Encoding Linear Constraints with Implication Chains to CNF

Ignasi Abío[1(✉)], Valentin Mayer-Eichberger[1,2], and Peter J. Stuckey[1,3]

[1] NICTA, Canberra, Australia
{ignasi.abio,valentin.mayer-eichberger,peterj.stuckey}@nicta.com.au
[2] University of New South Wales, Sydney, Australia
[3] University of Melbourne, Melbourne, Australia

Abstract. Linear constraints are the most common constraints occurring in combinatorial problems. For some problems which combine linear constraints with highly combinatorial constraints, the best solving method is translation to SAT. Translation of a single linear constraint to SAT is a well studied problem, particularly for cardinality and pseudo-Boolean constraints. In this paper we describe how we can improve encodings of linear constraints by taking into account implication chains in the problem. The resulting encodings are smaller and can propagate more strongly than separate encodings. We illustrate benchmarks where the encoding improves performance.

1 Introduction

In this paper we study linear integer constraints (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 $\{<, >, \leq, \geq, =, \neq\}$. We will assume w.l.o.g that $\#$ is $\leq$, the a_i are positive and all the domains of the variables are $\{0, 1..d_i\}$, since other cases can be reduced to this one.[1] Special case of linear constraints are: pseudo-Boolean (PB) constraints where the domain of each variable is $\{0..1\}$, cardinality (CARD) constraints where additionally $a_i = 1, 1 \leq i \leq n$, and at-most-one (AMO) constraints where additionally $a_0 = 1$.

Linear integer constraints appear in many combinatorial problems such as scheduling, planning or software verification, and, therefore, many different SMT solvers [9,14] and encodings [4,6,11] have been suggested for handling them. There are two main approaches to encoding linear constraints: cardinality constraints are encoded as some variation of a sorting network [3]; multi-decision diagrams (MDDs) are used to encode more general linear constraints [5], which in the special case of pseudo-Boolean constraints collapse to binary decision diagrams (BDDs).

[1] See [5] for details. Note that propagation is severely hampered by replacing equalities by a pair of inequalities.

G. Pesant (Ed.): CP 2015, LNCS 9255, pp. 3–11, 2015.
DOI: 10.1007/978-3-319-23219-5_1

Any form of encoding linear constraints to SAT introduces many intermediate Boolean variables, and breaks the constraint up into many parts. This gives us the opportunity to improve the encoding if we can recognize other constraints in the problem that help tighten the encoding of some part of the linear constraint.

Example 1. Consider a pseudo-Boolean constraint $x_1 + 2x_2 + 2x_3 + 4x_4 + 5x_5 + 6x_6 + 8x_7 \leq 14$ where we also have that $x_2 + x_3 + x_5 \leq 1$ we can rewrite the constraints as $x_1 + 2x_{235} + 4x_4 + 3x_5 + 6x_6 + 8x_7 \leq 14$ and $x_{235} \equiv (x_2 + x_3 + x_5 = 1)$, where x_{235} is a new Boolean variable. Notice, that x_{235} can be used to encode the at-most-one constraint. □

Example 2. Consider a pseudo-Boolean constraint $4x_1 + 2x_2 + 5x_3 + 4x_4 \leq 9$ and the implications $x_1 \leftarrow x_2$ (i.e. $x_1 \vee \neg x_2$) and $x_2 \leftarrow x_3$. Separately they do not propagate, but considered together we can immediately propagate $\neg x_3$. □

In this paper we show how to encode pseudo-Boolean constraints taking into account implication chains, as seen in Example 2. The resulting encodings are no larger than the separate encoding, but result in strictly stronger propagation. The approach also allows us to encode general linear integer constraints, and is a strict generalization of the MDD encoding of linear integer constraints [5]. We show how these new combined encodings are effective in practice on a set of hard real-life sports scheduling examples, and that the combination of pseudo-Booleans with implication chains arises in a wide variety of models.

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 $l_1 \vee \cdots \vee l_n$. An *implication* $x_1 \rightarrow x_2$ is notation for the clause $\neg x_1 \vee x_2$, similarly $x_1 \leftarrow x_2$ denotes $x_1 \vee \neg x_2$. 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* ($\top$) in A if $l \in A$, is *false* ($\perp$) in A if $\neg l \in A$, and is *undefined* in A otherwise. 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. A detailed explanation can be found in [7].

