Encoding Linear Constraints into SAT

Ignasi Abío[1] and Peter J. Stuckey[1,2]

[1] NICTA Victoria Laboratory, Australia.
[2] Department of Computing and Information Systems,
The University of Melbourne

Abstract. Linear integer constraints are one of the most important constraints in combinatorial problems since they are commonly found in many practical applications. Typically, encoding linear constraints to SAT performs poorly in problems with these constraints in comparison to constraint programming (CP) or mixed integer programming (MIP) solvers. But some problems contain a mix of combinatoric constraints and linear constraints, where encoding to SAT is highly effective. In this paper we define new approaches to encoding linear constraints into SAT, by extending encoding methods for pseudo-Boolean constraints. Experimental results show that these methods are not only better than the state-of-the-art SAT encodings, but also improve on MIP and CP solvers on appropriate problems. Combining the new encoding with lazy decomposition, which during runtime only encodes constraints that are important to the solving process that occurs, gives a robust approach to many highly combinatorial problems involving linear constraints.

1 Introduction

In this paper we study linear integer (LI) constraints, that is, constraints of the form $a_1x_1 + \cdots + a_nx_n \ \# \ a_0$, where the a_i are integer given values, the x_i are finite-domain integer variables, and the relation operator $\#$ belongs to $\{<, >, \leqslant, \geqslant, =\}$. We will assume w.l.o.g that $\#$ is $\leqslant$, the a_i are positive and all the domains of the variables are $[0, d_i]$, since other cases can be reduced to this one.[1]

Linear integer constraints appear in many combinatorial problems such as scheduling, planning or software verification; they are also present in MaxSAT problems [8]; or are part of some MaxSAT techniques, as in *Fu & Malik algorithm* [17] (and some other algorithms based on it). Therefore, all approaches to combinatorial optimization have studied how to best handle them, including for MIP solvers, CP solvers [21], SMT solvers [15, 11], and SAT solvers [26, 6]

In this paper we examine how we can extend the state-of-the-art methods for SAT encoding of pseudo-Boolean (PB) constraints of the form $a_1x_1 + \cdots + a_nx_n \ \# \ a_0$ where x_i are Booleans, to the general linear integer case.

The method proposed here roughly consists in encoding the linear integers constraints into Reduced Ordered Multi-Decision Diagram (MDD for short), and

[1] Although replacing an equality by two inequalities substantially reduces propagation strength.

then decomposing the MDD to SAT. There are different reasons for choosing this approach: firstly, most state-of-the-art encoding methods define one auxiliary variable for every different possible value of the partial sum $s_i = a_1x_1 + a_2x_2 + \cdots + a_ix_i$. However, some values of the partial sums may be equivalent in the constraint. For instance, if a_j is even for every $j > i$, there is no difference between $s_i = a_0$ and $s_i = a_0 - 1$. With MDDs, due to the reduction process, we can identify these situations, and encode all these indistinguishable values with a single variable, producing a more compact encoding.

Secondly, BDDs are one of the best methods for encoding pseudo-Boolean constraints into SAT [3], and MDDs seem the natural tool to generalize the pseudo-Boolean encoding. Although the resulting encoding may be exponential; however, in real-world problems we have not found any exponential examples.

The goal of this encoding is not for use in arbitrary problems involving LI constraints. In fact, a specific linear integer (MIP) solver will probably outperform any SAT encoding in problems with more LI constraints than Boolean clauses.

Nevertheless, a fairly common kind of combinatorial problem mainly consist of Boolean variables and clauses, but also a few integer variables and LI constraints. Among these problems, an important class correspond to SAT problems with a linear integer objective function. In these cases, SAT solvers are the optimal tool for solving the problem, but a good encoding for the linear integer constraints is needed to make the optimization effective. Therefore, in these problems the decomposition presented here can make a significant difference.

Note, however, that decomposing the constraint may not always be the best option. In some cases the encoding might produce a large number of variables and clauses, transforming an easy problem for a CP solver into a huge SAT problem. In some other cases, nevertheless, the auxiliary variables may give an exponential reduction of the search space. Lazy decomposition [5, 4] is a hybrid approach that has been successfully used to handle this issue for cardinality and pseudo-Boolean constraints. Here, we show that it also can be applied successfully on LI constraints.

The method proposed here uses the order encoding for representing the integer variables. In some cases, however, the domains of the integer variables are too large for order encoding. We also propose a new alternative method for encoding linear integer constraints with the logarithmic encoding.

The contributions of this paper are:

- A new encoding (MDD) for LI constraints using MDDs that outperforms the state-of-the-art encodings.
- An alternative encoding (BDD-Dec) for LI constraints for large constraints or variables with huge domains.
- An improved encoding of PB constraints which share coefficients, by converting these constraints to LI constraints.
- A rigorous and extensive experimental comparison of our methods with respect to other decompositions to SAT and other solvers. A total of 9 methods are compared, over approximately 3000 benchmarks, both industrial and crafted.

2 Preliminaries

2.1 SAT Solving

Let $\mathcal{X} = \{x_1, x_2, \ldots\}$ be a fixed set of propositional *variables*. If $x \in \mathcal{X}$ then x and $\neg x$ are *positive* and *negative literals*, respectively. The *negation* of a literal l, written $\neg l$, denotes $\neg x$ if l is x, and x if l is $\neg x$. A *clause* is a disjunction of literals $\neg x_1 \vee \cdots \vee \neg x_p \vee x_{p+1} \vee \cdots \vee x_n$, sometimes written as $x_1 \wedge \cdots \wedge x_p \rightarrow x_{p+1} \vee \cdots \vee x_n$. A *CNF formula* is a conjunction of clauses.

A (partial) *assignment* A is a set of literals such that $\{x, \neg x\} \not\subseteq A$ for any $x \in \mathcal{X}$, i.e., no contradictory literals appear. A literal l is *true* in A if $l \in A$, is *false* in A if $\neg l \in A$, and is *undefined* in A otherwise. True, false or undefined is the *polarity* of the literal l. A clause C is true in A if at least one of its literals is true in A. A formula F is true in A if all its clauses are true in A. In that case, A is a *model* of F. Systems that decide whether a formula F has any model are called SAT-solvers, and the main inference rule they implement is *unit propagation*: given a CNF F and an assignment A, find a clause in F such that all its literals are false in A except one, say l, which is undefined, add l to A and repeat the process until reaching a fix-point. See e.g. [23] for more details.

2.2 LCG and LD Solvers

Many modern CP solvers, so called Lazy Clause Generation (LCG) solvers, include the ability to explain their propagation and generate nogoods just as in SAT solvers. Propagation of LI constraints is well understood [21] and standard. And adding explanation for LI constraints is also well understood [16], although there are often a number of choices of explanation that result.

