

# MDD Propagation for Sequence Constraints

Willem-Jan van Hoeve

Tepper School of Business, Carnegie Mellon University  
5000 Forbes Avenue, Pittsburgh, PA 15213, U.S.A.  
vanhoeve@andrew.cmu.edu

**Abstract.** We study propagation for the *sequence* constraint in the context of constraint programming based on limited-width MDDs. Our first main contribution is proving that establishing MDD-consistency for *sequence* is NP-hard. Second, 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. Finally, we experimentally evaluate the performance of our proposed filtering algorithm. We show that large savings can be obtained in terms of search tree size and computation time with respect to the current best MDD approach that applies the decomposition of *sequence* into *among* constraints. In addition, we show the potential of our MDD propagator with respect to a state-of-the-art domain propagator for *sequence*, both in terms of search tree size and computation time.

## 1 Introduction

Recently, Andersen et al. [1] 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. It has been shown that MDDs of limited width can provide a much stronger relaxation of the solution space than the traditional Cartesian product of the variable domains, while in addition MDDs allow to represent and communicate more refined information between constraints. By propagating MDDs rather than variable domains between constraints, huge reductions in search tree size and computation time can be realized [1, 8, 7, 12, 3].

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 [10, 5, 9], 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 using 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. Such MDD propagation can be implemented in parallel to the existing domain representation in CP systems, thus complementing and potentially strengthening the domain propagation process.

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 edges from the MDD [8, 12]. An MDD refinement algorithm on the other hand, aims at splitting nodes in the MDD to more accurately reflect the solution space [7]. 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. However, increasing the size of the MDD immediately impacts the computation time, and one typically aims at balancing 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 in [1], similar to domain consistency in finite-domain constraint programming: Given an MDD, a constraint is MDD consistent if all edges 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. [1] show that establishing MDD consistency on the *alldifferent* constraint is NP-hard, while establishing traditional domain consistency can be done in polynomial time [16].

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 [2]. The *sequence* constraint finds applications in, e.g., car sequencing and employee scheduling problems [17, 14]. It is known that classical domain consistency can be established for *sequence* in polynomial time [13, 14, 4, 15]. Furthermore, in [12] an MDD filtering algorithm for *among* constraints establishing MDD consistency in polynomial time is presented. 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 show that establishing MDD consistency on the *sequence* constraint is NP-hard, using a reduction from 3-SAT. 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 specified variable ordering, it combines sub-constraints on contiguous variables, and existing approaches can handle this constraint fully by using bounds reasoning only. Interestingly, we show that these characteristics can still be utilized in a partial MDD filtering algorithm for *sequence* (which can also be applied to the *gen-sequence* constraint). Our algorithm 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. We experimentally evaluate the strength

of our partial MDD filtering algorithm for *sequence* on MDDs of various widths. We also compare the algorithm against the currently best known MDD approach that uses the natural decomposition of *sequence* into *among* constraints [12]. Our experiments show that similar to the classical domain propagation, also for MDD filtering the dedicated algorithm for *sequence* can indeed outperform the *among* decomposition in terms of search tree size and solving time, sometimes by several orders of magnitude. Finally, we experimentally compare our MDD propagator to a state-of-the-art domain propagator for *sequence*, and show that it can outperform the domain propagator both in terms of search tree size and computation time.

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. Then, in Section 3, we present the proof that establishing MDD consistency on *sequence* is NP-hard. In Section 4, the partial MDD filtering algorithm is presented. In Section 5, the experimental results are presented. We present final conclusions in Section 6.

## 2 Definitions

We first recall some basic definitions of MDD-based constraint programming, following [1, 12]. 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*  $L_1, \dots, L_{n+1}$ , where the layers  $L_1, \dots, L_n$  correspond respectively to variables  $x_1, \dots, x_n$ .  $L_1$  contains a single *top* node  $\mathbf{T}$ , and  $L_{n+1}$  contains two *bottom* nodes  $\mathbf{0}$  and  $\mathbf{1}$ , representing ‘false’ and ‘true’, respectively. The *width* of the MDD 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 edges 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 edge connects two adjacent layers. Let  $L(s)$  denote the layer of the node  $s$ . Each edge out of layer  $i$  is labeled with an element of the domain  $D(x_i)$  of  $x_i$ . The set  $E(s, t)$  of edges from node  $s$  to node  $t$  may contain multiple edges, and we denote each with its label. Let  $E^{in}(s)$  denote the set of edges coming into  $s$ .

