Encoding Linear Constraints with Implication Chains to CNF

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

[1] NICTA
[2] University of New South Wales
[3] University of Melbourne
`<firstname>.<lastname>@nicta.com.au`

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.

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. $\square$

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 $\neg x_3$. $\square$

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* ($\bot$) 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 $x_1 \leftarrow x_2 \wedge x_2 \leftarrow x_3 \wedge x_5 \leftarrow x_6 \wedge x_7 \leftarrow x_8$. $\square$

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 two 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 **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 **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 **MDDChain**$(1, C, S)$. The call **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.

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

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
```

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 $x_1, \ldots, x_n$.* □

5 Experiments

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

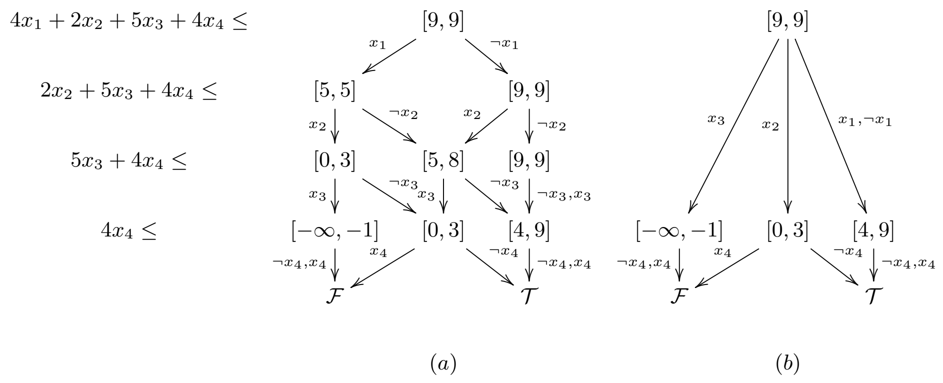

Fig. 1. The MDDs that result from $4x_1 + 2x_2 + 5x_3 + 4x_4 \leq 9$ encoded (a) without and (b) with the chain $x_1 \Leftarrow x_2 \Leftarrow x_3$.

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.

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,

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 ≤ 30 + best			cost ≤ 20 + best			cost ≤ 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

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]).

Acknowledgement 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), 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: Principles and Practice of Constraint Programming - 19th International Conference, CP 2013, Uppsala, Sweden, September 16-20, 2013. Proceedings. pp. 80–96 (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: Principles and Practice of Constraint Programming - 20th International Conference, CP 2014, Lyon, France, September 8-12, 2014. Proceedings. pp. 75–91 (2014)
6. Bailleux, O., Boufkhad, Y., Roussel, O.: New encodings of pseudo-boolean constraints into cnf. In: SAT. pp. 181–194 (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: Computer-aided Verification (CAV). Lecture Notes in Computer Science, vol. 5123, pp. 294–298. Springer (2008)
9. Dutertre, B., de Moura, L.: The YICES SMT Solver. Tech. rep., Computer Science Laboratory, SRI International (2006), available at `http://yices.csl.sri.com`
10. Eén, N., Biere, A.: Effective preprocessing in SAT through variable and clause elimination. In: Theory and Applications of Satisfiability Testing, 8th International Conference, SAT 2005, St. Andrews, UK, Proceedings. pp. 61–75 (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., Järvisalo, M., Biere, A.: Efficient CNF simplification based on binary implication graphs. In: Theory and Applications of Satisfiability Testing - SAT 2011 - 14th International Conference, SAT 2011, Ann Arbor, USA, 2011. Proceedings. pp. 201–215 (2011)
14. de Moura, L., Bjorner, N.: Z3: An Efficient SMT Solver. Tech. rep., Microsoft Research, Redmond (2007), available at `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., Beldiceanu, N.: A $\theta(n)$ bound-consistency algorithm for the increasing sum constraint. In: Principles and Practice of Constraint Programming - CP 2011 - 17th International Conference, CP 2011, Perugia, Italy, 2011. Proceedings. pp. 721–728 (2011)
17. 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)

A Proofs

In the following, let us fix a pseudo-Boolean constraint

$$C : \sum_{j=1}^{n} a_j x_j \leq a_0,$$

and a chain coverage $S = \{S_1, S_2, \dots, S_r\}$ with

$$S_1 = \{x_1 \Leftarrow x_2 \Leftarrow \cdots \Leftarrow x_{k_2-1}\}, \quad S_2 = \{x_{k_2} \Leftarrow x_{k_2+1} \Leftarrow \cdots \Leftarrow x_{k_3-1}\}, \dots$$

By convention, $k_1 = 1$ and $k_{r+1} = n + 1$.

