, USA, July 27-28, 1970. pp. 20–24. ACM (1970). https://doi.org/10.1145/800028.808480, https://doi.org/10.1145/800028.808480
17. Eén, N., Sörensson, N.: Translating pseudo-boolean constraints into SAT. J. Satisf. Boolean Model. Comput. **2**(1-4), 1–26 (2006). https://doi.org/10.3233/sat190014, https://doi.org/10.3233/sat190014
18. Han, B., Leblet, J., Simon, G.: Hard multidimensional multiple choice knapsack problems, an empirical study. Comput. Oper. Res. **37**(1), 172–181 (2010). https://doi.org/10.1016/j.cor.2009.04.006, https://doi.org/10.1016/j.cor.2009.04.006
19. Harvey, W., Stuckey, P.: Improving linear constraint propagation by changing constraint representation. Constraints **8**(2), 173–207 (2003)
20. Hölldobler, S., Manthey, N., Steinke, P.: A compact encoding of pseudo-boolean constraints into SAT. In: Glimm, B., Krüger, A. (eds.) KI 2012: Advances in Artificial Intelligence - 35th Annual German Conference on AI, Saarbrücken, Germany, September 24-27, 2012. Proceedings. Lecture Notes in Computer Science, vol. 7526, pp. 107–118. Springer (2012). https://doi.org/10.1007/978-3-642-33347-7_10, https://doi.org/10.1007/978-3-642-33347-7_10
21. Jabs, C., Berg, J., Järvisalo, M.: Core boosting in sat-based multi-objective optimization. In: Dilkina, B. (ed.) Integration of Constraint Programming, Artificial Intelligence, and Operations Research - 21st International Conference, CPAIOR 2024, Uppsala, Sweden, May 28-31, 2024, Proceedings, Part II. Lecture Notes in Computer Science, vol. 14743, pp. 1–19. Springer (2024). https://doi.org/10.1007/978-3-031-60599-4_1, https://doi.org/10.1007/978-3-031-60599-4_1

22. Joshi, S., Martins, R., Manquinho, V.M.: Generalized totalizer encoding for pseudo-boolean constraints. In: Pesant, G. (ed.) Principles and Practice of Constraint Programming - 21st International Conference, CP 2015, Cork, Ireland, August 31 - September 4, 2015, Proceedings. Lecture Notes in Computer Science, vol. 9255, pp. 200–209. Springer (2015). https://doi.org/10.1007/978-3-319-23219-5_15, https://doi.org/10.1007/978-3-319-23219-5_15
23. Philipp, T., Steinke, P.: Pblib - A library for encoding pseudo-boolean constraints into CNF. In: Heule, M., Weaver, S.A. (eds.) Theory and Applications of Satisfiability Testing - SAT 2015 - 18th International Conference, Austin, TX, USA, September 24-27, 2015, Proceedings. Lecture Notes in Computer Science, vol. 9340, pp. 9–16. Springer (2015). https://doi.org/10.1007/978-3-319-24318-4_2, https://doi.org/10.1007/978-3-319-24318-4_2
24. Soh, T., Le Berre, D., Banbara, M., Tamura, N.: scop: Sat-based constraint programming system. Proceedings of XCSP3 Competition 2018 (XCSP18) pp. 93–94 (2018)
25. Vandesande, D., Wulf, W.D., Bogaerts, B.: Qmaxsatpb: A certified maxsat solver. In: Gottlob, G., Inclezan, D., Maratea, M. (eds.) Logic Programming and Nonmonotonic Reasoning - 16th International Conference, LPNMR 2022, Genova, Italy, September 5-9, 2022, Proceedings. Lecture Notes in Computer Science, vol. 13416, pp. 429–442. Springer (2022). https://doi.org/10.1007/978-3-031-15707-3_33, https://doi.org/10.1007/978-3-031-15707-3_33
26. Warners, J.P.: A linear-time transformation of linear inequalities into conjunctive normal form. Inf. Process. Lett. **68**(2), 63–69 (1998). https://doi.org/10.1016/S0020-0190(98)00144-6, https://doi.org/10.1016/S0020-0190(98)00144-6
27. Zhou, N., Kjellerstrand, H.: The Picat-SAT compiler. In: Gavanelli, M., Reppy, J.H. (eds.) Practical Aspects of Declarative Languages - 18th International Symposium, PADL 2016, St. Petersburg, FL, USA, January 18-19, 2016. Proceedings. Lecture Notes in Computer Science, vol. 9585, pp. 48–62. Springer (2016). https://doi.org/10.1007/978-3-319-28228-2_4, https://doi.org/10.1007/978-3-319-28228-2_4
28. Zhou, N., Kjellerstrand, H.: Optimizing SAT encodings for arithmetic constraints. In: Beck, J.C. (ed.) Principles and Practice of Constraint Programming - 23rd International Conference, CP 2017, Melbourne, VIC, Australia, August 28 - September 1, 2017, Proceedings. Lecture Notes in Computer Science, vol. 10416, pp. 671–686. Springer (2017). https://doi.org/10.1007/978-3-319-66158-2_43, https://doi.org/10.1007/978-3-319-66158-2_43