More recently, Lazy Decomposition (LD) solvers have been proposed. An LD solver is a LCG solver that, when one complex constraint propagator is very active (that is, is frequently asked to generate explanations), then the solver replaces the propagator by either partially or totally decomposing the constraint into SAT (see [5, 4] for more details). The advantage of LD solvers is that the exposure of intermediate variables in the SAT encodings can substantially benefit search, but it avoids the up front cost of encoding all complex constraints, only those that are important in the solving process.

2.3 Order and Logarithmic Encoding

There are different methods for encoding integer variables into SAT (see for instance [27, 18]). In this paper we use the order and the logarithmic encoding.

Let y be an integer variable with domain $[0, d]$. The *order encoding* [19, 7] (sometimes called *ladder* or *regular*) introduces Boolean variables y^i for $0 \leqslant i < d$. A variable y^i is true iff $y \leqslant i$. The encoding also introduces the clauses $y^i \rightarrow y^{i+1}$ for $0 \leqslant i < d - 1$.

The *logarithmic encoding* introduces only $\log d$ variables y_b^i which codify the binary representation of the value of y, as $y = \sum_{i=0}^{\lfloor log(d) \rfloor} 2^i y_b^i$. It is a more compact encoding, but it usually gives poor propagation performance.

2.4 Multi Decision Diagrams

A directed acyclic graph is called an *ordered Multi Decision Diagram* if it satisfies the following properties:

- It has two terminal nodes, namely $\mathcal{T}$ (true) and $\mathcal{F}$ (false).
- Each non-terminal node is labeled by an integer variable $\{x_1, x_2, \cdots, x_n\}$. This variable is called *selector variable*.
- Every node labeled by x_i has the same number of outgoing edges, namely $d_i + 1$.
- If an edge connects a node with a selector variable x_i and a node with a selector variable x_j, then $j > i$.

The MDD is *quasi-reduced* if no isomorphic subgraphs exist. It is *reduced* if, moreover, no nodes with only one child exist. A *long edge* is an edge connecting two nodes with selector variables x_i and x_j such that $j > i + 1$. In the following we only consider quasi-reduced ordered MDDs without long edges, and we just refer to them as MDDs for simplicity.

An MDD represents a function

$$f : \{0, 1, \ldots, d_1\} \times \{0, 1, \ldots, d_2\} \times \cdots \times \{0, 1, \ldots, d_n\} \to \{0, 1\}$$

in the obvious way. Moreover, given the variable ordering, there is only one MDD representing that function. We refer to [25] for further details about MDDs.

3 Linear Integer Constraints

In this paper we consider linear integer constraints of the form $a_1x_1 + \cdots + a_nx_n \leqslant a_0$, where the a_i are positive integer coefficients and the x_i are integer variables with domains $[0, d_i]$. Other LI constraints can be easily reduced to this one:

$$a_1x_1 + \cdots + a_nx_n \;=\; a_0 \quad\Longrightarrow\quad \begin{cases} a_1x_1 + \cdots + a_nx_n \;\leqslant\; a_0 \;\wedge \\ a_1x_1 + \cdots + a_nx_n \;\geqslant\; a_0 \end{cases}$$

$$a_1x_1 + \cdots + a_nx_n \;<\; a_0 \quad\Longrightarrow\quad a_1x_1 + \cdots + a_nx_n \;\leqslant\; a_0 - 1$$

$$a_1x_1 + \cdots + a_nx_n \;\geqslant\; a_0 \quad\Longrightarrow\quad -a_1x_1 + \cdots + -a_nx_n \;\leqslant\; -a_0$$

$$a_1x_1 + \cdots + a_nx_n \;>\; a_0 \quad\Longrightarrow\quad -a_1x_1 + \cdots + -a_nx_n \;\leqslant\; -a_0 - 1$$

$$\left.\begin{array}{l} a_1x_1 + \cdots + a_ix_i + \cdots \\ +a_nx_n \;\leqslant\; a_0 \\ \text{when } x_i \in [l, u], l \neq 0, a_i > 0 \end{array}\right\} \quad\Longrightarrow\quad \begin{cases} a_1x_1 + \cdots + a_ix'_i + \cdots \\ +a_nx_n \;\leqslant\; a_0 + a_i \times l \;\wedge \\ x'_i \in [0, u - l] \;\wedge\; x'_i = x_i - l \end{cases}$$

$$\left.\begin{array}{l} a_1x_1 + \cdots + a_ix_i + \cdots \\ +a_nx_n \;\leqslant\; a_0 \\ \text{when } a_i < 0 \text{ and } x_i \in [l, u] \end{array}\right\} \quad\Longrightarrow\quad \begin{cases} a_1x_1 + \cdots + -a_ix'_i + \cdots \\ +a_nx_n \;\leqslant\; a_0 - a_i \times u \;\wedge \\ x'_i \in [0, u - l] \;\wedge\; x'_i = u - x_i \end{cases}$$

$$\left.\begin{array}{l} x_i \in [l, u],\; l \neq 0 \;\wedge\; x'_i = x_i - l \\ \wedge\; x_i^j \equiv x_i \leqslant j \;\; \text{for } l \leqslant j < u \end{array}\right\} \quad\Longrightarrow\quad x_i^{j-l} \equiv x'_i \leqslant j \;\; \text{for } 0 \leqslant j < u - l$$

$$\left.\begin{array}{l} x_i \in [l,u] \ \wedge \ x'_i = u - x_i \\ \wedge \ x_i^j \equiv x_i \leqslant j \ \text{ for } l \leqslant j < u \end{array}\right\} \quad \Longrightarrow \quad \neg x_i^{u-j-1} \equiv x'_i \leqslant j \ \text{ for } 0 \leqslant j < u-l$$

The goal of this paper is to find a SAT encoding for a given LI constraint. That is, given a LI constraint C, construct an equivalent formula F such that any model for F restricted to the variables of C is a model of C. Two extra properties are usually sought:

- *consistency checking by unit propagation* or simply *consistency*: whenever a partial assignment A cannot be extended to a model for C, unit propagation on F and A produces a contradiction (a literal l and its negation $\neg l$);
- *domain consistency* (again by unit propagation): given an assignment A that can be extended to a model of C, but such that $A \cup \{x\}$ cannot, unit propagation on F and A produces $\neg x$.

4 Construction of the MDD

In this section we describe an efficient method for building MDDs. Let us fix a LI constraint $a_1x_1 + \cdots + a_nx_n \leqslant a_0$ and a variable ordering $[x_1, x_2, \ldots, x_n]$. Before explaining the algorithm, we need a preliminary definition.

Let $\mathcal{M}$ be the MDD of the given LI constraint and let ν be a node of $\mathcal{M}$ with selector variable x_i. We define the *interval* of ν as the set of values α such that the MDD rooted at ν represents the LI constraint $a_ix_i + \cdots + a_nx_n \leqslant \alpha$. It is easy to see that this definition corresponds in fact to an interval.