We denote by $\mathcal{M}$ the MDD $\mathbf{MDDChain}(1, C, S)$, and by $\mathcal{B}$ the (usual) BDD of C. Let μ be the root of $\mathcal{M}$ and μ' be the root of $\mathcal{B}$. Given a node $\nu \in \mathcal{M}$ we define its *level*, denoted by $\mathbf{lvl}(\nu)$, as usual: the root has level 1, its child nodes have level 2, their child nodes have level 3, etc. This is, a node of level i has as selector variables $[x_{k_i}, x_{k_i+1}, \dots, x_{k_{i+1}-1}]$.

A node ν of level i has $d_i := k_{i+1} - k_i + 1$ child nodes, which we call $\mathbf{Child}(\nu, 0)$, $\mathbf{Child}(\nu, 1)$, …, $\mathbf{Child}(\nu, d_i - 1)$. $\mathbf{Child}(\nu, j)$ is the child node of ν when $x_{k_i} = x_{k_i+1} = \cdots = x_{k_i+j-1} = 1$ and $x_{k_i+j} = \cdots = x_{k_{i+1}-1} = 0$.

Given a node $\nu \in \mathcal{M}$, we define the interval of ν as the integer values α such that $C^i_\alpha \wedge S^i$ and $\mathcal{M}_\nu \wedge S^i$ are equivalent as logical functions (i.e., they have the same solutions), where

$$C^i_\alpha := \sum_{j=k_i}^{n} a_j x_j \leq \alpha,$$

$S^i := \{S_i, S_{i+1}, \dots, S_r\}$ and $\mathcal{M}_\nu$ is the function represented by the MDD rooted at ν. It is easy to see that it is in fact an integer interval.

Given a node $\nu \in \mathcal{M}$ at level i and a partial assignment

$$A = \{x_{k_i} = v_{k_i}, x_{k_i+1} = v_{k_i+1}, \dots, x_{k_{i+1}-1} = v_{k_{i+1}-1}\}$$

compatible with S, we define $\mathbf{Path}_A(\nu)$ as $\mathbf{Child}(\nu, v_{k_i} + v_{k_i+1} + \cdots + v_{k_{i+1}-1})$. In a similar way, given a partial assignment

$$A = \{x_{k_i} = v_{k_i}, x_{k_i+1} = v_{k_i+1}, \dots, x_{k_{i+s}-1} = v_{k_{i+s}-1}\}$$

compatible with S, we define $\mathbf{Path}_A(\nu)$ as

$$\mathbf{Child}(\mathbf{Child}(\cdots(\mathbf{Child}(\nu, v_{k_i}+v_{k_i+1}+\cdots+v_{k_{i+1}-1}), \cdots), v_{k_{i+s-1}}+v_{k_{i+s-1}+1}+\cdots+v_{k_{i+s}-1}).$$

Proposition 1 *Let $\nu \in \mathcal{M}$ be a node with $\mathbf{lvl}(\nu) = i$, with child nodes $\nu_0, \nu_1, \dots, \nu_{d_i-1}$. Let $[\beta_j, \gamma_j]$ be the interval of ν_j, and let us define*

$$\beta = \max\{\beta_j + \sum_{s=0}^{j-1} a_{k_i+s} \mid 0 \leq j < d_i\}, \qquad \gamma = \min\{\gamma_j + \sum_{s=0}^{j-1} a_{k_i+s} \mid 0 \leq j < d_i\}$$

Then, $\mathbf{Int}(\nu) = [\beta, \gamma]$.

Proof.

$\subseteq$: Given $\alpha \in [\beta, \gamma]$ we have to show that $\alpha \in \mathbf{Int}(\nu)$. Let

$$A = \{x_{k_i} = v_{k_i}, x_{k_i+1} = v_{k_i+1}, \dots, x_n = v_n\}$$

be an assignment on the variables $x_{k_i}, x_{k_i+1}, \dots, x_n$ satisfying S^i. We have to show that the assignment satisfies the constraint C^i_α if and only if $\mathbf{Path}_A(\nu) = \mathcal{T}$.