Let $[l..u]$ where l and u are integers represent the set $\{l, \ldots, u\}$. Let y be an integer variable with domain $[0..d]$. The *order encoding* introduces Boolean variables y_i for $1 \leq i \leq d$. A variable y_i is true iff $y < i$. The encoding also introduces the clauses $y_i \rightarrow y_{i+1}$ for $1 \leq i < d$.

2.2 Multi Decision Diagrams

A directed acyclic graph is called an *ordered Multi Decision Diagram* (MDD) 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 array of Booleans $[x_{i1}, \ldots, x_{id_i}]$ representing the order encoding of integer variable y_i where y_i ranges from $[0..d_i]$. The variable y_i is called the *selector variable.*
- Every node labeled by y_i has $d_i + 1$ outgoing edges, labelled $x_{i1}, \neg x_{i1}, \ldots, \neg x_{id_i}$.
- Each edge goes from a node with selector y_i to a node with selector variable y_j has $i < j$.

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 y_i and y_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. A *Binary Decision Diagram* (BDD) is an MDD where $\forall i, d_i = 1$.

An MDD represents a function $f : \{0, 1, \ldots, d_1\} \times \{0, 1, \ldots, d_2\} \times \cdots \times \{0, 1, \ldots, d_n\} \to \{\bot, \top\}$ in the obvious way. Moreover, given a fixed variable ordering, there is only one MDD representing that function. We refer to [17] for further details about MDDs.

A function f is *anti-monotonic* in argument y_i if $v_i \geq v'_i$ implies that $f(v_1, \ldots, v_{i-1}, v_i, v_{i+1}, \ldots, v_n) = \top \Rightarrow f(v_1, \ldots, v_{i-1}, v'_i, v_{i+1}, \ldots, v_n) = \top$ for all values $v_1, \ldots, v_n, v'_i$. An MDD is *anti-monotonic* if it encodes a function f that is anti-monotonic in all arguments. We shall only consider anti-monotonic MDDs in this paper.

Given an anti-monotonic MDD $\mathcal{M}$, we can encode it into CNF by introducing a new Boolean variable b_o to represent each node o in the MDD $\mathcal{M}$; unary clauses $\{b_{\mathcal{T}}, \neg b_{\mathcal{F}}, b_r\}$ where r is the root node of the MDD; and clauses $\{\neg b_o \vee b_{o_0}\} \cup \{\neg b_o \vee x_{ij} \vee b_{o_j} \mid j \in [1..d_i]\}$ for each node o of the form $\mathsf{mdd}([x_{i1}, \ldots, x_{id_i}], [o_0, o_1, \ldots, o_{d_i}])$. See [5] for more details.

We can encode arbitrary MDDs to SAT using Tseitin transformation but the encoding is substantially more complicated.

3 Pseudo Boolean Constraints and Chains

A *chain* $x_1 \Leftarrow x_2 \Leftarrow \cdots \Leftarrow x_n$ is a constraint requiring $x_1 \leftarrow x_2$, $x_2 \leftarrow x_3$, …, $x_{n-1} \leftarrow x_n$. A *unary chain* x_1 is the trivial case that imposes no constraint. A chain is *compatible* with an ordered list of Boolean variables $x_1, \ldots, x_n$ if the chain is of the form $x_l \Leftarrow x_{l+1} \Leftarrow \cdots \Leftarrow x_k, l \leq k$. Given an ordered list L of Boolean variables $x_1, \ldots, x_n$ a *chain coverage* S is a set of variable-disjoint compatible chains such that each variable appears in exactly one chain. We will sometimes treat a chain coverage S as a Boolean formula equivalent to the constraints of the chains appearing in S. Given a variable ordering and a set of disjoint compatible chains we can always construct a chain coverage by adding in unary chains.

Example 3. Given list L of variables $x_1, \ldots, x_9$ and chains $x_1 \Leftarrow x_2 \Leftarrow x_3$, $x_5 \Leftarrow x_6$, $x_7 \Leftarrow x_8$, then a chain coverage S of L is $\{x_1 \Leftarrow x_2 \Leftarrow x_3, x_4, x_5 \Leftarrow x_6, x_7 \Leftarrow x_8, x_9\}$. S represents the constraint $x_1 \leftarrow x_2 \wedge x_2 \leftarrow x_3 \wedge x_5 \leftarrow x_6 \wedge x_7 \leftarrow x_8$. □

Given a Boolean variable ordering L, a PB constraint C and chain coverage S of L we will demonstrate how to build an MDD to encode the constraint C taking into account the chain constraints in S. First lets examine how chains arise in models.