Example 1. Figure 1 contains the MDD of $3x_1+2x_2+5x_3 \leqslant 15$, where $x_1 \in [0,4]$, $x_2 \in [0,2]$ and $x_3 \in [0,3]$. The root interval is $[15,15]$: this means that the root does not correspond to any constraint $3x_1 + 2x_2 + 5x_3 \leqslant \alpha$, apart from $\alpha = 15$. This means that this constraint is not equivalent to $3x_1 + 2x_2 + 5x_3 \leqslant 14$ or $3x_1+2x_2+5x_3 \leqslant 16$. However, the left node with selector variable x_2 has interval $[15,16]$. This means that $2x_2+5x_3 \leqslant 15$ and $2x_2+5x_3 \leqslant 16$ are both represented by the MDD rooted at that node. In particular, that means that $2x_2 + 5x_3 \leqslant 15$ and $2x_2 + 5x_3 \leqslant 16$ are two equivalent constraints. □

The next proposition shows how to compute the intervals of every node:

Proposition 1 *Let $\mathcal{M}$ be the MDD of a LI constraint $a_1x_1 + \cdots + a_nx_n \leqslant a_0$. Then, the following holds:*

- *The interval of the true node $\mathcal{T}$ is $[0,\infty)$.*
- *The interval of the false node $\mathcal{F}$ is $(-\infty,-1]$.*
- *Let ν be a node with selector variable x_i and children $\{\nu_0, \nu_1, \ldots, \nu_{d_i}\}$. Let $[\beta_j, \gamma_j]$ be the interval of ν_j. Then, the interval of ν is $[\beta, \gamma]$, with*

$$\beta = \max\{\beta_r + ra_i \mid 0 \leqslant r \leqslant d_i\}, \qquad \gamma = \min\{\gamma_r + ra_i \mid 0 \leqslant r \leqslant d_i\}.$$

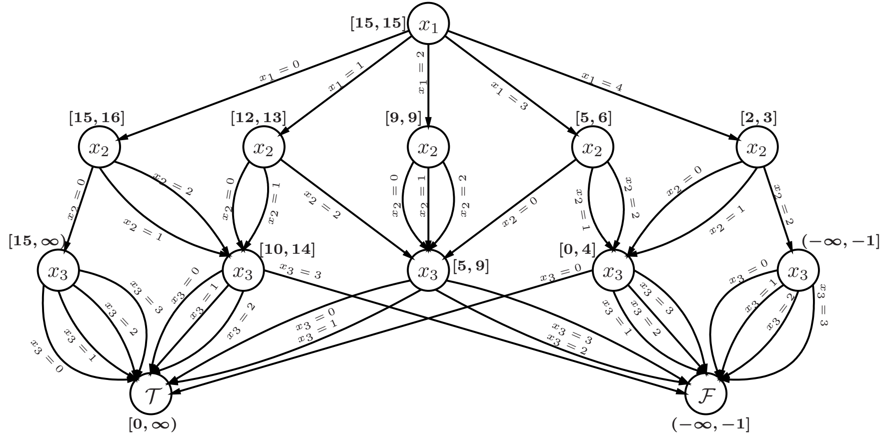

Fig. 1. MDD of $3x_1 + 2x_2 + 5x_3 \leqslant 15$.

The proof of this proposition is very similar to the Proposition 7 of [3].

Example 2. Again, let us consider the constraint $3x_1 + 2x_2 + 5x_3 \leqslant 15$, whose MDD is represented at Figure 1. By the previous Proposition, $\mathcal{T}$ and $\mathcal{F}$ have, respectively, intervals $[0, \infty)$ and $(-\infty, -1]$. Applying again the same proposition, we can compute the intervals of the nodes having x_3 as selector variable. For instance, the interval from the left node is

$$[0, \infty) \cap [5, \infty) \cap [10, \infty) \cap [15, \infty) = [15, \infty),$$

and the interval from the node having selector variable x_3 in the middle is

$$[0, \infty) \cap [5, \infty) \cap (-\infty, 9] \cap (-\infty, 14] = [5, 9].$$

After computing all the intervals from the nodes with selector variable x_3, we can compute the intervals of the nodes with selector variables x_2 in the same way, and, after that, we can compute the interval of the root. □

The key point of the MDDCreate algorithm, detailed in Algorithm 1 and Algorithm 2, is to label each node of the MDD with its interval $[\beta, \gamma]$.

In the following, for every $i \in \{1, 2, \ldots, n+1\}$, we use a set L_i consisting of pairs $([\beta, \gamma], \mathcal{M})$, where $\mathcal{M}$ is the MDD of the constraint $a_i x_i + \cdots + a_n x_n \leqslant a_0'$ for every $a_0' \in [\beta, \gamma]$ (i.e., $[\beta, \gamma]$ is the interval of $\mathcal{M}$). All these sets are kept in a tuple $\mathcal{L} = (L_1, L_2, \ldots, L_{n+1})$.

Note that by definition of the MDD's intervals, if both $([\beta_1, \gamma_1], \mathcal{M}_1)$ and $([\beta_2, \gamma_2], \mathcal{M}_2)$ belong to L_i then either $[\beta_1, \gamma_1] = [\beta_2, \gamma_2]$ or $[\beta_1, \gamma_1] \cap [\beta_2, \gamma_2] = \emptyset$. Moreover, the first case holds if and only if $\mathcal{M}_1 = \mathcal{M}_2$. Therefore, L_i can be represented with a *binary search tree-like* data structure, where insertions and searches can be done in logarithmic time. The function **search**(K, L_i) searches

Algorithm 1 Procedure MDDCreate

Require: Constraint $C : a_1x_1 + \cdots + a_nx_n \leqslant a_0$
Ensure: returns $\mathcal{M}$ the MDD of C.
1: **for all** i such that $1 \le i \le n$ **do**
2: $L_i \leftarrow \emptyset$.
3: **end for**
4: $L_{n+1} \leftarrow \Big\{ \big((-\infty, -1], \mathcal{F}\big), \ \big([0, \infty), \mathcal{T}\big) \Big\}$.
5: $\mathcal{L} \leftarrow (L_1, \ldots, L_{n+1})$.
6: $([\beta, \gamma], \mathcal{M}) \leftarrow$ **MDDConstruction**$(1, a_1x_1 + \cdots + a_nx_n \leqslant a_0, \mathcal{L})$.
7: **return** $\mathcal{M}$.