Let us define $j = v_{k_i} + v_{k_i+1} + \cdots + v_{k_{i+1}-1}$. Since $\beta \le \alpha \le \gamma$, by definition of β and γ,

$$\beta_j + a_{k_i} + \cdots + a_{k_i+j-1} \le \beta \le \alpha \le \gamma \le \gamma_j + a_{k_i} + \cdots + a_{k_i+j-1}.$$

Therefore,

$$\alpha - a_{k_i} - a_{k_i+1} - \cdots - a_{k_i+j-1} \in [\beta_j, \gamma_j].$$

Since $[\beta_j, \gamma_j]$ is the interval of ν_j, the assignment

$$A' = \{x_{k_{i+1}} = v_{k_{i+1}}, x_{k_{i+1}+1} = v_{k_{i+1}+1}, \dots, x_n = v_n\}$$

satisfies $C^{i+1}_{\alpha - a_{k_i} - a_{k_i+1} - \cdots - a_{k_i+j-1}}$ if and only if $\mathbf{Path}_{A'}(\nu_j) = \mathcal{T}$.
Notice that $\mathbf{Path}_A(\nu) = \mathbf{Path}_{A'}(\nu_j)$ by definition of $\mathbf{Path}$. Also notice that, since $j = v_{k_i} + v_{k_i+1} + \cdots + v_{k_{i+1}-1}$ and A satisfies S^i, we can deduce that $v_{k_i} = v_{k_i+1} = \cdots = v_{k_i+j-1} = 1$ and $v_{k_i+j} = \cdots = v_{k_{i+1}-1} = 0$.
Therefore, A' satisfies $C^{i+1}_{\alpha - a_{k_i} - a_{k_i+1} - \cdots - a_{k_i+j-1}}$ if and only if

$$\sum_{s=k_{i+1}}^{n} a_s v_s \le \alpha - a_{k_i} - \cdots - a_{k_i+j-1} = \alpha - a_{k_i} v_{k_i} - a_{k_i+1} v_{k_i+1} - \cdots - a_{k_{i+1}-1} v_{k_{i+1}-1},$$

if and only if A satisfies C^i_α.

$\supseteq$: Let α be in the interval of ν. We have to show that $\alpha \in [\beta, \gamma]$, this is,

$$\max\{\beta_j + \sum_{s=0}^{j-1} a_{k_i+s} \mid 0 \le j < d_i\} \le \alpha \le \min\{\gamma_j + \sum_{s=0}^{j-1} a_{k_i+s} \mid 0 \le j < d_i\}.$$

Take j in $0 \le j < d_i$, we have to show that $\beta_j + a_{k_i} + \cdots + a_{k_i+j-1} \le \alpha \le \gamma_j + a_{k_i} + \cdots + a_{k_i+j-1}$. Let

$$A = \{x_{k_{i+1}} = v_{k_{i+1}}, x_{k_{i+1}+1} = v_{k_{i+1}+1}, \dots, x_n = v_n\}$$

be any assignment that satisfies S^{i+1}. Let us define $v_{k_i} = v_{k_i+1} = \cdots = v_{k_i+j-1} = 1$, $v_{k_i+j} = \cdots = v_{k_{i+1}-1} = 0$ and

$$A' := A \cup \{x_{k_i} = v_{k_i}, x_{k_i+1} = v_{k_i+1}, \dots x_{k_{i+1}-1} = v_{k_{i+1}-1}\}.$$

Obviously A' is an assignment satisfying S^i, and $\mathbf{Path}_A(\nu_j) = \mathbf{Path}_{A'}(\nu)$.

– Let us assume that $\mathbf{Path}_A(\nu_j) = \mathcal{T}$. Since $\alpha \in \mathbf{Int}(\nu)$, A' satisfies C^i_α, so $\sum_{s=k_i}^n a_s v_s \leq \alpha$. Therefore,

$$\sum_{s=k_{i+1}}^{n} a_s v_s \leq \alpha - a_{k_i} v_{k_i} - a_{k_i+1} v_{k_i+1} - \cdots - a_{k_{i+1}-1} v_{k_{i+1}-1} = \alpha - a_{k_i} - \cdots - a_{k_i+j-1}$$

by definition of $v_{k_i}, \ldots, v_{k_{i+1}-1}$. That means that A satisfies $C^{i+1}_{\alpha - a_{k_i} - \cdots - a_{k_i+j-1}}$.

– Let us assume that $\mathbf{Path}_A(\nu_j) = \mathcal{F}$. Analogously, that means that A does not satisfy $C^{i+1}_{\alpha - a_{k_i} - \cdots - a_{k_i+j-1}}$.

Therefore, given any assignment A satisfying S^{i+1}, $\mathbf{Path}_A(\nu_j) = \mathcal{T}$ if and only if A satisfies $C^{i+1}_{\alpha - a_{k_i} - \cdots - a_{k_i+j-1}}$. By definition, that means that $\alpha - a_{k_i} - \cdots - a_{k_i+j-1} \in \mathbf{Int}(\nu_j) = [\beta_j, \gamma_j]$, so $\beta_j \leq \alpha - a_{k_i} - \cdots - a_{k_i+j-1} \leq \gamma$ as we wanted to prove.