At-most-one constraints. Given PB $C \equiv a_1x_1 + \ldots + a_nx_n \leq a_0$ and AMO $A \equiv x_1 + x_2 + \ldots + x_k \leq 1$. We can reformulate A using new variables x'_j where $x_1 + \ldots + x_k = x'_1 + \ldots + x'_k$ using the ladder encoding [12] which gives $x_i \rightarrow x'_i$ for $i = 1 \ldots k$, $x_i \rightarrow \neg x'_{i+1}$ and $x'_{i+1} \rightarrow x'_i$ for $i = 1 \ldots k-1$. If $[a'_1, \ldots, a'_k]$ is the sorted array of coefficients $[a_1, \ldots, a_k]$ then C can be written as $a'_1x'_1 + (a'_2 - a'_1)x'_2 + \cdots + (a'_k - a'_{k-1})x'_k + a_{k+1}x_{k+1} + \cdots a_nx_n \leq a_0$. The chain comes from the auxiliary variables: $x'_1 \Leftarrow x'_2 \Leftarrow \cdots \Leftarrow x'_k$.

General linear integer constraints. A general LI $C \equiv a_1y_1 + \cdots a_my_m \leq a_0$ can be expressed as a PB with chains. We encode each integer y_i with domain $[0..d_i]$ by the order encoding $[x_{i1}, \ldots, x_{id_i}]$ and then C can be rewritten as $a_1(\neg x_{11}) + \cdots + a_1(\neg x_{1d_1}) + \cdots + a_m(\neg x_{m1}) + \cdots + a_m(\neg x_{md_m}) \leq a_0$ with chains $\neg x_{i1} \Leftarrow \neg x_{i2} \Leftarrow \cdots \Leftarrow \neg x_{id_i}$.

Shared coefficients. Frequently, PB constraints contain a large number of coefficients that are the same. This structure can be exploited. A similar technique of grouping shared coefficients is described in [5] which in our context is restated using chains. Given a PB $C \equiv ax_1 + \cdots + ax_k + a_{k+1}x_{k+1} + \cdots + a_nx_n \leq a_0$ where the first k variables share the same coefficient. We introduce new variables $x'_1, \ldots, x'_k$ to encode the sum $x_1 + \cdots + x_k$ so $x_1 + \ldots + x_k = x'_1 + \ldots + x'_k$ and encode this constraint (usually using some form of sorting network [3]). This ensures that $x'_1 \Leftarrow x'_2 \Leftarrow \cdots \Leftarrow x'_k$. Then C can be rewritten as $ax'_1 + \cdots + ax'_k + a_{k+1}x_{k+1} + \cdots + a_nx_n \leq a_0$. There are several advantages with this rewritten version. The sorting network can be represented more compactly than the same logic in an MDD ($O(k \cdot \log^2 k)$ vs $O(k^2)$). Secondly, the introduced variables x'_j are meaningful for the constraint and are likely to be useful for branching and in conflict clause learning during the search. Moreover, the sorted variables may be reusable for rewriting other constraints.

Binary implicants. Finally, a more general method is to automatically extract chains from the global problem description. There are a number of methods to detect binary implicants of CNF encodings [10,13]. Given a set of binary implicants B and a PB constraint C we can search for a chain coverage S implied by B, and an ordering L of the variables in C with which S is compatible, and then encode the reordered constraint C making use of the chain coverage S.

In the experimental section of this paper we have only considered the first three types of chains.

4 Translating Through MDDs with Chains

The main algorithm in this section generalizes the construction of an MDD in [5]. We first restate definitions of the original algorithm and then show how to take advantage of chains in the new construction. The CNF decomposition has desirable properties, i.e. we show that the encoding is more compact and propagates stronger.

4.1 Preliminaries for the Construction

Let $\mathcal{M}$ be the MDD of pseudo-Boolean C 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 pseudo-Boolean constraint $a_i x_i + \cdots + a_n x_n \leq \alpha$. It is easy to see that this definition corresponds in fact to an interval. The key point in constructing the MDD 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 $a_i x_i + \cdots + a_n x_n \leq a'_0$ 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 $\mathbf{search}(K, L_i)$ searches 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 $\mathbf{insert}(([\beta, \gamma], \mathcal{M}), L_i)$ for insertions.