Algorithm 2 Procedure MDDConstruction

Require: $i \in \{1, 2, \ldots, n+1\}$, constraint $C : a_ix_i + \cdots + a_nx_n \leqslant a'_0$ and tuple $\mathcal{L}$
Ensure: returns $[\beta, \gamma]$ interval of C and $\mathcal{M}$ its MDD
1: $([\beta, \gamma], \mathcal{M}) \leftarrow$ **search**(a'_0, L_i).
2: **if** $[\beta, \gamma] \neq \emptyset$ **then**
3: **return** $([\beta, \gamma], \mathcal{M})$.
4: **else**
5: **for all** j such that $0 \le j \le d_i$ **do**
6: $([\beta_j, \gamma_j], \mathcal{M}_j) \leftarrow$ **MDDConstruction**$(i+1, a_{i+1}x_{i+1} + \cdots + a_nx_n \leqslant a'_0 - ja_i, \mathcal{L})$.
7: **end for**
8: $\mathcal{M} \leftarrow$ **mdd**$(x_i, [\mathcal{M}_0, \ldots, \mathcal{M}_{d_i}])$.
9: $[\beta, \gamma] \leftarrow [\beta_0, \gamma_0] \cap [\beta_1 + a_1, \gamma_1 + a_1] \cap \cdots \cap [\beta_{d_i} + d_ia_i, \gamma_{d_i} + d_ia_1]$.
10: **insert**$(([\beta, \gamma], \mathcal{M}), L_i)$.
11: **return** $([\beta, \gamma], \mathcal{M})$.
12: **end if**

whether there exists a pair $([\beta, \gamma], \mathcal{M}) \in L_i$ with $K \in [\beta, \gamma]$. Such a tuple is returned if it exists, otherwise an empty interval is returned in the first component of the pair. Similarly, we also use function **insert**$(([\beta, \gamma], \mathcal{M}), L_i)$ for insertions. The size of the MDD in the worst case is $O(na_0)$ (exponential in the size of the rhs coefficient) and algorithm complexity is $O(nw \log w)$ where w is the maximum width of the MDD ($w \le a_0$).

5 Encoding MDDs into CNF

In this section we generalize the encoding for monotonic BDDs described in [3] to monotonic MDDs. The encoding assumes that the selector variables are encoded with the ladder encoding.

Let $\mathcal{M}$ be an MDD with the variable ordering $[x_1, \ldots, x_n]$. Let $[0, d_i]$ be the domain of the i-th variable, and let $\{x_i^0, \ldots, x_i^{d_i-1}\}$ be the variables of the ladder encoding of x_i (i.e., x_i^j is true iff $x_i \leqslant j$). Let μ be the root of $\mathcal{M}$, and let $\mathcal{T}$ and $\mathcal{F}$ be respectively the true and false terminal nodes. In the following, given a non-terminal node ν of $\mathcal{M}$, we define SelVar(ν) as the selector variable of ν, and Child(ν, j) as the j-th child of ν.

The encoding introduces the variables $\{z_\nu \mid \nu \in \mathcal{M}\}$; and the clauses

$$\{z_\mu,\ z_\mathcal{T},\ \neg z_\mathcal{F}\} \cup \Big\{ \neg z_\nu \vee x_i^{j-1} \vee z_{\nu'} \mid \nu \in \mathcal{M} \setminus \{\mathcal{T}, \mathcal{F}\}, \\ \text{SelVar}(\nu) = x_i,\ 0 \leqslant j \leqslant d_i,\ \nu' = \text{Child}(\nu, j) \Big\},$$

where x_i^{-1} is a dummy false variable.

Notice that this encoding coincides with the BDD encoding of [3] if the MDD is a BDD.

Theorem 2 *Unit propagation on the encoding described above enforces domain consistency (and hence also consistency).* □

The proof is very similar to the BDD case described in [3].

Example 3. Let us consider the MDD represented in Figure 1. The encoding introduces the variables $z_1, z_2, \ldots, z_{11}, z_\mathcal{T}, z_\mathcal{F}$, one for each node of the MDD; and the following clauses:

$$\begin{array}{llll}
z_1, & z_\mathcal{T}, & \neg z_\mathcal{F}, & \neg z_1 \vee z_2, \\
\neg z_1 \vee x_1 \leqslant 0 \vee z_3, & \neg z_1 \vee x_1 \leqslant 1 \vee z_4, & \neg z_1 \vee x_1 \leqslant 2 \vee z_5, & \neg z_1 \vee x_1 \leqslant 3 \vee z_6, \\
\neg z_2 \vee z_7, & \neg z_2 \vee x_2 \leqslant 0 \vee z_8, & \neg z_2 \vee x_2 \leqslant 1 \vee z_8, & \neg z_3 \vee z_8, \\
\neg z_3 \vee x_2 \leqslant 0 \vee z_8, & \neg z_3 \vee x_2 \leqslant 1 \vee z_9, & \neg z_4 \vee z_9, & \neg z_4 \vee x_2 \leqslant 0 \vee z_9, \\
\neg z_4 \vee x_2 \leqslant 1 \vee z_9, & \neg z_5 \vee z_9, & \neg z_5 \vee x_2 \leqslant 0 \vee z_{10}, & \neg z_5 \vee x_2 \leqslant 1 \vee z_{10}, \\
\neg z_6 \vee z_{10}, & \neg z_6 \vee x_2 \leqslant 0 \vee z_{10}, & \neg z_6 \vee x_2 \leqslant 1 \vee z_{11}, & \neg z_7 \vee z_\mathcal{T}, \\
\neg z_7 \vee x_3 \leqslant 0 \vee z_\mathcal{T}, & \neg z_7 \vee x_3 \leqslant 1 \vee z_\mathcal{T}, & \neg z_7 \vee x_3 \leqslant 2 \vee z_\mathcal{T}, & \neg z_8 \vee z_\mathcal{T}, \\
\neg z_8 \vee x_3 \leqslant 0 \vee z_\mathcal{T}, & \neg z_8 \vee x_3 \leqslant 1 \vee z_\mathcal{T}, & \neg z_8 \vee x_3 \leqslant 2 \vee z_\mathcal{F}, & \neg z_9 \vee z_\mathcal{T}, \\
\neg z_9 \vee x_3 \leqslant 0 \vee z_\mathcal{T}, & \neg z_9 \vee x_3 \leqslant 1 \vee z_\mathcal{F}, & \neg z_9 \vee x_3 \leqslant 2 \vee z_\mathcal{F}, & \neg z_{10} \vee z_\mathcal{T}, \\
\neg z_{10} \vee x_3 \leqslant 0 \vee z_\mathcal{F}, & \neg z_{10} \vee x_3 \leqslant 1 \vee z_\mathcal{F}, & \neg z_{10} \vee x_3 \leqslant 2 \vee z_\mathcal{F}, & \neg z_{11} \vee z_\mathcal{F}, \\
\neg z_{11} \vee x_3 \leqslant 0 \vee z_\mathcal{F}, & \neg z_{11} \vee x_3 \leqslant 1 \vee z_\mathcal{F}, & \neg z_{11} \vee x_3 \leqslant 2 \vee z_\mathcal{F}. &
\end{array}$$

Notice that some clauses are redundant. This issue is handled in Section 7.2. □

6 Optimization Problems

In this section we describe how to deal with combinatorial problem where we minimize a linear integer optimization function. A similar idea is used in [14], where the authors use BDDs for encoding problems with pseudo-Boolean objectives. Combinatorial optimization problems can be efficiently solved with a branch-and-bound strategy. In this way, all the lemmas learned in the previous steps are reused for finding the next solutions or proving the optimality. For implementing a branch-and-bound, we need to be able to create a decomposition of the constraint $a_1x_1 + \cdots + a_nx_n \leqslant a'_0$ from the decomposition of $a_1x_1 + \cdots + a_nx_n \leqslant a_0$ where $a'_0 < a_0$.