□

Corollary 2 *Given a pseudo-Boolean constraint $C' \equiv a_{k_i} x_{k_i} + \cdots + a_n x_n \leq a'_0$ and chain coverage S^i on $[x_{k_i}, \ldots, x_n]$ then* $\mathbf{MDDChain}(k_i, C', S^i)$ *returns an interval $[\beta, \gamma]$ and an MDD $\mathcal{M}'$ such that $C' \wedge S = \mathcal{M}' \wedge S$ as functions; and $a'_0 \in [\beta, \gamma]$.*

Proof. By induction on $r + 1 - i$. If $i = r + 1$, then C' is $0 \leq a'_0$. If $a'_0 \geq 0$ then the algorithm returns $[0, \infty)$ and $\mathcal{T}$, so the results holds. If $a'_0 < 0$, the algorithm returns $(-\infty, -1]$ and $\mathcal{F}$.

Assume the result holds for $i+1$ and let us prove it for i. $\mathbf{MDDChain}(k_i, C^i_{a'_0}, S^i)$ calls

$$\mathbf{MDDChain}(k_{i+1}, C_j, S^{i+1}), \quad C_j := C^{i+1}_{a'_0 - \sum_{s=0}^{j-1} a_{k_i+j}}$$

for $0 \leq j < d_i$. By Induction hypothesis, these recursive calls return the $[\beta_j, \gamma_j]$ and $\mathcal{M}_j$, where $[\beta_j, \gamma_j]$ is the interval of the root of $\mathcal{M}_j$ and $\mathcal{M}_j \wedge S^{i+1} = C_j \wedge S^{i+1}$. By Proposition 1, the interval returned by $\mathbf{MDDChain}(k_i, C^i_{a'_0}, S^i)$ is correct; it is obvious that a'_0 belongs to it. By definition of MDD, $\mathcal{M}' = \mathsf{mdd}([x_{k_i}, \ldots, x_{k_{i+1}-1}], \mathcal{M}_0, \ldots, \mathcal{M}_{d_i-1})$ is also correct since each $\mathcal{M}_j$ is correct. □

Theorem 1. *Given a pseudo-Boolean constraint $C \equiv a_1 x_1 + \cdots + a_n x_n \leq a_0$ and chain coverage S on $[x_1, \ldots, x_n]$ then* $\mathbf{MDDChain}(1, C, S)$ *returns an MDD $\mathcal{M}$ representing function f such that constraint $C \wedge S \models f$. The running time of the algorithm is $O(n \cdot a_0 \cdot \log a_0)$.*

Proof. By the previous result, $C \wedge S \models f$. Let us prove now that the cost is $O(n \cdot a_0 \cdot \log a_0)$. Let w be the maximal width of $\mathcal{M}$. It is obvious that $w \leq a_0 + 2$ since, given i, each node ν of level i has a non-overlapping interval $[\beta, \gamma]$, and if $\nu \neq \mathcal{F}$, then $0 \leq \beta \leq a_0$.

The cost of a call to **MDDChain** is the cost of all the recursive calls it does plus the cost of **insert** and **search**. These functions have logarithmic cost in the number of elements of L_i, and the number of elements of L_i is bounded by w. Therefore, the cost of the algorithm is bounded by $\log w$ times the total number of recursive calls. Notice that, for each node $\nu \in \mathcal{M}$, the MDD of ν is only computed once: then, ν and its interval is added at L_i. The following time ν is needed, the conditional statement at line 2 would be evaluated to true; the algorithm will return at line 3, producing no recursive calls. Therefore, the total number of calls to **MDDChain** is $\sum_{\nu \in \mathcal{M}} \#child(\nu) = \sum_{\nu \in \mathcal{M}} d_i$.