4.2 Algorithm and Properties of the Construction

In this section we show how to translate a PB $C \equiv a_1 x_1 + \ldots a_n x_n \leq a_0$ and a chain coverage S for variable order $x_1, \ldots, x_n$. Algorithm 1 describes the construction of the MDD. The initial call is $\mathbf{MDDChain}(1, C, S)$. The call $\mathbf{MDDChain}(i, C', S)$ recursively builds an MDD for $C' \wedge S$ by building the i^{th} level. If the chain including x_i is $x_i \Leftarrow \cdots \Leftarrow x_k$ it builds an MDD node that has child nodes with selector x_{k+1}. If the chain for x_i is unary this is the usual MDD (BDD) construction.

Algorithm 1. Procedure **MDDChain**

```
Require: $i \in \{1, 2, \ldots, n+1\}$ and pseudo-Boolean constraint $C' : a_i x_i + \ldots + a_n x_n \leq a'_0$ and chain coverage $S$ on $[x_1, \ldots, x_n]$
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:     $\delta_0 \leftarrow 0$
 6:     let $\{x_i \Leftarrow x_{i+1} \Leftarrow \cdots \Leftarrow x_k\} \in S$    % including unary chain $x_i$
 7:     $u \leftarrow k - i + 1$
 8:     for all $j$ such that $0 \leq j \leq u$ do
 9:         $([\beta_j, \gamma_j], \mathcal{M}_j) \leftarrow$ MDDChain($k+1, a_{k+1}x_{k+1} + \cdots + a_n x_n \leq a'_0 - \delta_j, S$).
10:         $\delta_{j+1} \leftarrow \delta_j + a_{i+j}$
11:     end for
12:     $\mathcal{M} \leftarrow \mathsf{mdd}([x_i, \ldots, x_k], \mathcal{M}_0, \ldots, \mathcal{M}_u)$
13:     $[\beta, \gamma] \leftarrow [\beta_0, \gamma_0] \cap [\beta_1 + \delta_1, \gamma_1 + \delta_1] \cap \cdots \cap [\beta_u + \delta_u, \gamma_u + \delta_u]$.
14:     insert($([\beta, \gamma], \mathcal{M}), L_i$).
15:     return $([\beta, \gamma], \mathcal{M})$.
16: end if
```

Example 4. The MDDs that result from **MDDChain**$(1, C, S)$ where $C \equiv 4x_1 + 2x_2 + 5x_3 + 4x_4 \leq 9$ of Example 2 encoded with chain coverage (a) $S = \{x_1, x_2, x_3, x_4\}$ (no chains) and (b) $S = \{x_1 \Leftarrow x_2 \Leftarrow x_3, x_4\}$ are shown in Figure 1. The diagrams show $[\beta, \gamma]$ for each node with the remainder of the constraint at the left. Unit propagation of the CNF of (b) sets $x_3 = \bot$ immediately since $4x_4 \leq -1$ is $\bot$.

We can prove that the algorithm returns a correct MDD, that is no larger than the MDD (BDD) encoding of C, and that the resulting CNF encoding is domain consistent on the original variables $x_1, \ldots, x_n$. Proofs are available at [1].

Theorem 1. *Given a pseudo-Boolean constraint $C \equiv a_1x_1 + \cdots + a_nx_n \leq a_0$ and chain coverage S on $[x_1, \ldots, x_n]$ then* **MDDChain**$(1, C, S)$ *returns an MDD $\mathcal{M}$ representing function f such that constraint $C \wedge S \models f$. The running time of the algorithm is $O(n \cdot a_0 \cdot \log a_0)$.* □

Theorem 2. *Given a pseudo-Boolean constraint $C \equiv a_1x_1 + \cdots + a_nx_n \leq a_0$ and chain coverage S on $[x_1, \ldots, x_n]$ then the MDD* **MDDChain**$(1, C, S)$ *has no more nodes than* **MDDChain**$(1, C, \{x_1, \ldots, x_n\})$*, the BDD for C.* □

