Journal of Artificial Intelligence Research 50 (2014) 697-722 Submitted 10/13; published 07/14

MDD Propagation for Sequence Constraints

David Bergman DAVID.BERGMAN@BUSINESS.UCONN.EDU
School of Business, University of Connecticut
2100 Hillside Road, Unit 1041, Storrs, CT 06260

Andre A. Cire ACIRE@ANDREW.CMU.EDU
Willem-Jan van Hoeve VANHOEVE@ANDREW.CMU.EDU
Tepper School of Business, Carnegie Mellon University
5000 Forbes Avenue, Pittsburgh, PA 15213 USA

Abstract

We study propagation for the SEQUENCE constraint in the context of constraint programming based on limited-width MDDs. Our first contribution is proving that establishing MDD-consistency for SEQUENCE is NP-hard. Yet, we also show that this task is fixed parameter tractable with respect to the length of the sub-sequences. In addition, we propose a partial filtering algorithm that relies on a specific decomposition of the constraint and a novel extension of MDD filtering to node domains. We experimentally evaluate the performance of our proposed filtering algorithm, and demonstrate that the strength of the MDD propagation increases as the maximum width is increased. In particular, MDD propagation can outperform conventional domain propagation for SEQUENCE by reducing the search tree size and solving time by several orders of magnitude. Similar improvements are observed with respect to the current best MDD approach that applies the decomposition of SEQUENCE into AMONG constraints.

1. Introduction

The central inference process of constraint programming is constraint propagation (Rossi, van Beek, & Walsh, 2006; Dechter, 2003; Apt, 2003). While traditional constraint processing techniques were designed for explicitly defined relations of small arity, state-of-the-art constraint programming solvers apply specialized constraint propagation algorithms for global constraints of any arity, often based on efficient combinatorial methods such as network flows (van Hoeve & Katriel, 2006; Régin, 2011).

Conventional constraint propagation algorithms (or domain filtering algorithms) operate on individual constraints of a given problem. Their role is to identify and remove values in the variable domains that are inconsistent with respect to the constraint under consideration. Whenever the domain of a variable is updated (i.e., a value is removed), the constraints in which this variable appears can be reconsidered for inspection. This cascading process of propagating the changes in variable domains through the constraints continues until a fixed point is reached. Most constraint programming solvers assume that the variable domains are finite, which ensures termination of the constraint propagation process. Note that constraint propagation in itself may not be sufficient to determine the resolution of a given problem. Therefore, constraint propagation is normally applied at each search state in a systematic search process.

A major benefit of propagating variable domains is that it can be implemented efficiently in many cases. However, an inherent weakness of domain propagation is that it implicitly represents the Cartesian product of the variable domains as potential solution space. By communicating only domain changes, this limits the amount of information shared between constraints.

To address this shortcoming of domain propagation, Andersen, Hadzic, Hooker, and Tiedemann (2007) proposed the use of multi-valued decision diagrams (MDDs) as an alternative to variable domains in the context of constraint propagation. MDDs are directed acyclic layered graphs that can, in principle, compactly represent all solutions to a combinatorial problem (Wegener, 2000). Andersen et al. (2007) showed that MDDs of limited width can provide a much stronger relaxation of the solution space than the traditional Cartesian product of the variable domains, and as a consequence MDDs allow to represent and communicate more refined information between constraints. By propagating MDDs rather than variable domains, huge reductions in search tree size and computation time can be realized (Andersen et al., 2007; Hadzic, Hooker, O'Sullivan, & Tiedemann, 2008a; Hadzic, Hooker, & Tiedemann, 2008b; Hadzic, O'Mahony, O'Sullivan, & Sellmann, 2009; Hoda, van Hoeve, & Hooker, 2010; Cire & van Hoeve, 2012, 2013).

MDDs can be used to represent individual (global) constraints, subsets of constraints, or all constraints in a given problem. When representing individual constraints, as in the work of Hawkins, Lagoon, and Stuckey (2005) and Cheng and Yap (2008), the higher-level information carried by the MDD is lost when projecting this down to the variable domains for the traditional domain propagation. The highest potential for MDD propagation instead appears to be in representing specific subsets of constraints within the same MDD. That is, for a given set of constraints, we create and maintain one single limited-width MDD, which is then propagated through this constraint set. Since an MDD is defined with respect to a fixed variable ordering, it is most useful to select a subset of constraints compatible with this ordering. When applied in this way, MDD propagation can be implemented in parallel to the existing domain propagation in constraint programming systems, thus complementing and potentially strengthening the domain propagation process. For example, Cire and van Hoeve (2013) introduced MDD propagation for a subset of constraints representing disjunctive scheduling problems. They embedded this as a custom global constraint in the ILOG CP Optimizer constraint programming solver, which greatly improved the performance.

1.1 Methodology

Constraint propagation based on limited-width MDDs amounts to *MDD filtering* and *MDD refinement*. The role of an MDD filtering algorithm is to remove provably inconsistent arcs from the MDD (Hadzic et al., 2008b; Hoda et al., 2010). An MDD refinement algorithm on the other hand, aims at splitting nodes in the MDD to more accurately reflect the solution space (Hadzic et al., 2008a). In order to make this approach scalable and efficient, refinement algorithms must ensure that the MDD remains within a given maximum size (typically by restricting its maximum width—the number of nodes on any layer). By increasing this maximum width, the MDD relaxation can be strengthened to any desired level. That is, a maximum width of 1 would correspond to the traditional Cartesian product of the variable domains, while an infinite maximum width would correspond to an exact MDD representing all solutions. However, increasing the size of the MDD immediately impacts the computation time, and one typically needs to balance the trade-off between the strength of the MDD and the associated computation time.

In order to characterize the outcome of an MDD filtering algorithm, the notion of *MDD consistency* was introduced by Andersen et al. (2007), similar to domain consistency in finite-domain constraint programming: Given an MDD, a constraint is MDD consistent if all arcs in the MDD belong to at least one solution to the constraint. As a consequence of the richer data structure that an MDD represents, establishing MDD consistency may be more difficult than establishing domain consistency. For example, Andersen et al. show that establishing MDD consistency on the Alldifferent constraint is NP-hard, while establishing traditional domain consistency can be done in polynomial time (Régin, 1994).

1.2 Contributions

The main focus of this paper is the Sequence constraint, that is defined as a specific conjunction of Among constraints, where an Among constraint restricts the occurrence of a set of values for a sequence of variables to be within a lower and upper bound (Beldiceanu & Contejean, 1994). The Sequence constraint finds applications in, e.g., car sequencing and employee scheduling problems (Régin & Puget, 1997; van Hoeve, Pesant, Rousseau, & Sabharwal, 2009). It is known that classical domain consistency can be established for Sequence in polynomial time (van Hoeve, Pesant, Rousseau, & Sabharwal, 2006; van Hoeve et al., 2009; Brand, Narodytska, Quimper, Stuckey, & Walsh, 2007; Maher, Narodytska, Quimper, & Walsh, 2008; Downing, Feydy, & Stuckey, 2012). Furthermore, Hoda et al. (2010) present an MDD filtering algorithm for Among constraints establishing MDD consistency in polynomial time. However, it remained an open question whether or not MDD consistency for Sequence can be established in polynomial time as well.

In this work, we answer that question negatively and our first contribution is showing that establishing MDD consistency on the Sequence constraint is NP-hard. This is an important result from the perspective of MDD-based constraint programming. Namely, of all global constraints, the Sequence constraint has perhaps the most suitable combinatorial structure for an MDD approach; it has a prescribed variable ordering, it combines sub-constraints on contiguous variables, and existing approaches can handle this constraint fully by using bounds reasoning only.

As our second contribution, we show that establishing MDD consistency on the Sequence constraint is fixed parameter tractable with respect to the lengths of the sub-sequences (the Among constraints), provided that the MDD follows the order of the Sequence constraint. The proof is constructive, and follows from a generic algorithm to filter one MDD with another.

The third contribution is a partial MDD propagation algorithm for Sequence, that does not necessarily establish MDD consistency. It relies on the decomposition of Sequence into 'cumulative sums', and a new extension of MDD filtering to the information that is stored at its nodes.

Our last contribution is an experimental evaluation of our proposed partial MDD propagation algorithm. We evaluate the strength of our algorithm for MDDs of various maximum widths, and compare the performance with existing domain propagators for Sequence. We also compare our algorithm with the currently best known MDD approach that uses the natural decomposition of Sequence into Among constraints (Hoda et al., 2010). Our experiments demonstrate that MDD propagation can outperform domain propagation for Sequence by reducing the search tree size, and solving time, by several orders of magnitude. Similar results are observed with respect to MDD propagation of Among constraints. Our results thus provide further evidence for the power of MDD propagation in the context of constraint programming.

The remainder of this paper is structured as follows. In Section 2, we provide the necessary definitions of MDD-based constraint programming and the Sequence constraint. In Section 3, we present the proof that establishing MDD consistency on Sequence is NP-hard. Section 4 describes that establishing MDD consistency is fixed parameter tractable. In Section 5, the partial MDD filtering algorithm is presented. Section 6 shows the experimental results. We present final conclusions in Section 7.

2. Definitions

We first recall some basic definitions of MDD-based constraint programming, following the work of Andersen et al. (2007) and Hoda et al. (2010). In this work, an *ordered Multivalued Decision Diagram (MDD)* is a directed acyclic graph whose nodes are partitioned into $n+1$ (possibly empty) subsets or *layers* $L_1, \ldots, L_{n+1}$, where the layers $L_1, \ldots, L_n$ correspond respectively to variables $x_1, \ldots, x_n$. L_1 contains a single *root* node r, and L_{n+1} contains a single *terminal* node t. For a node u in the MDD, we let $L(u)$ denote the index of its layer. For an MDD M, the *width* $w(M)$ is the maximum number of nodes in a layer, or $\max_{i=1}^{n}\{|L_i|\}$. In MDD-based CP, the MDDs typically have a given fixed maximum width.