Since $d_i = k_{i+1} - k_i + 1$ and there are at most w nodes at level i, then

$$\sum_{\nu \in \mathcal{M}, \mathbf{lvl}(\nu)=i} d_i \leq w(k_{i+1} - k_i + 1).$$

Therefore,

$$\sum_{\nu \in \mathcal{M}} d_i \leq w(k_2 - k_1 + 1) + w(k_3 - k_2 + 1) + \cdots = w(r + k_{r+1} - k_1) = w(r + n - 1) \leq 2wn.$$

Therefore, the algorithm cost is $O(wn \log w) \leq O(na_0 \log a_0)$. $\square$

Proposition 3 (Proposition 7 from [4]) *Let ν be a node of $\mathcal{B}$ with selector variable x_i. Let ν_t and ν_f be its true and false children. Assume that $\mathbf{Int}(\nu) = [\beta, \gamma]$, $\mathbf{Int}(\nu_t) = [\beta_t, \gamma_t]$ and $\mathbf{Int}(\nu_f) = [\beta_f, \gamma_f]$. Then, $\beta = \max\{\beta_f, \beta_t + a_i\}$ and $\gamma = \min\{\gamma_f, \gamma_t + a_i\}$.*

Lemma 4 *Let ν be a node of $\mathcal{B}$ with selector variable x_i and interval $[\beta, \gamma]$. Take j with $i < j \leq n + 1$. For every $v_i, v_{i+1}, \ldots, v_{j-1} \in \{0, 1\}$, let us define $\nu_{v_i,\ldots,v_{j-1}} = \mathbf{Path}_{\{x_i = v_i, \ldots, x_{j-1} = v_{j-1}\}}(\nu)$, and $[\beta_{v_i,\ldots,v_{j-1}}, \gamma_{v_i,\ldots,v_{j-1}}] = \mathbf{Int}(\nu_{v_i,\ldots,v_{j-1}})$. Then,*

$$\beta = \max\left\{\beta_{v_i,\ldots,v_{j-1}} + \sum_{s=i}^{j-1} a_s v_s \mid v_i, v_{i+1}, \ldots, v_{j-1} \in \{0,1\}\right\}$$

and

$$\gamma = \min\left\{\gamma_{v_i,\ldots,v_{j-1}} + \sum_{s=i}^{j-1} a_s v_s \mid v_i, v_{i+1}, \ldots, v_{j-1} \in \{0,1\}\right\}.$$

Proof. The results is a direct consequence the previous Proposition. $\square$

Proposition 5 *Let $A = \{x_1 = v_1, x_2 = v_2, \ldots, x_{k_i - 1} = v_{k_i - 1}\}$ be an assignment on the variables $x_1, x_2, \ldots, x_{k_i-1}$ satisfying $S_1, S_2, \ldots, S_{i-1}$. Let $\nu = \mathbf{Path}_A(\mu)$ and $\nu' = \mathbf{Path}_A(\mu')$. Then $\mathbf{Int}(\nu) \supseteq \mathbf{Int}(\nu')$.*

Notice that $\nu = \mathbf{Path}_A(\mu) \in \mathcal{M}$ while $\nu' \in \mathcal{B}$.
Proof. Let i be the level of ν. We can prove the result by induction on $r + 1 - i$.

If $i = r + 1$, then either $\nu = \mathcal{T}$ or $\nu = \mathcal{F}$. Assume that $\nu = \mathcal{T}$. Since $\mathcal{B} \wedge S = \mathcal{M} \wedge S$, by definition of $\mathcal{M}$, $\nu' = \mathcal{T}$, and their interval is $[0, \infty)$ in both cases. Analogously, if $\nu = \mathcal{F}$ both intervals are $(-\infty, -1]$.

Assume the result is true for $i+1$. Let us define the assignments

$$\begin{array}{ll} A_0 & = A \cup \{x_{k_i} = x_{k_i+1} = \ldots = x_{k_{i+1}-1} = 0\} \\ A_1 & = A \cup \{x_{k_i} = 1, x_{k_i+1} = x_{k_i+2} = \ldots = x_{k_{i+1}-1} = 0\} \\ \ldots & \\ A_{d_i-1} & = A \cup \{x_{k_i} = x_{k_i+1} = \ldots = x_{k_{i+1}-1} = 1\} \end{array}$$