## Appendix 1: proofs

*Proof (Theorem 1).*

We show that the clauses generated by the integer tree decomposition are (after closing under unit propagation) the same as those for the GT encodings. We consider a single node $Z$ with child nodes $L$ and $R$ from the GT encoding and corresponding integer constraint $l(i) + r(i) \leq y_i$. The binary tree of the decomposition is the same as in the GT encoding, where $[\![y_i \geq v]\!] \equiv z_v$, $[\![l(i) \geq v]\!] \equiv l_v$ and $[\![r(i) \geq v]\!] \equiv r_v$. Encoding the ternary inequality of Eq. (6) yields the following clauses.

$$\bigwedge_{v \in D(l(i))} \bigwedge_{w \in D(r(i))} ([\![l(i) \geq v]\!] \wedge [\![r(i) \geq w]\!]) \rightarrow [\![y_i \geq v + w]\!]$$

$$\equiv \left( \bigwedge_{v \in D(l(i))} ([\![l(i) \geq v]\!] \wedge [\![r(i) \geq 0]\!]) \rightarrow [\![y_i \geq v]\!] \right)$$

$$\wedge \left( \bigwedge_{w \in D(r(i))} ([\![l(i) \geq 0]\!] \wedge [\![r(i) \geq w]\!]) \rightarrow [\![y_i \geq w]\!] \right)$$

$$\wedge \left( \bigwedge_{v \in D(l(i)) \setminus \{0\}} \bigwedge_{w \in D(r(i)) \setminus \{0\}} ([\![l(i) \geq v]\!] \wedge [\![r(i) \geq w]\!]) \rightarrow [\![y_i \geq v + w]\!] \right)$$

The case decomposition is correct since 0 is the lower bound of every internal variable $y_i$, except for $y_{root}$, which is fixed to $k$. We show these are the same clauses as generated for a GT node. The literals $[\![l(i) \geq 0]\!] = 1$ and $[\![r(i) \geq 0]\!] = 1$ by Eq. (1a). We can skip iterations where $[\![y_i \geq 0]\!] = 1$, since the resulting clause is satisfied. In the GT encoding, the unary clause $\neg a_{k+1}$ will fix $z_{k+1} = 0$ if present in its child node $Z$, by $z_{k+1} \rightarrow a_{k+1}$. This propagates through the entire tree, so that $z_{k+1} \equiv 0$ for all nodes. Consequently, the first two conjuncts are equivalent to the binary clauses Eqs. (5a) and (5b) of GT since $D(l(i)) = \{v \mid l_v \in L \setminus \{l_{k+1}\}\} \cup \{0\}$ and $D(r(i)) = \{v \mid r_v \in R \setminus \{r_{k+1}\}\} \cup \{0\}$:

$$\left( \bigwedge_{v \in D(r(i)) \setminus \{0\}} [\![l(i) \geq v]\!] \rightarrow [\![y_i \geq v]\!] \right) \wedge \left( \bigwedge_{w \in D(l(i)) \setminus \{0\}} [\![r(i) \geq w]\!] \rightarrow [\![y_i \geq w]\!] \right)$$

The remaining (third case) clauses are, after unit propagation, equivalent to the ternary clauses of the GT decomposition. Since, for any $v+w > k$, the literal $[\![y_i \geq v + w]\!] \equiv 0$, which is simply falsified (leading to a binary clause, rather than a ternary clause including $z_{k+1}$).

For the root node if $v + w \leq k$ then the literal $[\![y_i \geq v + w]\!] \equiv 1$ by Eq. (1a), and so the resulting clauses are trivial; otherwise, they agree with those in GT. Note that the GT decomposition adds Boolean variables $\{a_v \mid v \leq k\}$ for the

root node, but since they only appear positively, they can be trivially assigned to 1 (by SAT preprocessing).