All arcs of the MDD are directed from an upper to a lower layer; that is, from a node in some L_i to a node in some L_j with $i < j$. For our purposes it is convenient to assume (without loss of generality) that each arc connects two adjacent layers. Each arc out of layer L_i is labeled with an element of the domain $D(x_i)$ of x_i. For an arc a, we refer to the label it represents as $\ell(a)$. For notational convenience, we also write $\ell(u, v)$ instead of $\ell((u, v))$ for an arc (u, v). An element in $D(x_i)$ appears at most once as a label on the arcs out of a given node $u \in L_i$. The set $A(u, v)$ of arcs from node u to node v may contain multiple arcs, and we denote each with its label. Let $A^{in}(u)$ denote the set of arcs coming into node u. We define the size of an MDD M by the number of its arcs, i.e., $|M| = \big|\{a \mid a \in A^{in}(u), u \in L_i, i = 2, \ldots, n+1\}\big|$.

An arc with label v leaving a node in layer i represents an assignment $x_i = v$. Each path in the MDD from r to t can be denoted by the arc labels $v_1, \ldots, v_n$ on the path and is identified with the solution $(x_1, \ldots, x_n) = (v_1, \ldots, v_n)$. A path $v_1, \ldots, v_n$ is *feasible* for a given constraint C if setting $(x_1, \ldots, x_n) = (v_1, \ldots, v_n)$ satisfies C. Constraint C is feasible on an MDD if the MDD contains a feasible path for C.

A constraint C is called *MDD consistent* on a given MDD if every arc of the MDD lies on some feasible path. Thus MDD consistency is achieved when all redundant arcs (i.e., arcs on no feasible path) have been removed. We also say that such MDD is MDD consistent with respect to C. Domain consistency for C is equivalent to MDD consistency on an MDD of width one that represents the variable domains. That is, it is equivalent to MDD consistency on an MDD in which each layer L_i contains a single node s_i, and $A(s_i, s_{i+1}) = D(x_i)$ for $i = 1, \ldots, n$.

Lastly, we formally recall the definitions of Among (Beldiceanu & Contejean, 1994), Sequence (Beldiceanu & Contejean, 1994), and Gen-Sequence (van Hoeve et al., 2009) constraints. The Among constraint counts the number of variables that are assigned to a value in a given set S, and ensures that this number is between a given lower and upper bound:

Definition 1 *Let X be a set of variables, l, u integer numbers such that $0 \leq l \leq u \leq |X|$, and $S \subset \cup_{x \in X} D(x)$ a subset of domain values. Then we define* $\textsc{Among}(X, l, u, S)$ *as*

$$l \leq \sum_{x \in X} (x \in S) \leq u.$$

Note that the expression $(x \in S)$ is evaluated as a binary value, i.e., resulting in 1 if $x \in S$ and 0 if $x \notin S$. The Sequence constraint is the conjunction of a given Among constraint applied to every sub-sequence of length q over a sequence of n variables:

Definition 2 *Let X be an ordered set of n variables, q, l, u integer numbers such that $0 \leq q \leq n$, $0 \leq l \leq u \leq q$, and $S \subset \cup_{x \in X} D(x)$ a subset of domain values. Then*

$$\textsc{Sequence}(X, q, l, u, S) = \bigwedge_{i=1}^{n-q+1} \textsc{Among}(s_i, l, u, S),$$

where s_i represents the sub-sequence $x_i, \ldots, x_{i+q-1}$.

Finally, the *generalized* Sequence constraint extends the Sequence constraint by allowing the Among constraints to be specified with different lower and upper bounds, and sub-sequence length:

Definition 3 *Let X be an ordered set of n variables, k a natural number, $\vec{s}$, $\vec{l}$, $\vec{u}$ vectors of length k such that s_i is a sub-sequence of X, $l_i, u_i \in \mathbb{N}$, $0 \leq l_i \leq u_i \leq n$ for $i = 1, 2, \ldots, k$, and $S \subset \cup_{x \in X} D(x)$ a subset of domain values. Then*

$$\textsc{Gen-Sequence}(X, \vec{s}, \vec{l}, \vec{u}, S) = \bigwedge_{i=1}^{k} \textsc{Among}(s_i, l_i, u_i, S).$$

3. MDD Consistency for Sequence is NP-Hard

As stated before, the only known non-trivial NP-hardness result for a global constraint in the context of MDD-based constraint programming is that of Andersen et al. (2007) for the Alldifferent constraint. A challenge in determining whether a global constraint can be made MDD consistent in polynomial time is that this must be guaranteed for *any* given MDD. That is, in addition to the combinatorics of the global constraint itself, the shape of the MDD adds another layer of complexity to establishing MDD consistency. For proving NP-hardness, a particular difficulty is making sure that in the reduction, the MDD remains of polynomial size. For Sequence constraints, so far it was unknown whether a polynomial-time MDD consistency algorithm exists. In this section we answer that question negatively and prove the following result.

Theorem 1 *Establishing MDD consistency for* Sequence *on an arbitrary MDD is NP-hard even if the MDD follows the variable ordering of the* Sequence *constraint.*

Proof. The proof is by reduction from 3-SAT, a classical NP-complete problem (Garey & Johnson, 1979). We will show that an instance of 3-SAT is satisfied if and only if a particular Sequence constraint on a particular MDD M of polynomial size has a solution. Therefore, establishing MDD consistency for Sequence on an arbitrary MDD is at least as hard as 3-SAT.

Consider a 3-SAT instance on n variables $x_1, \dots, x_n$, consisting of m clauses $c_1, \dots, c_m$. We first construct an MDD that represents the basic structure of the 3-SAT formula (see Example 1 after this proof for an illustration). We introduce binary variables $y_{i,j}$ and $\overline{y}_{i,j}$ representing the literals x_j and $\overline{x}_j$ per clause c_i, for $i = 1, \dots, m$ and $j = 1, \dots, n$ (x_j and $\overline{x}_j$ may or may not exist in c_i). We order these variables as a sequence Y, first by the index of the clauses, then by the index of the variables, and then by $y_{i,j}, \overline{y}_{i,j}$ for clause c_i and variable x_j. That is, we have $Y = y_{1,1}, \overline{y}_{1,1},\, y_{1,2}, \overline{y}_{1,2}, \dots, y_{1,n}, \overline{y}_{1,n},\, \dots,\, y_{m,1}, \overline{y}_{m,1},\, \dots, y_{m,n}, \overline{y}_{m,n}$. We construct an MDD M as a layered graph, where the k-th layer corresponds to the k-th variable in the sequence Y.

A clause c_i is represented by $2n$ consecutive layers corresponding to $y_{i,1}, \dots, \overline{y}_{i,n}$. In such part of the MDD, we identify precisely those paths that lead to a solution satisfying the clause. The basis for this is a 'diamond' structure for each pair of literals $(y_{i,j}, \overline{y}_{i,j})$, that assigns either $(0, 1)$ or $(1, 0)$ to this pair. If a variable does not appear in a clause, we represent it using such a diamond in the part of the MDD representing that clause, thus ensuring that the variable can take any assignment with respect to this clause. For the variables that do appear in the clause, we will explicitly list out all allowed combinations.

More precisely, for clause c_i, we first define a local root node r_i representing layer $L\left(y_{i,1}\right)$, and we set $tag(r_i)$ = 'unsat'. For each node u in layer $L\left(y_{i,j}\right)$ (for $j = 1, \dots, n$), we do the following. If variable x_j does not appear in c_i, or if $tag(u)$ is 'sat', we create two nodes v, v' in $L\left(\overline{y}_{i,j}\right)$, one single node w in $L\left(y_{i,j+1}\right)$, and arcs (u, v) with label 1, (u, v') with label 0, (v, w) with label 0, and (v', w) with label 1. This corresponds to the 'diamond' structure. We set $tag(w) = tag(u)$. Otherwise (i.e., $tag(u)$ is 'unsat' and $y_{i,j}$ appears in c_i), we create two nodes v, v' in $L\left(\overline{y}_{i,j}\right)$, two nodes w, w' in $L\left(y_{i,j+1}\right)$, and arcs (u, v) with label 1, (u, v') with label 0, (v, w) with label 0, and (v', w') with label 1. If c_i contains as literal $y_{i,j}$, we set $tag(w)$ = 'sat' and $tag(w')$ = 'unsat'. Otherwise (c_i contains $\overline{y}_{i,j}$), we set $tag(w)$ = 'unsat' and $tag(w')$ = 'sat'.

This procedure will be initialized by a single root node r representing $L\left(y_{11}\right)$. We iteratively append the MDDs of two consecutive clauses c_i and c_{i+1} by merging the nodes in the last layer of c_i that are marked 'sat' into a single node, and let this node be the local root for c_{i+1}. We finalize the procedure by merging all nodes in the last layer that are marked 'sat' into the single terminal node t. By construction, we ensure that only one of y_{ij} and $\overline{y}_{ij}$ can be set to 1. Furthermore, the variable assignment corresponding to each path between layers $L\left(y_{i,1}\right)$ and $L\left(y_{i+1,1}\right)$ will satisfy clause c_i, and exactly n literals are chosen accordingly on each such path.