Notice that each A_j satisfies $S_1, S_2, \ldots, S_i$. Let us define $\nu_j = \mathbf{Path}_{A_j}(\mu)$, $\nu'_j = \mathbf{Path}_{A_j}(\mu')$, $[\beta_j, \gamma_j] = \mathbf{Int}(\nu_j)$ and $[\beta'_j, \gamma'_j] = \mathbf{Int}(\nu'_j)$. By induction hypothesis, $\mathbf{Int}(\nu_j) \supseteq \mathbf{Int}(\nu'_j)$. By Proposition 1,

$$\begin{aligned}\mathbf{Int}(\nu) &= [\beta_0, \gamma_0] \cap [\beta_1 + a_{k_i}, \gamma_1 + a_{k_i}] \cap \cdots = \\ &= \bigcap_{1 \geq v_{k_i} \geq \ldots \geq v_{k_{i+1}-1} \geq 0} [\beta_{v_{k_i} + \cdots + v_{k_{i+1}-1}} + a_{k_i} v_{k_i} + \cdots + a_{k_{i+1}-1} v_{k_{i+1}-1}, \\ &\qquad \gamma_{v_{k_i} + \cdots + v_{k_{i+1}-1}} + a_{k_i} v_{k_i} + \cdots + a_{k_{i+1}-1} v_{k_{i+1}-1}].\end{aligned}$$

By Lemma 4,

$$\begin{aligned}\mathbf{Int}(\nu') = \bigcap_{v_{k_i}, \ldots, v_{k_{i+1}-1} \in \{0,1\}} & [\beta'_{v_{k_i}, \ldots, v_{k_{i+1}-1}} + a_{k_i} v_{k_i} + \cdots + a_{k_{i+1}-1} v_{k_{i+1}-1}, \\ & \gamma'_{v_{k_i}, \ldots, v_{k_{i+1}-1}} + a_{k_i} v_{k_i} + \cdots + a_{k_{i+1}-1} v_{k_{i+1}-1}].\end{aligned}$$

Notice that for $v_{k_i} = \cdots = v_{k_i+j-1} = 1, v_{k_i+j} = \cdots = v_{k_{i+1}-1} = 0$, $\beta'_{v_{k_i}, \ldots, v_{k_{i+1}-1}} = \beta'_j$ and $\gamma'_{v_{k_i}, \ldots, v_{k_{i+1}-1}} = \gamma'_j$. Since

$$\Big\{v_{k_i}, \ldots, v_{k_{i+1}-1} \in \{0,1\}\Big\} \supseteq \Big\{1 \geq v_{k_i} \geq \ldots \geq v_{k_{i+1}-1} \geq 0\Big\},$$

$$\begin{aligned}&\mathbf{Int}(\nu') = \\ &\bigcap_{v_{k_i}, \ldots, v_{k_{i+1}-1} \in \{0,1\}} [\beta'_{v_{k_i}, \ldots, v_{k_{i+1}-1}} + a_{k_i} v_{k_i} + \cdots + a_{k_{i+1}-1} v_{k_{i+1}-1}, \\ &\qquad \gamma'_{v_{k_i}, \ldots, v_{k_{i+1}-1}} + a_{k_i} v_{k_i} + \cdots + a_{k_{i+1}-1} v_{k_{i+1}-1}] \subseteq \\ &\bigcap_{1 \geq v_{k_i} \geq \ldots \geq v_{k_{i+1}-1} \geq 0} [\beta'_{v_{k_i}, \ldots, v_{k_{i+1}-1}} + a_{k_i} v_{k_i} + \cdots + a_{k_{i+1}-1} v_{k_{i+1}-1}, \\ &\qquad \gamma'_{v_{k_i}, \ldots, v_{k_{i+1}-1}} + a_{k_i} v_{k_i} + \cdots + a_{k_{i+1}-1} v_{k_{i+1}-1}] = \\ &[\beta'_0, \gamma'_0] \cap [\beta'_1 + a_{k_i}, \gamma'_1 + a_{k_i}] \cap \cdots \subseteq \\ &[\beta_0, \gamma_0] \cap [\beta_1 + a_{k_i}, \gamma_1 + a_{k_i}] \cap \cdots = \\ &= \mathbf{Int}(\nu)\end{aligned}$$

□

Corollary 6 *Let A_1 and A_2 be two assignments on the variables $x_1, \ldots, x_{k_i-1}$ with $\mathbf{Path}_{A_1}(\mu') = \mathbf{Path}_{A_2}(\mu')$. Then $\mathbf{Path}_{A_1}(\mu) = \mathbf{Path}_{A_2}(\mu)$.*

This theorem is saying that if two BDD nodes are merged, then the equivalent nodes in the MDD are also merged.