*Proof (Theorem 2).* Encoding each ternary inequality $x_i + w_i \leq w_{i-1}$ yields the following clauses:

$$\begin{aligned} &\bigwedge_{v \in D(x_i)} \bigwedge_{u \in D(w_i)} ([\![x_i \geq v]\!] \wedge [\![w_i \geq w]\!]) \rightarrow [\![w_{i-1} \geq v + u]\!] \\ \equiv &\bigwedge_{v \in D(x_i)} \bigwedge_{u \in D(w_i)} ([\![x_i \geq v]\!] \wedge [\![w_{i-1} < v + u]\!]) \rightarrow [\![w_i < u]\!] \end{aligned}$$

We split the conjunctions:

$$\left( \bigwedge_{u=-k}^{0} ([\![x_i \geq 0]\!] \wedge [\![w_{i-1} < 0 + u]\!]) \rightarrow [\![w_i < u]\!] \right) \tag{13a}$$

$$\wedge \left( ([\![x_i \geq q_i]\!] \wedge [\![w_{i-1} < q_i - k]\!]) \rightarrow [\![w_i < -k]\!] \right) \tag{13b}$$

$$\wedge \left( \bigwedge_{u=-k+1}^{-q_i} ([\![x_i \geq q_i]\!] \wedge [\![w_{i-1} < q_i + u]\!]) \rightarrow [\![w_i < u]\!] \right) \tag{13c}$$

$$\wedge \left( \bigwedge_{u=-q_i+1}^{0} ([\![x_i \geq q_i]\!] \wedge [\![w_{i-1} < q_i + u]\!]) \rightarrow [\![w_i < u]\!] \right) \tag{13d}$$

When we equate $[\![w_i < u]\!] \equiv z_{i,-u+1}$, for $1 \leq i \leq n, -k \leq u \leq 0$ we can show that the four cases above correspond to the four cases in the SWC encoding. The first case, Eq. (13a), $[\![x_i \geq 0]\!] \equiv 1$, and $[\![w_i - 1 < -k]\!] \equiv 0$, and the resulting clauses are equivalent after unit propagation to Eq. (7a) and removing the trivial clause (when $u = -k$). In the second case, Eq. (13b), the clauses are equivalent once we note that $[\![w_i < -k]\!] \equiv 0$. The third case Eq. (13c), produces exactly the clauses from Eq. (7c), generated in reverse order, i.e. $\bigwedge_{u=-k+1}^{-q_i} \left( \left( b_i \wedge z_{i-1,-q_i-u} \right) \rightarrow z_{i,-u} \right) \equiv \bigwedge_{l=1}^{k-q_i} \left( (b_i \wedge z_{i-1,k-q_i-l}) \rightarrow z_{i,k-l} \right)$, where $l = u + k$. Finally, Eq. (13d) corresponds to Eq. (7d), as we have $[\![w_{i-1} < q_i + u]\!] \equiv 1$ since $q_i + u \geq q_i - q_i + 1 > 0$ (which is the upper bound of $w_{i-1}$).

*Proof (Theorem 3).*

The clauses for $x_i + y_{i-1} \leq y_i$ are:

$$\bigwedge_{v \in D(x_i)} \bigwedge_{d_{i-1,j} \in D(y_{i-1})} ([\![x_i \geq v]\!] \wedge [\![y_{i-1} \geq d_{i-1,j}]\!]) \rightarrow [\![y_i \geq d_{i-1,j} + v]\!]$$

Clearly, for every edge $((i-1, j), (i, j'), p) \in E$, there is a clause with antecedents $[\![x_i \geq v]\!]$ (where $v = pq_i$) and $[\![y_{i-1} \geq d_{i-1,j}]\!]$, which corresponds to

the clause from that same edge in the original encoding. It remains to be shown that the clause's consequent $[\![y_i \geq d_{i-1,j} + v]\!]$ matches the literal $[\![y_i \geq d_{i,j'}]\!]$ of the target node $(i, j')$. If $d_{i-1,j} = \max W_{i-1,j}$, and there is a $p$ edge $(i-1, j)$ and $(i, j')$, then $\max W_{i,j'-1} < d_{i,j} + v \leq \max W_{i,j'}$. By the domain construction of Eq. (11), we have $d_{i,j'-1} < w \leq d_{i,j'}$. Consequently, $[\![y_i \geq w]\!] \equiv [\![y_i \geq d_{i,j'}]\!]$ by Eq. (1b). Where $v = 0$, the clause simplifies to $[\![y_{i-1} \geq d_{i-1,j}]\!] \rightarrow [\![y_i \geq d_{i-1,j}]\!]$ by Eq. (1a).