We next need to ensure that for a feasible path in the MDD, each variable x_j will correspond to the same literal $y_{i,j}$ or $\overline{y}_{i,j}$ in each clause c_i. To this end, we impose the

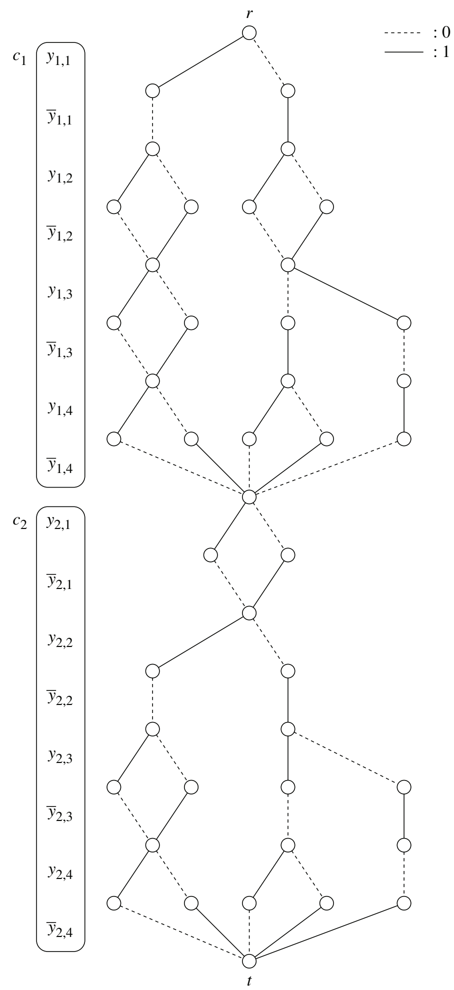


Figure 1: The MDD corresponding to Example 1.

constraint

$$\textsc{Sequence}(Y, q = 2n, l = n, u = n, S = \{1\}) \tag{1}$$

on the MDD M described above. If the sub-sequence of length $2n$ starts from a positive literal $y_{i,j}$, by definition there are exactly n variables that take value 1. If the sub-sequence starts from a negative literal $\overline{y}_{i,j}$ instead, the last variable in the sequence corresponds to the value x_j in the next clause c_{i+1}, i.e., $y_{i+1,j}$. Observe that all variables except for the first and the last in this sequence will take value 1 already $n-1$ times. Therefore, of the first and the last variable in the sequence (which represent x_j and its complement $\overline{x}_j$ in any order), only one can take the value 1. That is, x_j must take the same value in clause c_i and c_{i+1}. Since this holds for all sub-sequences, all variables x_j must take the same value in all clauses.

The MDD M contains $2mn + 1$ layers, while each layer contains at most six nodes. Therefore, it is of polynomial size (in the size of the 3-SAT instance), and the overall construction needs polynomial time. □

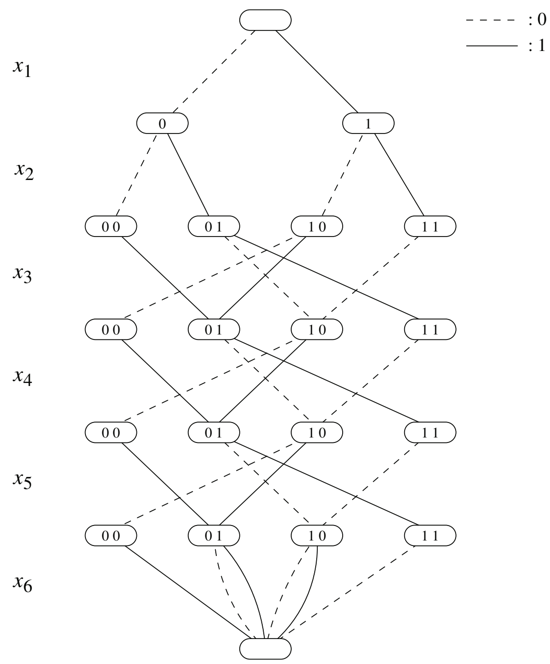


Figure 2: The exact MDD for the Sequence constraint of Example 2.

Example 1 *Consider the 3-SAT instance on four Boolean variables* x_1, x_2, x_3, x_4 *with clauses* $c_1 = (x_1 \vee \overline{x}_3 \vee x_4)$ *and* $c_2 = (x_2 \vee x_3 \vee \overline{x}_4)$*. The corresponding MDD used in the reduction is given in Figure 1.*

4. MDD Consistency for Sequence is Fixed Parameter Tractable

In this section we show that establishing MDD consistency for Sequence on an arbitrary MDD is fixed parameter tractable, with respect to the length of the sub-sequences q. It was already shown by van Hoeve et al. (2006, 2009) that an exact MDD for the Sequence constraint exists with $O(n2^q)$ nodes (i.e., the 'unfolded' automaton of the Regular constraint), as illustrated in the next example.

Example 2 *Consider the constraint* Sequence$(X, q = 3, l = 1, u = 2, S = \{1\})$ *where* $X = \{x_1, x_2, \ldots, x_6\}$ *is an ordered set of binary variables. The corresponding exact MDD, following the order of* X*, is presented in Figure 2. For convenience, each node in the MDD is labeled with the last* $q-1$ *labels that represent the sub-sequence up to that node (starting* $q-1$ *layers up). For example, the second node in the third layer represents decisions* $x_1 = 0$ *and* $x_2 = 1$*, corresponding to sub-sequence* 01*. To construct the next layer, we either append a 0 or a 1 to this sub-sequence (and remove the first symbol), leading to nodes labeled* 10 *and* 11*, respectively. Note that from nodes labeled* 00 *we must take an arc with label 1, because* $l = 1$*. Similarly for nodes labeled* 11 *we must take an arc with label 0, because* $u = 2$*. After* q

layers, all possible sub-sequences have been created (maximally $O(2^{q-1})$), which thus defines the width of the subsequent layers.

However, since we are given an arbitrary MDD, and not necessarily an exact MDD, we need some additional steps to exploit this connection. For this we apply a generic approach that will not only show fixed parameter tractability for Sequence, but in fact can be applied to determine whether MDD consistency is tractable for any constraint.

Our goal is to establish MDD consistency on a given MDD M with respect to another MDD M' on the same set of variables. This is compatible with our earlier definitions since M' can be interpreted to define a constraint. That is, M is MDD consistent with respect to M' if every arc in M belongs to a path (solution) that also exists in M'. For our purposes, we assume that M and M' follow the same variable ordering.

We can establish MDD consistency by first taking the intersection of M and M', and then removing all arcs from M that are not compatible with the intersection. Computing the intersection of two MDDs is well-studied, and we present a top-down intersection algorithm that follows our definitions in Algorithm 1. This description is adapted from the 'melding' procedure presented by Knuth (2009).

The intersection MDD, denoted by I, represents all possible paths (solutions) that are present both in M and M'. Each *partial* path in I from the root r^I to a node u thus will exist in M and M', with respective endpoints v, v'. This information is captured by associating with each node u in I a state $s(u) = (v, v')$ representing those nodes $v \in M$ and $v' \in M'$. The root of I is initialized as r^I with $s(r^I) := (r, r')$ where r and r' are the respective roots of M and M' (lines 1-2). The algorithm then, in a top-down traversal, considers a layer L_i^I in I, and augments a node $u \in L_i^I$ with $s(u) = (v, v')$ with an arc only if both M and M' have an arc with the same label out of v and v' respectively (lines 5-7). If the next layer already contains a node $\tilde{u}$ with the same state we re-use that node. Otherwise we add a new node $\tilde{u}$ to L_{i+1}^I and add the arc $(u, \tilde{u})$ to I. Note that the last layer of I contains a single terminal t^I with state $s(t^I) = (t, t')$, provided that I is not empty. In the last step (line 14) we clean up I by removing all arcs and nodes that do not belong to a feasible path. This can be done in a bottom-up traversal of I. Observe that this algorithm does not necessarily create a *reduced* MDD.

Algorithm 2 presents an algorithm to establish MDD-consistency on M with respect to M'. We first compute the intersection I of M and M' (line 1). We then traverse M in a top-down traversal, and for each layer L_i^M we identify and remove infeasible arcs. For this, we define a Boolean array Support$[u, l]$ (initialized to 0) that represents whether an arc out of node $u \in M$ with label l has support in I (line 3). In line 4, we consider all arcs out of layer L_i^I in I. If an arc $a = (v, \tilde{v})$ exists in L_i^I with label l and $s(v) = (u, u')$, we mark the associated arc out of u as supported by setting Support$[u, l] := 1$ (lines 4-6). We then remove all arcs out of L_i^M that have no support (lines 7-9). Lastly, we again clean up M by removing all arcs and nodes that do not belong to a feasible path (line 11).

Theorem 2 *Algorithm 2 establishes MDD-consistency on M with respect to M' in $O(|M| \cdot w(M')$ time and space.*

Proof. The correctness of Algorithm 1 follows by induction on the number of layers. To prove that Algorithm 2 establishes MDD-consistency, consider an arc $a = (u, \tilde{u})$ in M after

Algorithm 1 Intersection(M,M')

Input: MDD M with root r, MDD M' with root r'. M and M' are defined on the same ordered sequence of n variables.
Output: MDD I with layers $L_1^I, \dots, L_{n+1}^I$ and arc set A^I. Each node u in I has an associated state $s(u)$.

create node r^I with state $s(r^I) := (r, r')$
$L_1^I := \{r^I\}$
for $i = 1$ **to** n **do**
  $L_{i+1}^I := \{\}$
  for all $u \in L_i^I$ with $s(u) = (v, v')$ **do**
    for all $a = (v, \tilde{v}) \in M$ and $a' = (v', \tilde{v}') \in M'$ such that $\ell(a) = \ell(a')$ **do**
      create node $\tilde{u}$ with state $s(\tilde{u}) := (\tilde{v}, \tilde{v}')$
      if $\exists\, \tilde{w} \in L_{j+1}^I$ with $s(\tilde{w}) = s(\tilde{u})$ **then** $\tilde{u} := \tilde{w}$
      else L_{i+1}^I += $\tilde{u}$ **end if**
      add arc $(u, \tilde{u})$ with label $\ell(a)$ to arc set A^I
    end for
  end for
end for
remove all arcs and nodes from I that are not on a path from r^I to $t^I \in L_{n+1}^I$
return I

Algorithm 2 MDD-Consistency(M,M')

Input: MDD M with root r, MDD M' with root r'. M and M' are defined on the same ordered sequence of n variables.
Output: M that is MDD-consistent with respect to M'

create $I :=$ Intersection(M,M')
for $i = 1$ **to** n **do**
  create array Support$[u, l] := 0$ for all $u \in L_i^M$ and arcs out of u with label l
  for all arcs $a = (v, \tilde{v})$ in A^I with $s(v) = (u, u')$ such that $v \in L_i^I$ **do**
    Support$[u, \ell(a)] := 1$
  end for
  for all arcs $a = (u, \tilde{u})$ in M such that $u \in L_i^M$ **do**
    if Support$[u, \ell(a)] = 0$ **then** remove a from M **end if**
  end for
end for
remove all arcs and nodes from M that are not on a path from r to $t \in L_{n+1}^M$
return M

applying the algorithm. There exists a node $v \in I$ with $s(v) = (u, u')$ such that solutions represented by the paths from r to u in M and from r' to u' in M' are equivalent. There also exists an arc $a^I = (v, \tilde{v}) \in A^I$ with the same label as a. Consider $s(\tilde{v}) = (w, w')$. Since M and I are decision diagrams, a label appears at most once on an arc out of a node. Therefore, $w = \tilde{u}$. Since a^I belongs to I, there exist paths from w (or $\tilde{u}$) to t in M and from w' to t' in M' that are equivalent. Hence, a belongs to a feasible path in M (from r to u, then along a into $\tilde{u}$ and terminating in t) for which an equivalent path exists in M' (from r' to u', then into w' and terminating in t').