Proof. Given A_1 and A_2, let $\nu_s = \mathbf{Path}_{A_s}(\mu)$, $\nu'_s = \mathbf{Path}_{A_s}(\mu')$. Since $\nu'_1 = \nu'_2$, by the previous result $\mathbf{Int}(\nu_1) \supseteq \mathbf{Int}(\nu'_1) = \mathbf{Int}(\nu'_2) \subseteq \mathbf{Int}(\nu_2)$. Therefore $\mathbf{Int}(\nu_1) \cap \mathbf{Int}(\nu_2) \supseteq \mathbf{Int}(\nu'_1) \neq \emptyset$. By definition of interval, that means that $\nu_1 = \nu_2$. □

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 $\mathbf{MDDChain}(1, C, S)$ has no more nodes than $\mathbf{MDDChain}(1, C, \{x_1, \ldots, x_n\})$, the BDD for C.*

Proof. We can define the injective function $f : \mathcal{M} \to \mathcal{B}$ as, given a node ν, let A be the minimal partial assignment[2] such that $\mathbf{Path}_A(\mu) = \nu$. Then, $f(\nu) := \mathbf{Path}_A(\mu')$. The function is injective due to the previous result. □

Proposition 7 *Let ν be a node of $\mathcal{M}$ at level i, and A be a partial assignment on the variables $\{x_1, \ldots, x_n\}$ consistent with S and closed under unit propagation on S. Then:*

- *b_ν is enforced by unit propagation of $\mathbf{MDDChain}(1, C, S) \wedge A$ if and only if there exists an assignment $B = \{x_1 = v_1, x_2 = v_2, \ldots, x_{k_i-1} = v_{k_i-1}\}$ satisfying S such that $\mathbf{Path}_B(\mu) = \nu$ and for all $s \in [1, k_i - 1]$ with $v_s = 1$, $\{x_s = v_s\} \in A$.*
- *$\neg b_\nu$ is enforced by unit propagation of $\mathbf{MDDChain}(1, C, S) \wedge A$ if and only if there exists an assignment $B = \{x_{k_i} = v_{k_i}, x_{k_i+1} = v_{k_i+1}, \ldots, x_n = v_n\}$ satisfying S such that $\mathbf{Path}_B(\nu) = \mathcal{F}$ and for all $s \in [k_i, n]$ with $v_s = 1$, $\{x_s = v_s\} \in A$.*

Proof.

- By induction on i. If $i = 1$, $\nu = \mu$, so b_ν is always enforced by unit propagation. $B = \emptyset$ is an assignment satisfying the conditions. Let us assume the result holds for i, and let us prove it for $i + 1$.
  - $\Rightarrow$ Assume that b_ν has been propagated. The clause that propagated b_ν can only be $\neg b_{\nu_0} \vee b_\nu$ or $\neg b_{\nu_0} \vee \neg x_{k_{i-1}+j} \vee b_\nu$.
    If $\neg b_{\nu_0} \vee b_\nu$ propagated b_ν, then $\nu = \mathbf{Child}(\nu_0, 0)$ and b_{ν_0} has been propagated. Since $\mathbf{lvl}(\nu_0) = i$, by induction hypothesis that means that exists an assignment $B' = \{x_1 = v_1, \ldots, x_{k_i-1} = v_{k_i-1}\}$ satisfying S such that $\mathbf{Path}_{B'}(\mu) = \nu_0$ and for all $v_s = 1$, $\{x_s = v_s\} \in A$. Then, the assignment $B = B' \cup \{x_{k_i} = \ldots = x_{k_{i+1}-1} = 0\}$ satisfies the required conditions.
    If $\neg b_{\nu_0} \vee \neg x_{k_{i-1}+j} \vee b_\nu$ propagated b_ν, then $\nu = \mathbf{Child}(\nu_0, j+1)$, $x_{k_{i-1}+j} \in A$ and b_{ν_0} has been propagated. As before, by induction hypothesis there exists an assignment $B' = \{x_1 = v_1, \ldots, x_{k_i-1} = v_{k_i-1}\}$ satisfying S such that $\mathbf{Path}_{B'}(\mu) = \nu_0$ and for all $v_s = 1$, $\{x_s = v_s\} \in A$. Take
    $$B = B' \cup \{x_{k_i} = \ldots = x_{k_i+j} = 1, x_{k_i+j+1} = \ldots = x_{k_{i+1}-1}\}.$$

[2] We can easily define an order on the partial assignments; for instance, using the lex order

B obviously satisfies S, $\mathbf{Path}_B(\mu) = \mathbf{Child}(\mathbf{Path}_{B'}(\mu), j) = \nu$ and for all $v_s = 1$, $\{x_s = v_s\} \in A$ since A is closed under unit propagation on S.

$\Leftarrow$ Assume that there exists an assignment B on the variables $x_1, \dots, x_{k_{i+1}-1}$ satisfying the required conditions. Let $j = v_{k_i} + v_{k_i+1} + \cdots + v_{k_{i+1}-1}$. Let $B' = \{x_s = v_s \in B \mid 1 \leq s < k_i\}$, and let $\nu_0 = \mathbf{Path}_{B'}(\mu)$. Then, since B' also satisfies the required conditions, by induction hypothesis b_{ν_0} has been propagated. Notice that $\nu = \mathbf{Child}(\nu_0, j)$.
If $j = 0$, clause $\neg b_{\nu_0} \vee b_\nu$ propagates b_ν. If $j > 0$, then $\{x_{k_i+j-1} = 1\} \in A$ since B satisfies the required conditions. Therefore, clause $\neg b_{\nu_0} \vee \neg x_{k_{i-1}+j-1} \vee b_\nu$ propagates b_ν.