This is easy for cardinality constraints, since, when we have encoded a constraint $x_1 + \cdots + x_n \leqslant a_0$ with a sorting network, we can encode $x_1 + \cdots + x_n \leqslant a'_0$ by adding a single clause (see [9]).

Algorithm 3 MDD Construction: Optimization version

Require: Constraint $C : a_1x_1 + \cdots + a_nx_n \leqslant a'_0$ and tuple $\mathcal{L}$.
Ensure: returns $\mathcal{M}$ the MDD of C.
1: $([\beta, \gamma], \mathcal{M}) \leftarrow$ **MDDConstruction**$(1, a_1x_1 + \cdots + a_nx_n \leqslant a'_0, \mathcal{L})$.
2: **return** $\mathcal{M}$.

In order to reuse the previous encodings for the MDD encoding of an LI constraint, we have to save the tuple $\mathcal{L}$ used in Algorithm 1. When a new solution of cost $a'_0 + 1$ is found, Algorithm 3 is called.

Notice that the encoding creates at most one variable for every element of $L_i \in \mathcal{L}$, $1 \leqslant i \leqslant n$. Therefore, after finding optimality, the encoding has generated at most na_0 variables in total, where a_0 is the cost of the first solution found. The number of clauses generated can be bounded by na_0d, where $d = \max\{d_i\}$.

7 Improvements

In this section we describe some improvements of the method. The first improvement is to reorder the constraint such that $a_1 \geqslant a_2 \geqslant \cdots \geqslant a_n$. The MDD obtained in this way is usually smaller.

7.1 Grouping Identical Coefficients

Let us fix the LI constraint $C : a_1x_1 + \cdots + a_nx_n \leqslant a_0$. Assume that some coefficients are equal; for simplicity, let us assume $a_1 = a_2 = \cdots = a_r$. In this case, we can define the integer variable $s = x_1 + \cdots + x_r$ and decompose the constraint $C' : a_1s + a_{r+1}x_{r+1} + \cdots + a_nx_n \leqslant a_0$ instead of C. The domain of s is $[0, d_s]$ with $d_s = \min\{a_0/a_1, d_1 + \cdots + d_r\}$.

Notice that we do not need to encode the constraint $s = x_1 + \cdots + x_r$ defining the integer variables s, instead we can encode $c \equiv s \geqslant x_1 + \cdots + x_r$ since we are only interested in lower bounds. The encoding of c can be done with cardinality networks [2], which usually gives a more compact encoding than the MDD of c.

In industrial problems where constraints are not randomly generated, the coefficients have some meaning. It may be likely, that a large LI constraint has only a few different coefficients. In this case this technique can be very effective.

7.2 Removing Subsumed Clauses

The encoding explained at Section 5 can easily be improved by removing some unnecessary clauses. We apply the following rule when producing the encoding:

Given a non-terminal node ν with $\mathrm{SelVar}(\nu) = x_i$, if $\mathrm{Child}(\nu, j) = \mathrm{Child}(\nu, j-1)$, then the clause $\neg z_z\nu \vee x_i^{j-1} \vee z_{\nu'}$ is subsumed by the clause $\neg z_\nu \vee x_i^{j-2} \vee z_{\nu'}$; therefore, we can remove it.

Additionally, we also improve the encoding by reinstating long edges (since the dummy nodes used to eliminate long edges do not provide any information); that is, we encode the reduced MDD instead of the quasi-reduced MDD.

7.3 Solution Phase Saving

In decision problems, last phase saving described in [24] has proven to be a very effective strategy. According to this scheme, when the SAT solver makes a decision, the variable is chosen with the same polarity as in the last assignment.

However, in optimization problems this is not the best option. As seen in [1], a better strategy is to take the polarity that minimizes the objective function in the variables which directly appear in the objective function, or the polarity in the last solution in the other variables. That method, called *solution phase saving*, emulates a local search: after finding a solution, the method explores the neighbourhood of the solution in order to find a better solution nearby.

7.4 Lazy Decomposition

Lazy decomposition [5, 4] has proved to be very successful for handling cardinality and pseudo-Boolean constraints. Lazy decomposition for LI constraints implements each LI constraint as a propagator initially, and later when we see that a constraint is generating many explanations we replace the propagator fully or partially by a SAT decomposition. We use the approach of [4] which fully replaces a propagator. Our strategy for when to decompose an LI constraint is: when the constraint has generated more explanations than half its encoding size, or generated more than 50,000 explanations; except that we never encode LI constraints whose encoding size is ≥ 50 million clauses.

8 Related Work and Extensions

The simplest decomposition of linear integer constraints to SAT uses binary adders (Adder) [28]. The encoding is very compact, but it has a poor performance in practice since information does not propagate effectively through the encoding.

Decision diagrams methods have been widely used to handle LI constraints. The best current method [10] (BDD) uses a logarithmic encoding of the coefficients to create a BDD of the constraint, sorting the variables in a clever way. The encoding size is reduced to $O(n \log d \sum a_i)$, which is polynomial in the domain size but exponential in the coefficient size. We can improve this encoding if we also decompose the coefficients as is done in [3]: in this way, the encoding size is $O(n^2 \log d \log a_m)$, where a_m is the largest coefficient. We call this BDD-Dec.

Example 4. Consider the LI constraint $3x_1+2x_2+5x_3 \leqslant 15$ from Example 1. After encoding the integer variables with the logarithmic encoding, the constraint becomes the pseudo-Boolean $3x^0_{b,1} + 6x^1_{b,1} + 12x^2_{b,1} + 2x^0_{b,2} + 4x^1_{b,2} + 5x^0_{b,3} + 10x^1_{b,3} \leqslant 15$. Bartzis and Bultan [10] construct the BDD of the pseudo-Boolean $2x^0_{b,2} + 3x^0_{b,1} + 4x^1_{b,2} + 5x^0_{b,3} + 6x^1_{b,1} + 10x^1_{b,3} + 12x^2_{b,1} \leqslant 15$. Our method decomposes the coefficients (i.e., considers $x^0_{b,1} + 2x^0_{b,1}$ instead of $3x^0_{b,1}$) and builds the resulting BDD; so we encode the constraint $x^0_{b,1} + x^0_{b,3} + 2x^0_{b,2} + 2x^0_{b,1} + 2x^1_{b,1} + 2x^1_{b,3} + 4x^1_{b,2} + 4x^0_{b,3} + 4x^1_{b,1} + 4x^2_{b,1} + 8x^1_{b,3} + 8x^2_{b,1} \leqslant 15$. □

Formally, the BDD-Dec method encodes LI constraint $a_1x_1 + \cdots + a_nx_n \leqslant a_0$ with $x_i \in [0, d_i], 1 \leq i \leq n$ by first creating the PB constraint

$$\sum_{i=1}^{n} \sum_{j \in 0..\lfloor \log_2 d_i \rfloor, (d_i \div 2^j) \bmod 2 = 1} \sum_{k \in 0..\lfloor \log_2 a_i \rfloor, (a_i \div 2^k) \bmod 2 = 1} 2^{j+k} \times x_{b,i}^{j} \leq a_0$$

over the logarithmic encoding variables x_b and encoding this using the state-of-the-art encoding for PB constraints given in [3].