Regarding the time complexity for computing the intersection, a coarse upper bound multiplies n (line 3), $w(M) \cdot w(M')$ (line 5), and $d_{\max}^2$ (line 6), where $d_{\max}$ represents the maximum degree out of a node, or $\max_{x \in X} |D(x)|$. We can amortize these steps since the for-loops in lines 3 and 6 consider each arc in M once for comparison with arcs in M'. Each arc is compared with at most $w(M')$ arcs (line 6); here we assume that we can check in constant time whether a node has an outgoing arc with a given label (using an arc-label list). This gives a total time complexity of $O(|M| \cdot w(M'))$. The memory requirements are bounded by the size of the intersection, which is at most $O(n \cdot w(M) \cdot w(M') \cdot d_{\max}) = O(|M| \cdot w(M'))$. This dominates the complexity of Algorithm 2, since lines 2-12 can be performed in linear time and space (in the size of M). □

Observe that Algorithm 2 no longer ensures that each *solution* in M is represented by some path in M', as is the case for the intersection. MDD-consistency merely establishes that each *arc* in M belongs to some solution that is also in M'. Although MDD intersections are stronger than MDD consistency, their limitation is that the width of the intersection MDD may be as large as the product of the widths of M and M'. Therefore intersecting M with multiple MDDs will, in general, increase the size of the resulting MDD exponentially.

We next apply Theorem 2 to the Sequence constraint.

Corollary 1 *Let X be an ordered sequence of variables, $C =$ Sequence(X, q, l, u, S) a sequence constraint, and M an arbitrary MDD following the variable ordering of X. Establishing MDD consistency for C on M is fixed parameter tractable with respect to parameter q.*

Proof. We know that there exists an exact MDD M' of size $O(n2^{q-1})$ that represents C (van Hoeve et al., 2006, 2009). Applying Theorem 2 gives an MDD-consistency algorithm with time and space complexity $O(|M|\, 2^{q-1})$, and the result follows. □

We note that Theorem 2 can also be applied to obtain the tractability of establishing MDD consistency on other constraints. Consider for example the constraint Among$(x_1, x_2, \ldots, x_n, l, u, S)$. For any variable ordering, we can construct an exact MDD in a top-down procedure by associating with each node v the number of variables taking a value in S along the path from r to v, representing the 'length' of that path. Nodes with the same length are equivalent and can be merged. Because the largest layer has at most $u + 1$ different path lengths, the exact MDD has size $O(nu)$, and by Theorem 2 establishing MDD consistency is tractable for Among. Indeed, Hoda et al. (2010) also showed that MDD consistency can be established for this constraint, with quadratic time complexity.

The converse of Theorem 2 does not hold: There exist constraints for which MDD consistency can be established in polynomial time on any given MDD, while a minimal reduced exact MDD has exponential size. As a specific example, consider linear inequality constraints of the form $\sum_{i=1}^{n} a_i x_i \geq b$ where x_i is an integer variable, a_i is a constant, for $i = 1, \ldots, n$, and b is a constant. MDD consistency can be established for such constraints in linear time, for any given MDD, by computing for each arc the longest r-t path (relative to the coefficients a_i) that uses that arc (Andersen et al., 2007). However, Hosaka, Takenaga, Kaneda, and Yajima (1997) provide the following explicit linear inequality. For k even and $n = k^2$, consider $\sum_{1 \leq i,j \leq k} a_{ij} x_{ij} \geq k(2^{2k} - 1)/2$, where x_{ij} is a binary variable, and $a_{ij} = 2^{i-1} + 2^{k+j-1}$, for $1 \leq i, j \leq k$. They show that, for any variable order, the size of the reduced ordered BDD for this inequality is bounded from below by $\Omega(2^{\sqrt{n}/2})$.

5. Partial MDD Filtering for Sequence

In many practical situations the value of q will lead to prohibitively large exact MDDs for establishing MDD consistency, which limits the applicability of Corollary 1. Therefore we next explore a more practical partial filtering algorithm that is polynomial also in q.

One immediate approach is to propagate the Sequence constraint in MDDs through its natural decomposition into Among constraints, and apply the MDD filtering algorithms for Among proposed by Hoda et al. (2010). However, it is well-known that for classical constraint propagation based on variable domains, the Among decomposition can be substantially improved by a dedicated domain filtering algorithm for Sequence (van Hoeve et al., 2006, 2009; Brand et al., 2007; Maher et al., 2008). Therefore, our goal in this section is to provide MDD filtering for Sequence that can be stronger in practice than MDD filtering for the Among decomposition, and stronger than domain filtering for Sequence. In what follows, we assume that the MDD at hand respects the ordering of the variables in the Sequence constraint.

5.1 Cumulative Sums Encoding

Our proposed algorithm extends the original domain consistency filtering algorithm for Sequence by van Hoeve et al. (2006) to MDDs, following the 'cumulative sums' encoding as proposed by Brand et al. (2007). This representation takes the following form. For a sequence of variables $X = x_1, x_2, \ldots, x_n$, and a constraint Sequence(X, q, l, u, S), we first introduce variables $y_0, y_1, \ldots, y_n$, with respective initial domains $D(y_i) = [0, i]$ for $i = 1, \ldots, n$. These variables represent the cumulative sums of X, i.e., y_i represents $\sum_{j=1}^{i} (x_j \in S)$ for $i = 1, \ldots, n$. We now rewrite the Sequence constraint as the following system of constraints:

$$y_i = y_{i-1} + \delta_S(x_i) \qquad \forall i \in \{1, \ldots, n\}, \tag{2}$$

$$y_{i+q} - y_i \geq l \qquad \forall i \in \{0, \ldots, n-q\}, \tag{3}$$

$$y_{i+q} - y_i \leq u \qquad \forall i \in \{0, \ldots, n-q\}, \tag{4}$$

where $\delta_S : X \to \{0, 1\}$ is the indicator function for the set S, i.e., $\delta_S(x) = 1$ if $x \in S$ and $\delta_S(x) = 0$ if $x \notin S$. Brand et al. show that establishing singleton bounds consistency on this system suffices to establish domain consistency for the original Sequence constraint.

In order to apply similar reasoning in the context of MDDs, the crucial observation is that the domains of the variables $y_0, \dots, y_n$ can be naturally represented at the *nodes* of the MDD. In other words, a node v in layer L_i represents the domain of y_{i-1}, restricted to the solution space formed by all r-t paths containing v. Let us denote this information for each node v explicitly as the interval $[\mathrm{lb}(v), \mathrm{ub}(v)]$, and we will refer to it as the 'node domain' of v. Following the approach of Hoda et al. (2010), we can compute this information in linear time by one top-down pass, by using equation (2), as follows:

$$\begin{aligned} \mathrm{lb}(v) &= \min_{(u,v)\in A^{in}(v)} \left\{\mathrm{lb}(u) + \delta_S\left(\ell(u,v)\right)\right\}, \\ \mathrm{ub}(v) &= \max_{(u,v)\in A^{in}(v)} \left\{\mathrm{ub}(u) + \delta_S\left(\ell(u,v)\right)\right\}, \end{aligned} \tag{5}$$

for all nodes $v \neq r$, while $[\mathrm{lb}(r), \mathrm{ub}(r)] = [0, 0]$.

As the individual Among constraints are now posted as $y_{i+q} - y_i \geq l$ and $y_{i+q} - y_i \leq u$, we also need to compute for a node v in layer L_{i+1} all its ancestors from layer L_i. This can be done by maintaining a vector $\mathcal{A}_v$ of length $q+1$ for each node v, where $\mathcal{A}_v[i]$ represents the set of ancestor nodes of v at the i-th layer above v, for $i = 0, \dots, q$. We initialize $\mathcal{A}_r = [\{r\}, \varnothing, \dots, \varnothing]$, and apply the recursion

$$\begin{aligned} &\mathcal{A}_v[i] = \cup_{(u,v)\in A^{in}(v)} \mathcal{A}_u[i-1] \qquad \text{for } i = 1, 2, \dots, q, \\ &\mathcal{A}_v[0] = \{v\}. \end{aligned}$$

The resulting top-down pass itself takes linear time (in the size of the MDD), while a direct implementation of the recursive step for each node takes $O(q \cdot (w(M))^2)$ operations for an MDD M. Now, the relevant ancestor nodes for a node v in layer L_{i+q} are stored in $\mathcal{A}_v[q]$, a subset of layer L_i. We similarly compute all descendant nodes of v in a vector $\mathcal{D}_v$ of length $q+1$, such that $\mathcal{D}_v[i]$ contains all descendants of v in the i-th layer below v, for $i = 0, 1, \dots, q$. We initialize $\mathcal{D}_t = [\{t\}, \varnothing, \dots, \varnothing]$.

However, for our purposes we only need to maintain the minimum and maximum value of the union of the domains of $\mathcal{A}_v$, resp., $\mathcal{D}_v$, because constraints (3) and (4) are inequalities; see the application of $\mathcal{A}_v$ and $\mathcal{D}_v$ in rules (8) below. This makes the recursive step more efficient, now taking $O(qw(M))$ operations per node.