*Proof (Lemma 1).* No assumptions are made on the domains, except that all are non-empty (otherwise the problem would be trivially unsatisfiable). However, we do know that the clause $([\![x \geq v]\!] \wedge [\![y \geq w]\!]) \rightarrow [\![z \geq v + w]\!]$ exists if and only if $v \in D(x)$ and $w \in D(y)$.

**Lower Bound Propagation** Given a fixed lower bound $v$ for $x$ and lower bound $w$ for $y$, then since these bounds are in $D(x)$ and $D(y)$ respectively, we have $[\![x \geq v]\!]$ and $[\![y \geq w]\!]$ hold. Hence, the clause $[\![x \geq v]\!] \wedge [\![y \geq w]\!] \rightarrow [\![z \geq v+w]\!]$ will propagate the correct bound (or cause failure if $v + w > ub(z)$).

**Upper Bound Propagation** Suppose we have a fixed upper bound $u$ for $z$, then we have $[\![z \leq u]\!] \equiv \neg[\![z \geq u']\!], d^z_i \leq u < u' = d^z_{i+1}$ for some adjacent domain values $d^z_i, d^z_{i+1} \subseteq D(z)$. So $\neg[\![z \geq u']\!]$ holds. By the IC assumption, $\neg[\![z \geq u'']\!], u'' > u'$ also holds.

Suppose we have a fixed lower bound $w \in D(y)$ for $y$ (which might be the original lower bound), then we know that $[\![y \geq w]\!]$ holds. Propagation should enforce that $x \leq u - w$. Now $[\![x \leq u - w]\!] \equiv \neg[\![x \geq u - w + 1]\!] \equiv \neg[\![x \geq v]\!], d^x_j < u - w + 1 \leq v = d^x_{j+1}$ for some adjacent domain values $d^x_i, d^x_{j+1} \subseteq D(x)$. Consider the clause $[\![x \geq v]\!] \wedge [\![y \geq w]\!] \rightarrow [\![z \geq v + w]\!]$. Since $v + w \geq (u - w + 1) + w = u + 1 \geq u'$, we have that $\neg[\![z \geq v + w]\!]$ holds by the IC assumption, and the clause propagates $\neg[\![x \geq v]\!]$. Note also, for any value $v' > v, v' \in D(x)$ we have the clause $[\![x \geq v']\!] \wedge [\![y \geq w]\!] \rightarrow [\![z \geq v' + w]\!]$, which propagates $\neg[\![x \geq v']\!]$ again by the IC assumption on $z$. This proves that upper bounds are correctly propagated, and any propagated upper bound also satisfies the IC assumption.

*Proof (Theorem 4).* First, domain consistency on linear inequality constraints is equivalent to bounds($\mathbb{R}$) consistency [15]. Next, the (non-zero) lower bounds on the leaf integers $x_i = q_i b_i$ are set by the fact that $[\![x_i \geq q_i]\!] \equiv b_i$. Lemma 1 shows that the lower bounds are correctly propagated up the tree. For the top most inequality $l(root) + r(root) \leq y_{root}$, the IC is satisfied automatically since $y_{root}$ has a singular domain, thus all literals mentioning $y_{root}$ are either true or false by definition. This means upper bounds propagate to the children of the root correctly and satisfy the IC. By induction, using Lemma 1 all upper bounds propagated by the tree are correct and always satisfy the IC. Finally, if an upper bound propagated on the leaf integer $x_i = q_i b_i$ removes value $q_i$, then it sets $\neg[\![x_i \geq q_i]\!] \equiv \neg b_i$.

*Proof (Theorem 5).* The fact that the combined encodings using Eq. (4) and Eq. (12) enforce bounds ($\mathbb{R}$) consistency of $\sum_{i=1}^{n} q_i b_i = k$, follows from its equivalence to bounds($\mathbb{R}$) consistency (or equivalently domain consistency) of the two inequalities $\sum_{i=1}^{n} q_i b_i \leq k$ and $\sum_{i=1}^{n} q_i b_i \geq k$ [19].