Note however, logarithmic encoding of integers, while compact, is usually a bad option since it significantly hampers propagation of information, leading to poor solving performance. Neither BDD or BDD-Dec enforce consistency.

The most similar encoding to the approach we define is the *support encoding* (Support) [26, 6]. While the encodings both effectively define auxiliary variables for the values of the partial sums $s_i = a_1x_1 + \cdots + a_ix_i$, the support encoding fails to identify which values of these partial sums are indistinguishable in the constraint. The result is to create an encoding equivalent to a non-reduced ordered MDD. If the MDD cannot be reduced further (for instance, if all the coefficients are 1), the two encodings would be identical (ignoring the further improvement discussed above). In general, however, the support encoding generates redundant variables and clauses. Another important improvement in our encoding is to group identical coefficients (see Section 7.1).

9 Experimental Results

In this section we compare our encoding with other LI constraints encodings and specific LI solvers. Unfortunately, we have not found in the literature a rigorous comparison of the different approaches for solving SAT+MIP problems. Here, we consider state-of-the-art methods from different areas and a huge set of benchmarks (about 2,900) coming from different industrial and crafted families.

We do not expect to be the best method in all the families of the problems. The main goals of this section are to:

- Detect the type of problems where it is worthwhile to encode an LI constraint instead of using a specific solver.
- Decide, in these problems, which encoding is better.
- Evaluate the lazy decomposition approach with different encodings.

All experiments were performed in a 2x2GHz Intel Quad Core Xeon E5405, with 2x6MB of Cache and 16 GB of RAM. All the benchmarks we used can be found at `www.cs.mu.oz.au/~pjs/encodeli/` in MiniZinc [22] format with scripts to generate CNF format.

We compare our new encodings MDD and BDD-Dec with those from the literature Adder, BDD and Support. We also consider the Gurobi mixed integer programming solver [20] (Gurobi) and the Barcelogic SMT solver [13] (LCG). We also consider the use of lazy decomposition [4] together with the two domain-consistent encoding approaches: using the MDD decomposition explained here (LD-MDD), and using the support encoding (LD-Sup).

	Different values of n						Different values of $a_{\max}$						Different values of d					
	5	10	20	40	80	160	1	2	4	8	16	32	1	2	4	10	25	100
Adder	0.05	9.55	186	276	296	300	57.3	60.4	74.9	114	105	117	0.02	0.15	1.84	32.7	80.3	215
BDD	0.04	7.11	185	272	298	300	21.1	39.2	59.1	111	110	129	**0.01**	0.06	0.58	26.8	77.6	220
BDD-Dec	0.12	4.73	163	269	295	300	10.5	25.6	47.7	90.9	84.8	82.5	**0.01**	0.13	0.46	18.6	56.3	202
MDD	0.05	6.45	175	268	290	300	51.8	57.2	62.9	103	107	119	**0.01**	0.03	0.17	18.9	80.6	258
Support	0.12	16.0	197	278	300	300	53.9	78.6	90.3	142	133	145	0.02	0.07	0.57	32.8	108	272
LD-MDD	**0.01**	3.23	165	264	287	300	44.3	48.2	59.4	90.0	91.5	91.1	0.02	0.01	0.09	16.4	63.1	244
LD-Sup	**0.01**	8.44	179	270	288	300	50.3	68.7	76.6	115	108	110	0.02	0.01	0.19	21.9	82.0	254
LCG	**0.01**	4.87	173	265	288	300	117	97.2	88.0	118	94.0	70.6	0.02	0.01	0.13	22.9	75.3	242
Gurobi	**0.01**	**0.09**	**0.02**	**0.03**	**0.02**	**0.03**	**0.01**	**0.01**	**0.01**	**0.01**	**0.02**	**0.02**	**0.01**	**0.02**	**0.01**	**0.01**	**0.02**	**0.01**

Table 1. Multiple knapsack solving average time.

The Barcelogic SAT solver was used for all the SAT-based methods; this ensured that the lazy decomposition approaches were implemented using the same solver. The solution phase saving policy (see Section 7.3) was used in all SAT-based methods. Gurobi used its default settings.

9.1 Multiple Knapsack

First we consider the classic multiple knapsack problem.

$$\begin{array}{llllll} \text{Max} & a_1^0 x_1 & + a_2^0 x_2 & + \cdots + a_n^0 x_n & & \text{such that} \\ & a_1^1 x_1 & + a_2^1 x_2 & + \cdots + a_n^1 x_n & \leqslant a_0^1 & \\ & \ldots & & & & \\ & a_1^m x_1 & + a_2^m x_2 & + \cdots + a_n^m x_n & \leqslant a_0^m, & \end{array}$$

where x_i are integer variables with domain $[0, d]$ and the coefficients belong to $[0, a_{\max}]$.

Since it only consists of linear integer constraints it is ideal for MIP solvers. We consider this problem since it is easy to modify the parameters of the constraints, and, therefore, we can easily compare the encodings in different situations. More precisely, we have considered different constraint sizes, coefficient sizes and domain sizes. In these problems, n is the number of variables, $m = 20$ is the number of LI constraints, $d + 1$ is the domain size of the variables; and $a_{\max}$ is the bound of the coefficients.

Table 1 contains the results on these benchmarks. For each parameter configuration, 100 benchmarks are considered and the average time for solving them is reported. Timeout are considered as 300s response in the average computation. In columns 2-7 $m = 20$, $d = 20$, $a_{\max} = 10$ and different values of n are taken. Columns 8-13 consider different values of $a_{\max}$, with $m = 20$, $n = 15$ and $d = 20$. In columns 14-19 $n = 15$, $m = 20$ and $a_{\max} = 10$, with different values of d. For each group of problems, the best encoding is underlined and the best method is bolded.

As expected Gurobi is by far the best method. SAT-based methods do not compete, however, we can effectively compare the encodings in different situations. In general, BDD-Dec and MDD are the best encodings, MDD is specially efficient if the domains are small and BDD-Dec in large ones. Also, notice that lazy decomposition performs, in general, better than both the decomposition and the propagator approaches.

	15s	60s	300s	900s	3600s
Adder	0.905	0.963	<u>0.986</u>	<u>0.992</u>	<u>0.996</u>
BDD	0.784	0.859	0.928	0.957	0.977
BDD-Dec	<u>**0.929**</u>	<u>0.967</u>	0.985	0.99	0.994
MDD	0.727	0.75	0.858	0.889	0.899
Support	0.727	0.774	0.861	0.872	0.876
LD-MDD	0.918	**0.982**	0.992	0.994	0.996
LD-Sup	0.918	0.98	0.991	0.993	0.994
LCG	0.92	0.981	**0.993**	**0.995**	**0.997**
Gurobi	0.598	0.618	0.647	0.671	0.721

Table 2. Average quality from 2040 RCPSP benchmarks.