Alternatively, we can approximate this information by only maintaining a minimum and maximum node domain value for each *layer*, instead of a list of ancestor layers. This will compromise the filtering, but may be more efficient in practice, as it only requires to maintain two integers per layer.

5.2 Processing the Constraints

We next process each of the constraints (2), (3), and (4) in turn to remove provably inconsistent arcs, while at the same time we filter the node information.

Starting with the ternary constraints of type (2), we remove an arc (u, v) if $\mathrm{lb}(u) + \delta_S(\ell(u, v)) > ub(v)$. Updating $[\mathrm{lb}(v), \mathrm{ub}(v)]$ for a node v is done similar to the rules (5) above:

$$\begin{aligned} \mathrm{lb}(v) &= \max\left\{\mathrm{lb}(v), \min_{(u,v)\in A^{in}(v)} \left\{\mathrm{lb}(u) + \delta_S(\ell(u,v))\right\}\right\}, \\ \mathrm{ub}(v) &= \min\left\{\mathrm{ub}(v), \min_{(u,v)\in A^{in}(v)} \left\{\mathrm{ub}(u) + \delta_S(\ell(u,v))\right\}\right\}, \end{aligned} \tag{6}$$

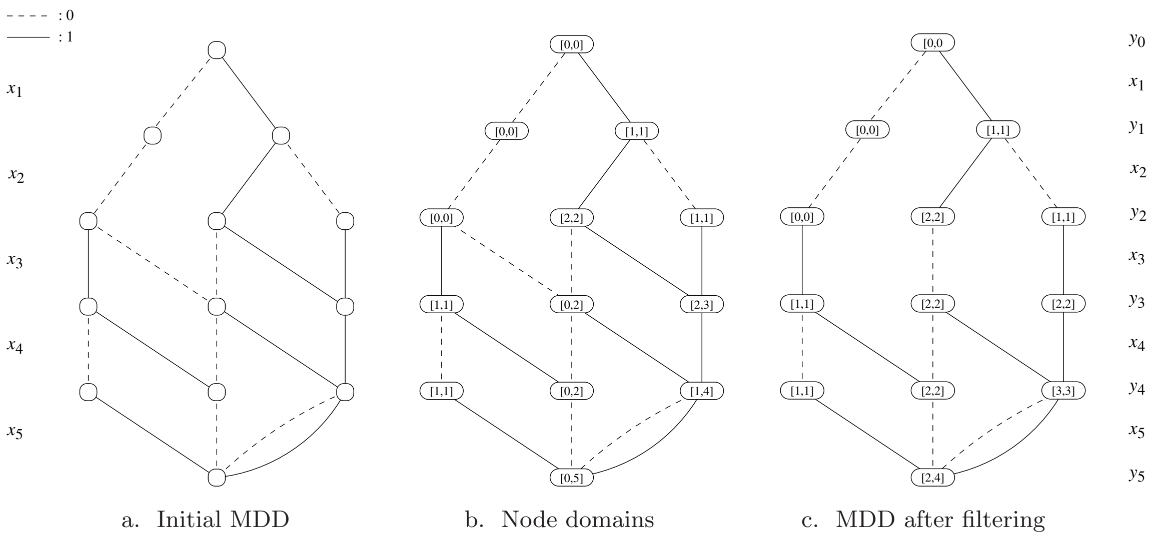


Figure 3: MDD propagation for the constraint Sequence$(X, q = 3, l = 1, u = 2, S = \{1\})$ of Example 3.

In fact, the resulting algorithm is a special case of the MDD consistency equality propagator of Hadzic et al. (2008a), and we thus inherit the MDD consistency for our ternary constraints.

Next, we process the constraints (3) and (4) for a node v in layer L_{i+1} ($i = 0, \ldots, n$). Recall that the relevant ancestors from L_{i+1-q} are $\mathcal{A}_v[q]$, while its relevant descendants from L_{i+1+q} are $\mathcal{D}_v[q]$. The variable corresponding to node v is y_i, and it participates in four constraints:

$$\begin{aligned} y_i &\geq l + y_{i-q}, \\ y_i &\leq u + y_{i-q}, \\ y_i &\leq y_{i+q} - l, \\ y_i &\geq y_{i+q} - u. \end{aligned} \tag{7}$$

Observe that we can apply these constraints to filter *only* the node domain $[\mathrm{lb}(v), \mathrm{ub}(v)]$ corresponding to y_i. Namely, the node domains corresponding to the other variables y_{i-q} and y_{i+q} may find support from nodes in layer L_{i+1} other than v. We update $\mathrm{lb}(v)$ and $\mathrm{ub}(v)$ according to equations (7):

$$\begin{aligned} \mathrm{lb}(v) &= \max\{\ \mathrm{lb}(v),\quad l + \min_{u \in \mathcal{A}_v[q]} \mathrm{lb}(u),\quad \min_{w \in \mathcal{D}_v[q]} \mathrm{lb}(w) - u\ \}, \\ \mathrm{ub}(v) &= \min\{\ \mathrm{ub}(v),\quad u + \max_{u \in \mathcal{A}_v[q]} \mathrm{ub}(u),\quad \max_{w \in \mathcal{D}_v[q]} \mathrm{ub}(w) - l\ \}. \end{aligned} \tag{8}$$

The resulting algorithm is a specific instance of the generic MDD consistent binary constraint propagator presented by Hoda et al. (2010), and again we inherit the MDD consistency for these constraints. We can process the constraints in linear time (in the size of the MDD) by a top-down and bottom-up pass through the MDD.

Example 3 *Consider the constraint* Sequence$(X, q = 3, l = 1, u = 2, S = \{1\})$ *with the ordered sequence of binary variables* $X = \{x_1, x_2, x_3, x_4, x_5\}$. *Assume we are given the MDD in Figure 3.a. In Figure 3.b. we show the node domains that result from processing rules (5). Figure 3.c. shows the resulting MDD after processing the constraints via the rules (6) and (8). For example, consider the middle node in the fourth layer, corresponding to variable* y_3. *Let this node be* v. *It has initial domain* $[0, 2]$, *and* $\mathcal{A}_v[q]$ *only contains the root node, which has domain* $[0, 0]$. *Since* $l = 1$, *we can reduce the domain of* v *to* $[1, 2]$. *We can next consider the arcs into* v, *and conclude that value 1 in its domain is not supported. This further reduces the domain of* v *to* $[2, 2]$, *and allows us to eliminate one incoming arc (from the first node of the previous layer).*

The resulting MDD in Figure 3.c. reflects all possible deductions that can be made by our partial algorithm. We have not established MDD consistency however, as witnessed by the infeasible path $(1, 1, 0, 0, 0)$.

Observe that our proposed algorithm can be applied immediately to the more general Gen-Sequence constraints in which each Among constraint has its individual l, u and q. The cumulative sums encoding can be adjusted in a straightforward manner to represent these different values.

5.3 Formal Analysis

We next formally compare the outcome of our partial MDD filtering algorithm with MDD propagation for the Among encoding and domain propagation for Sequence. First, we recall the following theorem.

Theorem 3 (Brand et al., 2007, Thm. 4) *Bounds consistency on the cumulative sums encoding is incomparable to bounds consistency on the* Among *encoding of* Sequence.

Note that since all variable domains in the Among and cumulative sums encoding are ranges (intervals of integer values), bounds consistency is equivalent to domain consistency.

Corollary 2 *MDD consistency on the cumulative sums encoding is incomparable to MDD consistency on the* Among *encoding of* Sequence.

Proof. We apply the examples from the proof of Theorem 4 in the work of Brand et al.. Consider the constraint Sequence$(X, q = 2, l = 1, u = 2, S = \{1\})$ with the ordered sequence of binary variables $X = \{x_1, x_2, x_3, x_4\}$ having domains $D(x_i) = \{0, 1\}$ for $i = 1, 2, 4$, and $D(x_3) = \{0\}$. We apply the 'trivial' MDD of width 1 representing the Cartesian product of the variable domains. Establishing MDD consistency on the cumulative sums encoding yields

$$y_0 \in [0, 0], y_1 \in [0, 1], y_2 \in [1, 2], y_3 \in [1, 2], y_4 \in [2, 3],$$
$$x_1 \in \{0, 1\}, x_2 \in \{0, 1\}, x_3 \in \{0\}, x_4 \in \{0, 1\}.$$

Establishing MDD consistency on the Among encoding, however, yields

$$x_1 \in \{0, 1\}, \mathbf{x_2} \in \{\mathbf{1}\}, x_3 \in \{0\}, \mathbf{x_4} \in \{\mathbf{1}\}.$$

Consider the constraint Sequence($X, q = 3, l = 1, u = 1, S = \{1\}$) with the ordered sequence of binary variables $X = \{x_1, x_2, x_3, x_4\}$ having domains $D(x_i) = \{0, 1\}$ for $i = 2, 3, 4$, and $D(x_1) = \{0\}$. Again, we apply the MDD of width 1 representing the Cartesian product of the variable domains. Establishing MDD consistency on the cumulative sums encoding yields

$$y_0 \in [0,0], y_1 \in [0,0], y_2 \in [0,1], y_3 \in [1,1], y_4 \in [1,1],$$
$$x_1 \in \{0\}, x_2 \in \{0,1\}, x_3 \in \{0,1\}, \mathbf{x_4} \in \{\mathbf{0}\},$$

while establishing MDD consistency on the Among encoding does not prune any value. □

As an additional illustration of Corollary 2, consider again Example 3 and Figure 3. MDD propagation for the Among encoding will eliminate the value $x_4 = 0$ from the infeasible path $(1, 1, 0, 0, 0)$, whereas our example showed that MDD propagation for cumulative sums does not detect this.

Theorem 4 *MDD consistency on the cumulative sums encoding of* Sequence *is incomparable to domain consistency on* Sequence*.*

Proof. The first example in the proof of Corollary 2 also shows that domain consistency on Sequence can be stronger than MDD consistency on the cumulative sums encoding.