Theorem 3. *Given a pseudo-Boolean constraint $C \equiv a_1x_1 + \cdots + a_nx_n \leq a_0$ and chain coverage S on $[x_1, \ldots, x_n]$ then unit propagation on the CNF encoding of* **MDDChain**$(1, C, S) \wedge S$ *enforces domain consistency of $C \wedge S$ on variables $x_1, \ldots, x_n$.* □

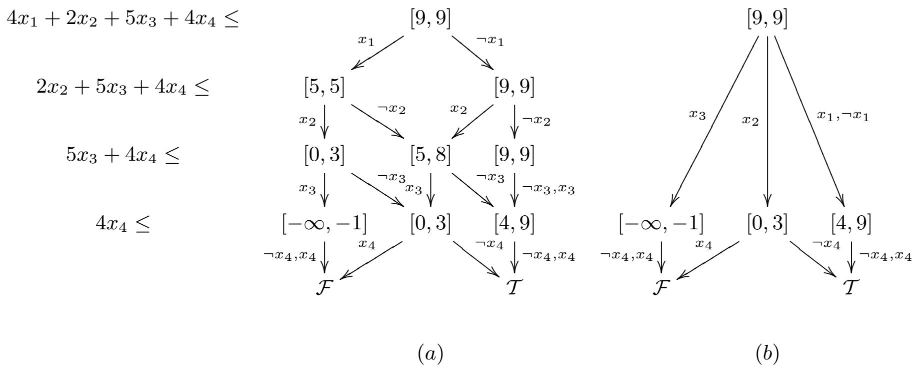

Fig. 1. The MDDs that result from $4x_1 + 2x_2 + 5x_3 + 4x_4 \leq 9$ encoded (a) without and (b) with the chain $x_1 \Leftarrow x_2 \Leftarrow x_3$.

5 Experiments

To illustrate the advantage of combined compilation we consider a challenging combinatorial optimization problem where both AMO and shared coefficients chains arise.

Sports league scheduling is a challenging combinatorial optimization problem. We consider 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). We aim 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 in [2].

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 $\sum_j w_{i,j} x_{i,j} \leq K_i$ 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.

Desired constraints typically refer to critical weeks in the schedule, e.g. around Christmas, or other key dates, and preferences of different teams almost always clash. Double round-robin tournaments contain a lot of AMO and EO constraints (for instance, each week each team meets exactly one team). These AMO constraints can be used to simplify the desired constraints.

Table 1. Results for sports league scheduling, showing the number of runs that find a solution of different quality after different time limits (seconds).

Quality	Some solution			cost $\leq$ 30 + best			cost $\leq$ 20 + best			cost $\leq$ 10 + best		
Timelimit	300	900	3600	300	900	3600	300	900	3600	300	900	3600
MDD1	148	190	199	21	55	107	17	35	74	6	25	51
MDD2	151	**194**	199	27	59	115	19	38	81	12	25	43
MDD3	**160**	191	**200**	**56**	**107**	**162**	**45**	**72**	**121**	**41**	**52**	**87**
LCG	69	123	172	21	29	51	18	21	35	14	20	27
Gurobi	0	0	0	0	0	0	0	0	0	0	0	0

The benchmark consists of 10 instances and each method is run 20 times with different seeds, for 200 total runs. Compared methods are: MDD1, the usual MDD (in fact, BDD) method to encode PBs [4]; MDD2, the method of [5] using sorting networks for the identical coefficients and then using an MDD; MDD3, the method defined herein; LCG, using lazy clause generation [15]; and Gurobi, using the MIP solver Gurobi. *Barcelogic* [8] SAT Solver was used in methods MDD1, MDD2, MDD3 and LCG.

The results can be shown at Table 1. The number of times a solution has been found within the time limit can be found at columns 2-4. Columns 5-7 present the number of times (within the timelimit) a method finds a solution of cost at most $best + 30$, where $best$ is the cost of the best solution found by any method. Similarly, columns 8-10 and 11-13 contain the number of times a solution of cost at most $best + 20$ and $best + 10$ has been found.

As we can see the new encoding substantially improves on previous encodings of the problem. For these sports leagues scheduling problems it is well known that other solving approaches do not compete with SAT encoding [2].