9.2 RCPSP

Resource-constrained project scheduling problem [12] (RCPSP) is possibly the most studied scheduling problem. It consists of tasks consuming one or more resources, precedences between some tasks, and resources. Here we consider the case of non-preemptive tasks and renewable resources with a constant resource capacity over the planning horizon. A solution is a schedule of all tasks so that all precedences and resource constraints are satisfied.

Usually, the objective of RCPSP is to find a solution minimizing the makespan. Here, however, the objective is to minimize a weighted sum of start times, i.e., minimize $\sum w_i s_i$, where w_i is the weight of the i-th task and s_i is its starting time. These weights represent the importance of the tasks: usually, a company not only needs to finish all the tasks in the minimum time, but also wants to give more importance to some of them. For the examples considered here, only ten tasks have a non-zero weight, but terminal tasks (this is, tasks with no successors with respect the precedence constraints) are never given a zero weight. Here we have considered the 600 RCPSP problems with 120 tasks (ie, the largest ones).

The results are summarized in Table 2. Columns contain the *quality* average after X seconds. Quality is computed by dividing the cost of the best known solution by the cost of the current solution of the method; therefore, quality $= 0$ if no solution has been found, and quality $= 1$ when the best solution has been found. Again, the best method is bolded and the best encoding is underlined.

Since in these benchmarks, the variables' domains are very large (frequently $d > 200$); the logarithmic encodings Adder and BDD-Dec are the best encodings. MDD and Support have a similar performance, they are far from the best methods. However, the best method is LCG. Both lazy decomposition methods perform almost identically to LCG. It is clear (and well known) that MIP is not competitive on RCPSP problems.

9.3 Graph Coloring

The classical graph coloring problem consists in, given a graph, assign to each node a color $\{0, 1, \ldots, c-1\}$ such that two nodes connected by an edge have different colors. Usually, the problem consists in finding a solution that minimizes the number of colors (i.e., c). In this section we have considered a variant of this problem. Let us consider a graph that can be colored with c colors: For each node ν of the graph, let us define an integer value a_ν. Now, we want to color the

	15s	60s	300s	900s	3600s
Adder	0.421	0.468	0.511	0.527	0.546
BDD	0.404	0.444	0.483	0.497	0.513
BDD-Dec	0.417	0.462	0.499	0.512	0.53
MDD	0.615	**0.624**	**0.644**	**0.651**	**0.657**
Support	0.605	0.615	0.636	0.641	0.648
LD-MDD	0.616	0.621	0.64	0.642	0.648
LD-Sup	0.613	0.618	0.635	0.639	0.643
LCG	**0.617**	0.623	0.64	0.643	0.646
Gurobi	0.443	0.45	0.452	0.453	0.454

Table 3. Average quality from 320 graph coloring benchmarks.

graph with c colors $\{0, 1, \ldots, c-1\}$ minimizing the function $\sum a_\nu x_\nu$, where x_ν is the color of the node ν.

We have considered the 80 graph coloring instances from `http://mat.gsia.cmu.edu/COLOR08/` that have less than 500 nodes. For each graph problem, we have considered 4 different benchmarks: in the i-th one, $1 \leqslant a_\nu \leqslant 3i-2$ for $i = 1, 2, 3, 4$. Results are presented on Table 3 similarly to the previous section.

The best encoding in this problem is clearly MDD. The best methods are LCG and LD-MDD and MDD. Gurobi and logarithmic methods are not a good option in these problems.

9.4 Sport Leagues Scheduling

The last experiment considers scheduling a double round-robin sports league of N teams. All teams meet each other once in the first $N-1$ weeks and again in the second $N-1$ weeks, with exactly one match per team each week. A given pair of teams must play at the home of one team in one half, and at the home of the other in the other half, and such matches must be spaced at least a certain minimal number of weeks apart. Additional constraints include, e.g., that no team ever plays at home (or away) three times in a row, other (public order, sportive, TV revenues) constraints, blocking given matches on given days, etc.

Additionally, the different teams can propose a set of constraints with some importance (low, medium or high). It is desired not only to maximize the number of these constraints satisfied, but also to assure that at least some of the constraints of every team are satisfied. More information can be found at [1].

Low-importance constraints are given a weight of 1; medium-importance, 5, and high-importance, 10. For every constraint proposed by a team i, a new Boolean variable $x_{i,j}$ is created. This variable is set to true if the constraint is violated. For every team, a pseudo-Boolean constraint $\sum_j w_{i,j} x_{i,j} \leqslant K_i$ is imposed. The objective function to minimize is $\sum_i \sum_j w_{i,j} x_{i,j}$. The data is based on real-life instances.

Even though this problem only has pseudo-Boolean constraints, linear integer constraints arise from grouping identical coefficients. We have considered 10 different problems with 20 random seeds. In all the problems, the optimal value was found around 30. The results are shown in Table 4.

For sports league scheduling problems MDD is clearly the best encoding, followed by BDD-Dec. MDD and LD-MDD are the best methods. Gurobi is unable to handle these problems well (at least with the best model we could devise).

	15s	60s	300s	900s	3600s
Adder	0	0	0.031	0.108	0.167
BDD	0.038	0.061	0.26	0.397	0.537
BDD-Dec	0.037	0.069	0.267	0.441	0.582
MDD	0.034	0.073	**0.292**	**0.46**	0.583
Support	0.035	0.07	0.254	0.404	0.545
LD-MDD	**0.039**	**0.077**	0.277	0.443	**0.592**
LD-Sup	0.035	0.066	0.269	0.417	0.555
LCG	0.023	0.063	0.159	0.287	0.421
Gurobi	0	0	0	0	0

Table 4. Average quality from 200 sport scheduling league benchmarks.

The BDD encoding for these problems is equivalent to using the BDD encoding of [3] for the original pseudo-Boolean constraints. Comparing BDD and MDD illustrates that by using LI constraint encoding we can improve one of the best known approach for PB constraints, in cases where the PB shares coefficients.

10 Conclusion

We have introduced a new domain-consistent encoding (MDD) for linear integer constraints. For small and medium-sized domains, this decomposition substantially improves the current state-of-the-art SAT encodings for LI constraints. It uniformly beats the only other domain consistent encoding (Support) as execution time increases. Combining this encoding with lazy decomposition, we create a hybrid method for LI constraints which is robust across the benchmark suite and rarely substantially bettered by any encoding or propagation method.

We have also introduced a new method (BDD-Dec) for encoding LI constraints based on the logarithmic decomposition of domains and coefficients, substantially improving on the previous state-of-the-art logarithmic method (BDD). This provides a robust alternative to domain-consistent methods in problems with large domains.