To show the opposite, consider a constraint Sequence($X, q, l, u, S = \{1\}$) with a set of binary variables of arbitrary size, arbitrary values q, l, and $u = |X| - 1$. Let M be the MDD defined over X consisting of two disjoint paths from r to t: the arcs on one path all have label 0, while the arcs on the other all have value 1. Since the projection onto the variable domains gives $x \in \{0, 1\}$ for all $x \in X$, domain consistency will not deduce infeasibility. However, establishing MDD consistency with respect to M on the cumulative sums encoding will detect this. □

Even though formally our MDD propagation based on cumulative sums is incomparable to domain propagation of Sequence and MDD propagation of Among constraints, in the next section we will show that in practice our algorithm can reduce the search space by orders of magnitude compared to these other methods.

6. Computational Results

The purpose of our computational results is to evaluate empirically the strength of the partial MDD propagator described in Section 5. We perform three main comparisons. First, we want to assess the impact of increasing the maximum width of the MDD on the filtering. Second, we want to compare the MDD propagation with the classical domain propagation for Sequence. In particular, we wish to evaluate the computational overhead of MDD propagation relative to domain propagation, and to what extent MDD propagation can outperform domain propagation. Third, we compare the filtering strength of our MDD propagator for Sequence to the filtering strength of the MDD propagators for the individual Among constraints, being the best MDD approach for Sequence so far (Hoda et al., 2010).

We have implemented our MDD propagator for Sequence as a custom global constraint in IBM ILOG CPLEX CP Optimizer 12.4, using the C++ interface. Recall from Section 5 that for applying rules (8) we can either maintain a minimum and maximum value for the q previous ancestors and descendants of each node, or approximate this by maintaining these values simply for each layer. We evaluated both strategies and found that the latter did reduce the amount of filtering, but nonetheless resulted in much more efficient performance (about twice as fast on average). Hence, the reported results use that implementation.

For the MDD propagator for Among, we apply the code of (Hoda et al., 2010). For the domain propagation, we applied three models. The first uses the domain consistent propagator for Sequence by van Hoeve et al. (2009), running in $O(n^3)$ time. The second uses the domain consistent propagator for Sequence based on a network flow representation by Maher et al. (2008), which runs in $O(n^2)$ time.[1] As third model, we applied the decomposition into cumulative sums, which uses no explicit global constraint for Sequence. Propagating this decomposition also takes $O(n^2)$ in the worst case, as it considers $O(n)$ variables and constraints while the variable domains contain up to n elements. We note that for almost all test instances, the cumulative sums encoding established domain consistency on Sequence. As an additional advantage, the cumulative sums encoding permits a more insightful comparison with our MDD propagator, since both are based on the cumulative sums decomposition.

We note that Brand et al. (2007) introduce the 'multiple-Sequence' constraint that represents the conjunction of multiple Sequence constraints on the same set of ordered variables (as in our experimental setup). Narodytska (2011) shows that establishing bounds consistency on such system is already NP-hard, and presents a domain consistent propagator that encodes the system as an automaton for the Regular constraint. The algorithm runs in $O(nm^q)$ time, where n represents the number of variables, m the number of Sequence constraints, and q the length of the largest subsequence.

In order to compare our algorithms with the multiple-Sequence constraint, we conducted experiments to identify a suitable testbed. We found that instances for which the multiple-Sequence constraint would not run out of memory could be solved instantly by using any domain propagator for the individual Sequence constraints, while creating the data structures for the multiple-Sequence constraint took substantially more time on average. For instances that were more challenging (as described in the next sections), the multiple-Sequence constraint could not be applied due to memory issues. We therefore excluded this algorithm from the comparisons in the sections below.

Because single Sequence constraints can be solved in polynomial time, we consider instances with multiple Sequence constraints in our experiments. We assume that these are defined on the same ordered set of variables. To measure the impact of the different propagation methods correctly, all approaches apply the same fixed search strategy, i.e., following the given ordering of the variables, with a lexicographic value ordering heuristic. For each method, we measure the number of backtracks from a failed search state as well as the solving time. All experiments are performed using a 2.33GHz Intel Xeon machine.

1. We thank Nina Narodytska for sharing the implementation with us.

6.1 Systems of Sequence Constraints

We first consider systems of multiple Sequence constraints that are defined on the same set of variables. We generate instances with $n = 50$ variables each having domain $\{0, 1, \ldots, 10\}$, and 5 Sequence constraints. For each Sequence constraint, we set the length of subsequence uniform randomly between $[5, n/2)$ as

$$q = (\texttt{rand()}\%((n/2) - 5)) + 5.$$

Here, rand() refers to the standard C++ random number generator, i.e., rand()$\%k$ selects a number in the range $[0, k-1]$. Without the minimum length of 5, many of the instances would be very easy to solve by either method. We next define the difference between l and u as $\Delta := (\texttt{rand()}\%q)$, and set

$$l := (\texttt{rand()}\%(q - \Delta)),$$
$$u := l + \Delta.$$

Lastly, we define the set of values S by first defining its cardinality as (rand()$\%11) + 1$, and then selecting that many values uniformly at random from $\{0, 1, \ldots, 10\}$. We generated 250 such instances in total.[2]

We solve each instance using the domain consistency propagator for Sequence, the cumulative sums encoding (domain propagation), and the MDD propagator with maximum widths 2, 4, 8, 16, 32, 64, 128. Each method is given a maximum time limit of 1,800 seconds per instance.

We compare the performance of domain propagation and MDD propagation in Figure 4. In this figure, we report for each given time point how many instances could be solved within that time by a specific method. The three domain propagation methods are represented by 'Cumulative Sums' (the cumulative sums decomposition), 'Sequence - HPRS' (the Sequence propagator in van Hoeve et al., 2006, 2009), and 'Sequence - Flow' (the flow-based propagator in Maher et al., 2008). Observe that the cumulative sums domain propagation, although not guaranteed to establish domain consistency, outperforms both domain consistent Sequence propagators. Also, MDD propagation with maximum width 2 can already substantially outperform domain propagation. We can further observe that larger maximum widths require more time for the MDDs to be processed, but in the end it does allow to solve more instances: maximum MDD width 128 permits to solve all 250 instances within the given time limit, whereas domain propagation can respectively solve 220 (Sequence - Flow), 230 (Sequence - HPRS), and 232 (Cumulative Sums) instances.

To illustrate the difference between domain and MDD propagation in more detail, Figure 5 presents scatter plots comparing domain propagation (cumulative sums) with MDD propagation (maximum width 32). This comparison is particularly meaningful because both propagation methods rely on the cumulative sums representation. For each instance, Figure 5.a depicts the number of backtracks while Figure 5.b depicts the solving time of both methods. The instances that were not solved within the time limit are collected under 'TO' (time out) for that method. Figure 5.a demonstrates that MDD propagation can lead to dramatic search tree reductions, by several orders of magnitude. Naturally, the MDD

2. All instances are available at `http://www.andrew.cmu.edu/user/vanhoeve/mdd/`.

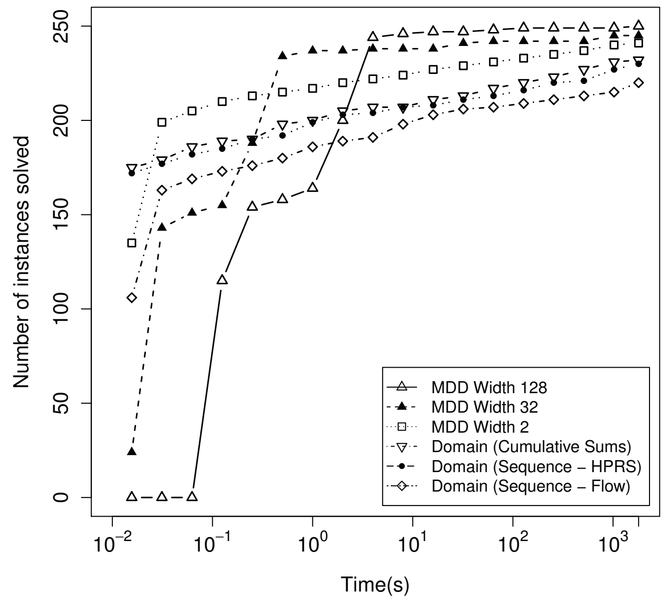
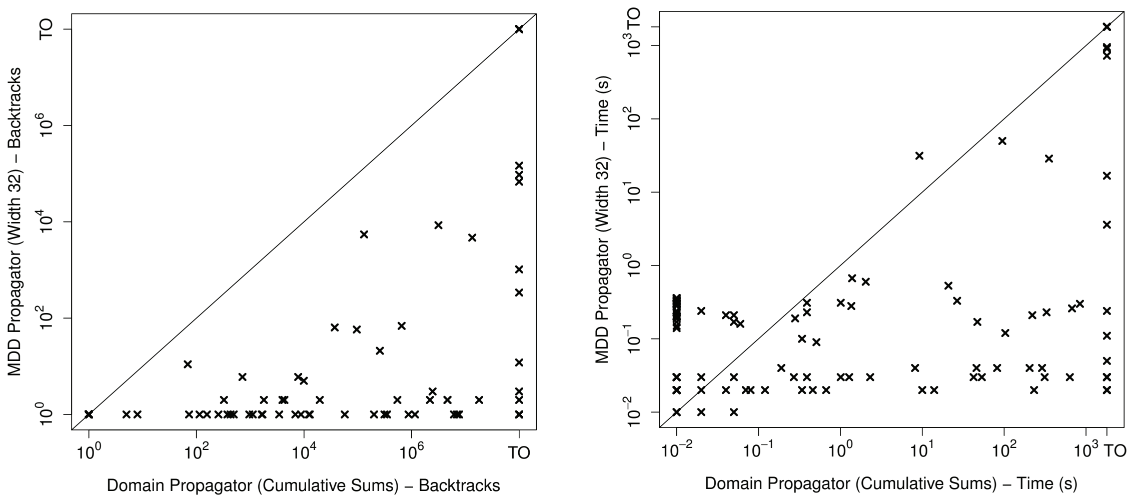


Figure 4: Performance comparison of domain and MDD propagators for the Sequence constraint. Each data point reflects the total number of instances that are solved by a particular method within the corresponding time limit.

propagation comes with a computational cost, but Figure 5.b shows that for almost all instances (especially the harder ones), the search tree reductions correspond to faster solving times, again often several orders of magnitude.