6 Conclusion and Future Work

We demonstrate a new method for encoding pseudo-Boolean constraints taking into account implications chains. The improved encoding is beneficial on hard benchmark problems. The approach is an extension on earlier work on encoding linear constraints to SAT [5]. The approach is related to the propagator for the `increasing_sum` constraint $y = a_1 y_1 + \cdots + a_n y_n \wedge y_1 \leq y_2 \leq \cdots \leq y_n$ described in [16], which combines a linear constraint with a "chain" of integer inequalities. Interestingly, `increasing_sum` is not directly encodable as an MDD using the method herein, but it does suggest that the methods can be extended to arbitrary sets of chains all compatible with a global variable order. Another interesting direction for future work is to consider combining chains with the sorting networks encodings of linear constraints (e.g. [11]).

Acknowledgments. NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.

References

1. Abio, I., Mayer-Eichberge, V., Stuckey, P.: Encoding linear constraints with implication chains to CNF. Tech. rep., University of Melbourne (2015). http://www.people.eng.unimelb.edu.au/pstuckey/papers/cp2015a.pdf
2. Abío, I.: Solving hard industrial combinatorial problems with SAT. Ph.D. thesis, Technical University of Catalonia (UPC) (2013)
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.) CP 2013. LNCS, vol. 8124, pp. 80–96. Springer, Heidelberg (2013)
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. (JAIR) **45**, 443–480 (2012)
5. Abío, I., Stuckey, P.J.: Encoding linear constraints into SAT. In: O'Sullivan, B. (ed.) CP 2014. LNCS, vol. 8656, pp. 75–91. Springer, Heidelberg (2014)
6. Bailleux, O., Boufkhad, Y., Roussel, O.: New encodings of pseudo-boolean constraints into CNF. In: Kullmann, O. (ed.) SAT 2009. LNCS, vol. 5584, pp. 181–194. Springer, Heidelberg (2009)
7. Biere, A., Heule, M., van Maaren, H., Walsh, T. (eds.): Handbook of Satisfiability, Frontiers in Artificial Intelligence and Applications, vol. 185. IOS Press (2009)
8. Bofill, M., Nieuwenhuis, R., Oliveras, A., Rodríguez-Carbonell, E., Rubio, A.: The barcelogic SMT solver. In: Gupta, A., Malik, S. (eds.) CAV 2008. LNCS, vol. 5123, pp. 294–298. Springer, Heidelberg (2008)
9. Dutertre, B., de Moura, L.: The YICES SMT Solver. Tech. rep., Computer Science Laboratory, SRI International (2006). http://yices.csl.sri.com
10. Eén, N., Biere, A.: Effective preprocessing in SAT through variable and clause elimination. In: Bacchus, F., Walsh, T. (eds.) SAT 2005. LNCS, vol. 3569, pp. 61–75. Springer, Heidelberg (2005)
11. Eén, N., Sörensson, N.: Translating Pseudo-Boolean Constraints into SAT. Journal on Satisfiability, Boolean Modeling and Computation **2**(1–4), 1–26 (2006)
12. Gent, I.P., Prosser, P., Smith, B.M.: A 0/1 encoding of the `GACLex` constraint for pairs of vectors. In: ECAI 2002 workshop W9: Modelling and Solving Problems with Constraints. University of Glasgow (2002)
13. Heule, M.J.H., Järvisalo, M., Biere, A.: Efficient CNF simplification based on binary implication graphs. In: Sakallah, K.A., Simon, L. (eds.) SAT 2011. LNCS, vol. 6695, pp. 201–215. Springer, Heidelberg (2011)
14. de Moura, L., Bjorner, N.: Z3: An Efficient SMT Solver. Tech. rep., Microsoft Research, Redmond (2007). http://research.microsoft.com/projects/z3
15. Ohrimenko, O., Stuckey, P., Codish, M.: Propagation via lazy clause generation. Constraints **14**(3), 357–391 (2009)
16. Petit, T., Régin, J.-C., Beldiceanu, N.: A Θ(n) bound-consistency algorithm for the increasing sum constraint. In: Lee, Jimmy (ed.) CP 2011. LNCS, vol. 6876, pp. 721–728. Springer, Heidelberg (2011)
17. Srinivasan, A., Ham, T., Malik, S., Brayton, R.: Algorithms for discrete function manipulation. In: 1990 IEEE International Conference on Computer-Aided Design, ICCAD 1990, pp. 92–95. Digest of Technical Papers (1990)