– The result is analogous to the previous one.

□

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

Proof. Let A be a partial assignment on the variables $x_1, \dots, x_n$. We have to prove that:

1. If there exists no complete assignment $A' \supseteq A$ on variables $x_1, \dots, x_n$ satisfying $\mathbf{MDDChain}(1, C, S) \wedge S$, then unit propagation detects an inconsistency.
2. If there is such an assignment, but all of them contain $\{x_s = v_s\}$, then $\{x_s = v_s\}$ is propagated.

1. First of all let us assume that there is no complete assignment $A' \supseteq A$ on variables $x_1, \dots, x_n$ satisfying S. Since the encoding of S obviously enforces domain consistency of S, then unit propagation on $A \wedge S$ detects an inconsistency. We can assume, therefore, that A is consistent with S but it is not consistent with $\mathbf{MDDChain}(1, C, S) \wedge S$.
Let $\overline{A}$ be the set of literals obtained by unit propagation on $A \wedge S$. Then, $\overline{A}$ cannot be extended to a complete assignment satisfying $\mathbf{MDDChain}(1, C, S) \wedge S$. We have to prove that unit propagation detects an inconsistency.
Let us define
$$A' = \overline{A} \cup \{x_s = 0 \mid x_s = 1 \notin \overline{A}\}.$$
A' is a complete assignment with $A' \supseteq \overline{A} \supseteq A$ and A' satisfies S. By hypothesis, it does not satisfy $\mathbf{MDDChain}(1, C, S)$ so, $\mathbf{Path}_{A'}(\mu) = \mathcal{F}$. By the previous result, this means that unit propagation on $\overline{A} \wedge \mathbf{MDDChain}(1, C, S)$ propagates $\neg b_\mu$. Since b_μ is a unit clause, a contradiction is found.
2. Let us assume that A can be extended to a complete assignment satisfying $\mathbf{MDDChain}(1, C, S) \wedge S$, and that every of these extensions contain $\{x_{k_i+j} = v_{k_i+j}\}$ for a fix value v_{k_i+j}. Let $\overline{A}$ be the set of literals obtained by unit propagation on $A \wedge S$.
If every extension of A satisfying S contains $\{x_{k_i+j} = v_{k_i+j}\}$, since domain consistency on S is enforced, $\{x_{k_i+j} = v_{k_i+j}\} \in \overline{A}$ and the result holds. Let

us assume now that x_{k_i+j} is undefined on $\overline{A}$. By monotonicity of C, if $\overline{A}$ admits an extension satisfying $\mathbf{MDDChain}(1, C, S) \wedge S$, then

$$A' = \overline{A} \cup \{x_s = 0 \mid x_s = 1 \notin \overline{A}\}$$

satisfies $\mathbf{MDDChain}(1, C, S) \wedge S$. Notice in particular that $v_{k_i+j} = 0$.
Let us define the assignment

$$B = \{x_{k_{i'}+j'} = v_{k_{i'}+j'} \mid x_{k_{i'}+j'} = v_{k_{i'}+j'} \in A',\ 0 \le j' < d_{i'},\ 1 \le i' \le r, i' \neq i\} \cup \\ \cup \{x_{k_i} = \cdots = x_{k_i+j} = 1, x_{k_i+j+1} = \cdots = x_{k_{i+1}-1} = 0\}.$$

$B \supseteq \overline{A}$ since x_{k_i+j} is undefined on $\overline{A}$. B obviously satisfies S. Therefore, since $B \supseteq A$ and $\{x_{k_i+j} = v_{k_i+j}\} \notin B$, B does not satisfy $\mathbf{MDDChain}(1, C, S)$. That means that $\mathbf{Path}_B(\mu) = \mathcal{F}$. Let us define $B_1 = \{x_s = v_s \in B \mid s < k_i\}$, $B_2 = \{x_s = v_s \in B \mid s \ge k_{i+1}\}$, $\nu_1 = \mathbf{Path}_{B_1}(\mu)$ and $\nu_2 = \mathbf{Child}(\nu_1, j+1)$. Therefore, $\mathbf{Path}_{B_2}(\nu_2) = \mathbf{Path}_B(\mu) = \mathcal{F}$.
Notice that B_1 and B_2 satisfy the conditions of the previous Proposition. Therefore, b_{ν_1} and $\neg b_{\nu_2}$ are propagated. Therefore, clause $\neg b_{\nu_1} \vee \neg x_{k_i+j} \vee b_{\nu_2}$ propagates $\neg x_{k_i+j}$.

$\square$