We next evaluate the impact of increasing maximum widths of the MDD propagator. In Figure 6, we present for each method the 'survival function' with respect to the number of backtracks (a.) and solving time (b.). Formally, when applied to combinatorial backtrack search algorithms, the survival function represents the probability of a run taking more than x backtracks (Gomes, Fernández, Selman, & Bessière, 2005). In our case, we approximate this function by taking the proportion of instances that need at least x backtracks (Figure 6.a), respectively seconds (Figure 6.b). Observe that these are log-log plots. With respect to the search tree size, Figure 6.a clearly shows the strengthening of the MDD propagation when the maximum width is increased. In particular, the domain propagation reflects the linear behavior over several orders of magnitude that is typical for heavy-tailed runtime distributions. Naturally, similar behavior is present for the MDD propagation, but in a much weaker form for increasing maximum MDD widths. The associated solving times are presented in Figure 6.b. It reflects similar behavior, but also takes into account the initial computational overhead of MDD propagation.



a. Number of backtracks

b. Solving time

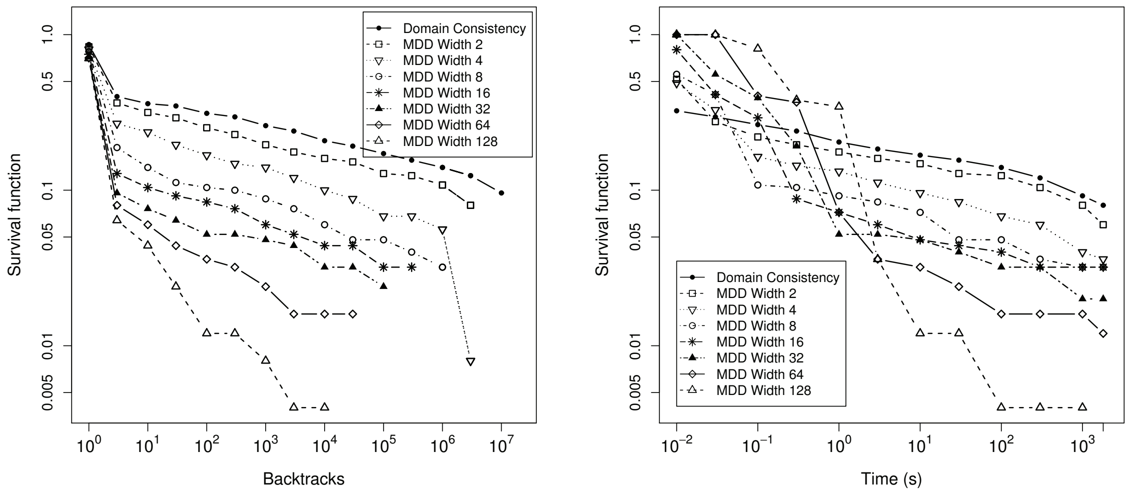
Figure 5: Comparing domain and MDD propagation for Sequence constraints. Each data point reflects the number of backtracks (a.) resp. solving time in seconds (b.) for a specific instance, when solved with the best domain propagator (cumulative sums encoding) and the MDD propagator with maximum width 32. Instances for which either method needed 0 backtracks (a.) or less than 0.01 seconds (b.) are excluded. Here, TO stands for 'timeout' and represents that the specific instance could not be solved within 1,800s (Fig. b.). In Figure a., these instances are labeled separately by TO (at tick-mark 10^8); note that the reported number of backtracks after 1,800 seconds may be much less than 10^8 for these instances. All reported instances with fewer than 10^8 backtracks were solved within the time limit.

6.2 Nurse Rostering Instances

We next consider a more structured problem class inspired by nurse rostering problems. The problem is to design a work schedule for a nurse over a given horizon of n days. On each day, a nurse can either work a day shift (D), evening shift (E), night shift (N), or have a day off (O). We introduce a variable x_i for each day $i = 1, \ldots, n$, with domain $D(x_i) = \{O, D, E, N\}$ representing the shift. We impose the eight Sequence constraints modeling the requirements listed in Table 1.

By the combinatorial nature of this problem, the size of the CP search tree turns out to be largely independent on the length of the time horizon, when a lexicographic search (by increasing day i) is applied. We however do consider instances with various time horizons ($n = 40$, 60, 80, 100), to address potential scaling issues.

The results are presented in Table 2. The columns for 'Domain Sequence' show the total number of backtracks (BT) and solving time in seconds (CPU) for the domain consistent Sequence propagator. Similarly, the columns for 'Domain Cumul. Sums' show this infor-



a. Survival function with respect to backtracks   b. Survival function with respect to solving time

Figure 6: Evaluating the impact of increased width for MDD propagation via survival function plots with respect to search backtracks (a.) and solving time (b.). Both plots are in log-log scale. Each data point reflects the percentage of instances that require at least that many backtracks (a.) resp. seconds (b.) to be solved by a particular method.

Requirement	Sequence(X, q, l, u, S)
At least 20 work shifts every 28 days:	Sequence$(X, 28, 20, 28, \{D, E, N\})$
At least 4 off-days every 14 days:	Sequence$(X, 14, 4, 14, \{O\})$
Between 1 and 4 night shifts every 14 days:	Sequence$(X, 14, 1, 4, \{N\})$
Between 4 and 8 evening shifts every 14 days:	Sequence$(X, 14, 4, 8, \{E\})$
Nights shifts cannot appear on consecutive days:	Sequence$(X, 2, 0, 1, \{N\})$
Between 2 and 4 evening/night shifts every 7 days:	Sequence$(X, 7, 2, 4, \{E, N\})$
At most 6 work shifts every 7 days:	Sequence$(X, 7, 0, 6, \{D, E, N\})$

Table 1: Nurse rostering problem specification. Variable set X represents the shifts to be assigned over a sequence of days. The possible shifts are day (D), evening (E), night (N), and day off (O).

mation for the cumulative sums domain propagation. The subsequent columns show these numbers for the MDD propagator, for MDDs of maximum width 1, 2, 4, and 8. Note that propagating an MDD of width 1 corresponds to domain propagation, and indeed the associated number of backtracks is equivalent to the domain propagator of the cumulative sums. As a first observation, a maximum width of 2 already reduces the number of backtracks by a factor 8.3. For maximum width of 8 the MDD propagation even allows to solve the

	Domain Sequence		Domain Cumul. Sums		MDD Width 1		MDD Width 2		MDD Width 4		MDD Width 8	
n	BT	CPU	BT	CPU	BT	CPU	BT	CPU	BT	CPU	BT	CPU
40	438,059	43.83	438,059	32.26	438,059	54.27	52,443	12.92	439	0.44	0	0.02
60	438,059	78.26	438,059	53.40	438,059	80.36	52,443	18.36	439	0.68	0	0.04
80	438,059	124.81	438,059	71.33	438,059	106.81	52,443	28.58	439	0.94	0	0.06
100	438,059	157.75	438,059	96.27	438,059	135.37	52,443	37.76	439	1.22	0	0.10

Table 2: Comparing domain propagation and the MDD propagation for Sequence on nurse rostering instances. Here, n stands for the number of variables, BT for the number of backtracks, and CPU for solving time in seconds.

problem without search. The computation times are correspondingly reduced, e.g., from 157s (resp. 96s) for the domain propagators to 0.10s for the MDD propagator (width 8) for the instance with $n = 100$. Lastly, we can observe that in this case MDD propagation does not suffer from scaling issues when compared to domain propagation.

As a final remark, we also attempted to solve these nurse rostering instances using the Sequence domain propagator of CP Optimizer (`IloSequence`). It was able to solve the instance with $n = 40$ in 1,150 seconds, but none of the others instances were solved within the time limit of 1,800 seconds.

6.3 Comparing MDD Filtering for Sequence and Among

In our last experiment, we compare our Sequence MDD propagator to the MDD propagator for Among constraints by Hoda et al. (2010). Our main goal is to determine whether a large MDD is by itself sufficient to solve these problem (irrespective of propagating Among or a cumulative sums decomposition), or whether the additional information obtained by our Sequence propagator makes the difference.

We apply both methods, MDD propagation for Sequence and MDD propagation for Among, to the data set of Section 6.1 containing 250 instances. The time limit is again 1,800 seconds, and we run the propagators with maximum MDD widths 2, 8, 32, and 128.

We first compare the performance of the MDD propagators for Among and Sequence in Figure 7. The figure depicts the number of instances that can be solved within a given time limit for the various methods. The plot indicates that the Among propagators are much weaker than the Sequence propagator, and moreover that larger maximum widths alone do not suffice: using the Sequence propagator with maximum width 2 outperforms the Among propagators for all maximum widths up to 128.