As future work, we want to combine lazy decomposition with logarithmic methods for large-domain problems. We are also designing a lazy decomposition solver which dynamically selects which type of encoding apply to every constraint to decompose.

Acknowledgments NICTA is funded by the Australian Government through the Department of Communications and the Australian Research Council through the ICT Centre of Excellence Program.

References

1. Abío, I.: Solving hard industrial combinatorial problems with SAT. Ph.D. thesis, Technical University of Catalonia (UPC) (2013)
2. Abío, I., Nieuwenhuis, R., Oliveras, A., Rodríguez-Carbonell, E.: A Parametric Approach for Smaller and Better Encodings of Cardinality Constraints. In: Schulte, C. (ed.) Proceedings of the 19th International Conference on Principles and Practice of Constraint Programming. LNCS, vol. 8124, pp. 80–96. Springer (2013)

3. Abío, I., Nieuwenhuis, R., Oliveras, A., Rodríguez-Carbonell, E., Mayer-Eichberger, V.: A New Look at BDDs for Pseudo-Boolean Constraints. Journal of Artificial Intelligence Research (JAIR) 45(1), 443–480 (Sep 2012)
4. Abío, I., Nieuwenhuis, R., Oliveras, A., Rodríguez-Carbonell, E., Stuckey, P.J.: To Encode or to Propagate? The Best Choice for Each Constraint in SAT. In: Schulte, C. (ed.) Proceedings of the 19th International Conference on Principles and Practice of Constraint Programming. LNCS, vol. 8124, pp. 97–106. Springer (2013)
5. Abío, I., Stuckey, P.J.: Conflict-Directed Lazy Decomposition. In: 18th International Conference on Principles and Practice of Constraint Programming. pp. 70–85. CP'12 (2012)
6. Ansótegui, C., Bofill, M., Manyà, F., Villaret, M.: Extending Multiple-Valued Clausal Forms with Linear Integer Arithmetic. In: ISMVL. pp. 230–235. IEEE (2011)
7. Ansótegui, C., Manyà, F.: Mapping problems with finite-domain variables to problems with boolean variables. In: Proceedings of the 7th international conference on Theory and Applications of Satisfiability Testing. pp. 1–15. SAT '04, Springer-Verlag, Berlin, Heidelberg (2005)
8. Argelich, J., Manyà, F.: Exact Max-SAT solvers for over-constrained problems. Journal of Heuristics 12(4-5), 375–392 (Sep 2006)
9. Asín, R., Nieuwenhuis, R., Oliveras, A., Rodríguez-Carbonell, E.: Cardinality Networks: a theoretical and empirical study. Constraints 16(2), 195–221 (2011)
10. Bartzis, C., Bultan, T.: Efficient BDDs for bounded arithmetic constraints. International Journal on Software Tools for Technology Transfer 8(1), 26–36 (2006)
11. Berezin, S., Ganesh, V., Dill, D.: An Online Proof-Producing Decision Procedure for Mixed-Integer Linear Arithmetic. In: Garavel, H., Hatcliff, J. (eds.) Tools and Algorithms for the Construction and Analysis of Systems, Lecture Notes in Computer Science, vol. 2619, pp. 521–536. Springer Berlin Heidelberg (2003)
12. Blazewicz, J., Lenstra, J., Kan, A.: Scheduling subject to resource constraints: classification and complexity. Discrete Applied Mathematics 5(1), 11–24 (1983)
13. Bofill, M., Nieuwenhuis, R., Oliveras, A., Rodríguez-Carbonell, E., Rubio, A.: The Barcelogic SMT Solver. In: Computer-aided Verification (CAV). Lecture Notes in Computer Science, vol. 5123, pp. 294–298. Springer (2008)
14. Bofill, M., Palahí, M., Suy, J., Villaret, M.: Boosting Weighted CSP Resolution with Shared BDDs. In: 12th International Workshop on Constraint Modelling and Reformulation (ModRef 2013). pp. 57–73 (2013)
15. Dutertre, B., de Moura, L.: A Fast Linear-Arithmetic Solver for DPLL(T). In: Ball, T., Jones, R.B. (eds.) 18th International Conference on Computer Aided Verification, CAV '06. Lecture Notes in Computer Science, vol. 4144, pp. 81–94. Springer (2006)
16. Feydy, T., Stuckey, P.: Lazy clause generation reengineered. In: Gent, I. (ed.) Proceedings of the 15th International Conference on Principles and Practice of Constraint Programming. LNCS, vol. 5732, pp. 352–366. Springer (2009)
17. Fu, Z., Malik, S.: Solving the minimum-cost satisfiability problem using SAT based branch-and-bound search. In: Proceedings of the 2006 IEEE/ACM international conference on Computer-aided design. pp. 852–859. ICCAD '06, ACM, New York, NY, USA (2006)
18. Gent, I.P.: Arc consistency in SAT. In: in Proceedings of ECAI 2002. pp. 121–125. IOS Press (2002)

19. Gent, I.P., Nightingale, P.: A new encoding of AllDifferent into SAT. 3rd International Workshop on Modelling and reformulating Constraint Satisfaction Problems (CP2004) pp. 95–110 (2004)
20. Gurobi Optimization, I.: Gurobi optimizer reference manual (2013), `http://www.gurobi.com`
21. Harvey, W., Stuckey, P.: Improving linear constraint propagation by changing constraint representation. Constraints 8(2), 173–207 (2003)
22. Nethercote, N., Stuckey, P., Becket, R., Brand, S., Duck, G., Tack, G.: Minizinc: Towards a standard CP modelling language. In: Bessiere, C. (ed.) Proceedings of the 13th International Conference on Principles and Practice of Constraint Programming. LNCS, vol. 4741, pp. 529–543. Springer (2007)
23. Nieuwenhuis, R., Oliveras, A., Tinelli, C.: Solving SAT and SAT Modulo Theories: From an abstract Davis–Putnam–Logemann–Loveland procedure to DPLL(T). Journal of the ACM, JACM 53(6), 937–977 (2006)
24. Pipatsrisawat, K., Darwiche, A.: On modern clause-learning satisfiability solvers. Journal of Automated Reasoning 44(3), 277–301 (2010)
25. Srinivasan, A., Ham, T., Malik, S., Brayton, R.: Algorithms for discrete function manipulation. In: Computer-Aided Design, 1990. ICCAD-90. Digest of Technical Papers., 1990 IEEE International Conference on. pp. 92–95 (1990)
26. Tamura, N., Taga, A., Kitagawa, S., Banbara, M.: Compiling finite linear CSP into SAT. Constraints 14(2), 254–272 (2009)
27. Walsh, T.: SAT v CSP. In: Dechter, R. (ed.) Proceedings of the 6th International Conference on Principles and Practice of Constraint Programming. Lecture Notes in Computer Science, vol. 1894, pp. 441–456. Springer (2000)
28. Warners, J.P.: A Linear-Time Transformation of Linear Inequalities into Conjunctive Normal Form. Information Processing Letters 68(2), 63–69 (1998)