The scatter plot in Figure 8 compares the MDD propagators for Among and Sequence in more detail, for widths 2, 8, 32, and 128 (instances that take 0 backtracks, resp. less than 0.01 seconds, for either method are discarded from Figure 8.a, resp. 8.b). For smaller widths, there are several instances that the Among propagator can solve faster, but the relative strength of the Sequence propagator increases with larger widths. For width 128, the Sequence propagator can achieve orders of magnitude smaller search trees and

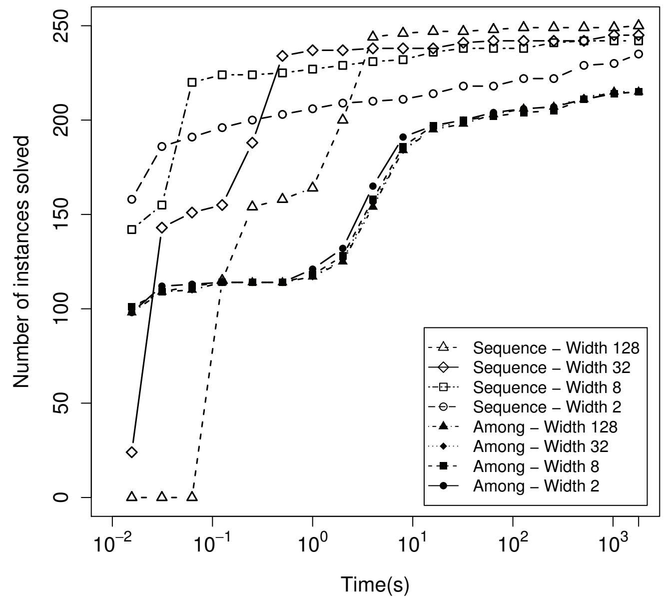
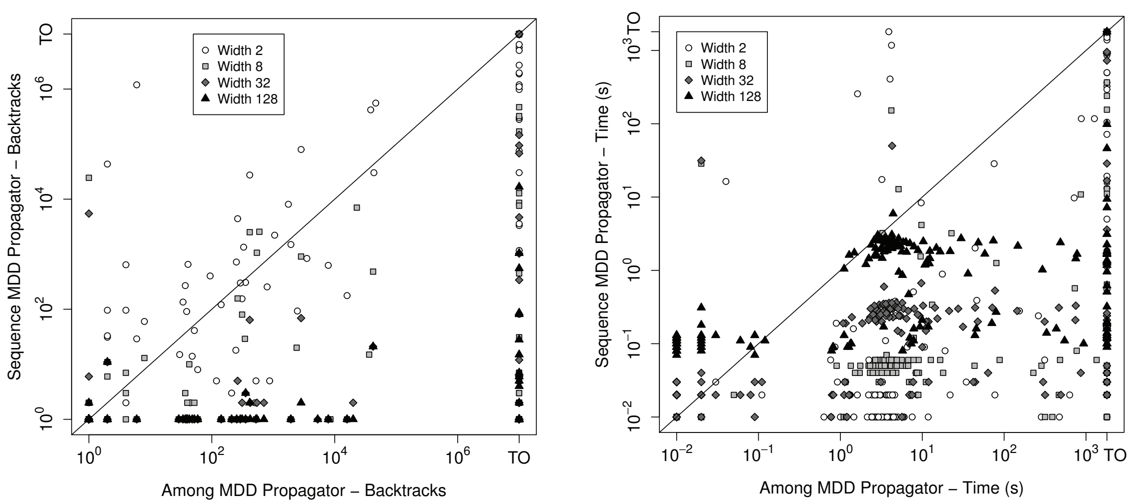


Figure 7: Performance comparison of MDD propagation for Sequence and Among for various maximum widths. Each data point reflects the total number of instances that are solved by a particular method within the corresponding time limit.



a. Number of backtracks

b. Solving time

Figure 8: Evaluating MDD propagation for Sequence and Among for various maximum widths via scatter plots with respect to search backtracks (a.) and solving time (b.). Both plots are in log-log scale and follow the same format as Figure 5.

solving time than the Among propagators, which again demonstrates the advantage of MDD propagation for Sequence when compared to the Among decomposition.

7. Conclusion

Constraint propagation with limited-width MDDs has recently been shown to be a powerful alternative to the conventional propagation of variable domains in constraint programming. In this work, we have studied MDD propagation for the Sequence constraint, which appears in, e.g., rostering and scheduling applications. We have first proved that establishing MDD consistency for Sequence is NP-hard. However, we have also shown that this task is fixed parameter tractable with respect to the length of the sub-sequences defined by the constraint, provided that the MDD follows the variable ordering specified by the constraint. We then proposed a practical MDD propagation algorithm for Sequence that is also polynomial in the length of the sub-sequences, which is based on a cumulative decomposition. We provided extensive experimental results comparing our MDD propagator for Sequence to domain propagators for Sequence as well as an existing MDD propagator for Among. Our computational experiments have shown that our MDD propagator for Sequence can outperform domain propagators by orders by magnitude in terms of search tree size and solving time. Similar results were obtained when compared to the existing MDD propagator for Among, which demonstrates that in practice a large MDD alone is not sufficient to solve these problems; specific MDD propagators for global constraints such as Sequence can lead to orders of magnitude speedups.

Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant No. CMMI-1130012, and a Google Research Award. We also thank the reviewers whose comments helped improve the paper.

References

Andersen, H. R., Hadzic, T., Hooker, J. N., & Tiedemann, P. (2007). A Constraint Store Based on Multivalued Decision Diagrams. In *Proceedings of CP*, Vol. 4741 of *LNCS*, pp. 118–132. Springer.

Apt, K. R. (2003). *Principles of Constraint Programming*. Cambridge University Press.

Beldiceanu, N., & Contejean, E. (1994). Introducing global constraints in CHIP. *Journal of Mathematical and Computer Modelling*, *20*(12), 97–123.

Brand, S., Narodytska, N., Quimper, C., Stuckey, P., & Walsh, T. (2007). Encodings of the Sequence Constraint. In *Proceedings of CP*, Vol. 4741 of *LNCS*, pp. 210–224. Springer.

Cheng, K., & Yap, R. (2008). Maintaining Generalized Arc Consistency on Ad Hoc r-Ary Constraints. In *Proceedings of CP*, Vol. 5202 of *LNCS*, pp. 509–523. Springer.

Cire, A. A., & van Hoeve, W.-J. (2012). MDD Propagation for Disjunctive Scheduling. In *Proceedings of ICAPS*, pp. 11–19. AAAI Press.

Cire, A. A., & van Hoeve, W.-J. (2013). Multivalued Decision Diagrams for Sequencing Problems. *Operations Research*, *61*(6), 1411–1428.

Dechter, R. (2003). *Constraint Processing*. Morgan Kaufmann.

Downing, N., Feydy, T., & Stuckey, P. (2012). Explaining Flow-Based Propagation. In *Proceedings of CPAIOR*, Vol. 7298 of *LNCS*, pp. 146–162. Springer.

Garey, M., & Johnson, D. (1979). *Computers and Intractability - A Guide to the Theory of NP-Completeness*. Freeman.

Gomes, C. P., Fernández, C., Selman, B., & Bessière, C. (2005). Statistical Regimes Across Constrainedness Regions. *Constraints*, *10*(4), 317–337.

Hadzic, T., Hooker, J. N., O'Sullivan, B., & Tiedemann, P. (2008a). Approximate Compilation of Constraints into Multivalued Decision Diagrams. In *Proceedings of CP*, Vol. 5202 of *LNCS*, pp. 448–462. Springer.

Hadzic, T., Hooker, J. N., & Tiedemann, P. (2008b). Propagating Separable Equalities in an MDD Store. In *Proceedings of CPAIOR*, Vol. 5015 of *LNCS*, pp. 318–322. Springer.

Hadzic, T., O'Mahony, E., O'Sullivan, B., & Sellmann, M. (2009). Enhanced Inference for the Market Split Problem. In *Proceedings of ICTAI*, pp. 716–723. IEEE.

Hawkins, P., Lagoon, V., & Stuckey, P. (2005). Solving Set Constraint Satisfaction Problems Using ROBDDs. *JAIR*, *24*(1), 109–156.

Hoda, S., van Hoeve, W.-J., & Hooker, J. N. (2010). A Systematic Approach to MDD-Based Constraint Programming. In *Proceedings of CP*, Vol. 6308 of *LNCS*, pp. 266–280. Springer.

Hosaka, K., Takenaga, Y., Kaneda, T., & Yajima, S. (1997). Size of ordered binary decision diagrams representing threshold functions. *Theoretical Computer Science*, *180*, 47–60.

Knuth, D. E. (2009). *The Art of Computer Programming, Volume 4, Fascicle 1: Bitwise Tricks & Techniques; Binary Decision Diagrams*. Addison-Wesley Professional.

Maher, M., Narodytska, N., Quimper, C.-G., & Walsh, T. (2008). Flow-Based Propagators for the SEQUENCE and Related Global Constraints. In *Proceedings of CP*, Vol. 5202 of *LNCS*, pp. 159–174. Springer.

Narodytska, N. (2011). *Reformulation of Global Constraints*. Ph.D. thesis, University of New South Wales.

Régin, J.-C. (1994). A Filtering Algorithm for Constraints of Difference in CSPs. In *Proceedings of AAAI*, Vol. 1, pp. 362–367. AAAI Press.

Régin, J.-C. (2011). Global Constraints: A Survey. In Van Hentenryck, P., & Milano, M. (Eds.), *Hybrid Optimization*, pp. 63–134. Springer.

Régin, J.-C., & Puget, J.-F. (1997). A Filtering Algorithm for Global Sequencing Constraints. In *Proceedings of CP*, Vol. 1330 of *LNCS*, pp. 32–46. Springer.

Rossi, F., van Beek, P., & Walsh, T. (Eds.). (2006). *Handbook of Constraint Programming*. Elsevier.

van Hoeve, W.-J., & Katriel, I. (2006). Global Constraints. In Rossi, F. van Beek, P., & Walsh, T. (Eds.), *Handbook of Constraint Programming*, chap. 6. Elsevier.

van Hoeve, W.-J., Pesant, G., Rousseau, L.-M., & Sabharwal, A. (2006). Revisiting the Sequence Constraint. In *Proceedings of CP*, Vol. 4204 of *LNCS*, pp. 620–634. Springer.

van Hoeve, W.-J., Pesant, G., Rousseau, L.-M., & Sabharwal, A. (2009). New Filtering Algorithms for Combinations of Among Constraints. *Constraints*, *14*, 273–292.

Wegener, I. (2000). *Branching Programs and Binary Decision Diagrams: Theory and Applications*. SIAM monographs on discrete mathematics and applications. Society for Industrial and Applied Mathematics.