An edge with label  $v$  leaving a node in layer  $i$  represents an assignment  $x_i = v$ . Each path in the MDD from  $\mathbf{T}$  to  $\mathbf{0}$  or  $\mathbf{1}$  can be denoted by the edge labels  $v_1, \dots, v_n$  on the path and is identified with the assignment  $(x_1, \dots, x_n) = (v_1, \dots, v_n)$ . For our purposes, it is convenient to generate only the portion of an MDD that contains paths from  $\mathbf{T}$  to  $\mathbf{1}$ , i.e., all paths leading to satisfiability.

A path  $v_1, \dots, v_n$  is *feasible* for a given constraint  $C$  if setting  $(x_1, \dots, x_n) = (v_1, \dots, v_n)$  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 edge of the MDD lies on some feasible path. Thus MDD consistency is achieved when all redundant edges (i.e., edges on no feasible path) have been removed. 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  $E(s_i, s_{i+1}) = D(x_i)$  for  $i = 1, \dots, n$ .

Finally, we formally recall the definitions of *among* [2], *sequence* [2], and *gen-sequence* [14] 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 *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*

$$\text{sequence}(X, q, l, u, S) = \bigwedge_{i=1}^{n-q+1} \text{among}(s_i, l, u, S),$$

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

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,  $\mathbf{s}, \mathbf{l}, \mathbf{u}$  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  $i = 1, 2, \dots, k$ , and  $S \subset \cup_{x \in X} D(x)$  a subset of domain values. Then*

$$\text{gen-sequence}(X, \mathbf{s}, \mathbf{l}, \mathbf{u}, S) = \bigwedge_{i=1}^k \text{among}(s_i, l_i, u_i, S).$$

### 3 MDD-Consistency for Sequence is NP-Hard

As stated before, the only known NP-hardness result for a global constraint in the context of MDD-based constraint programming is that of Andersen et al. [1] for the *alldifferent* constraint. A particular difficulty of 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.

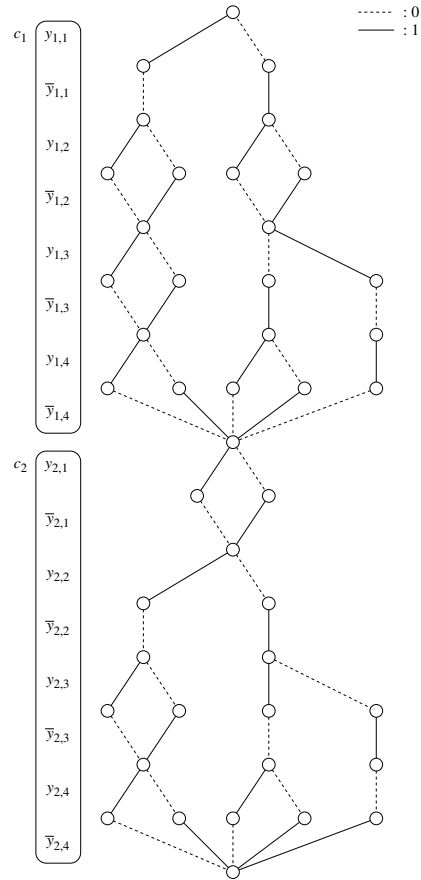
**Theorem 1.** *Establishing MDD consistency for sequence on an arbitrary MDD is NP-hard even if the MDD follows the variable ordering of the sequence constraint.*

*Proof.* The proof is by reduction from 3-SAT, a classical NP-complete problem [6]. 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  $\bar{y}_{i,j}$  representing the literals  $x_j$  and  $\bar{x}_j$  per clause  $c_i$ , for  $i = 1, \dots, m$  and  $j = 1, \dots, n$  ( $x_j$  and  $\bar{x}_j$  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  $y_{i,j}, \bar{y}_{i,j}$  for clause  $c_i$  and variable  $x_j$ . That is, we have  $Y = y_{1,1}, \bar{y}_{1,1}, y_{1,2}, \bar{y}_{1,2}, \dots, y_{1,n}, \bar{y}_{1,n}, \dots, y_{m,1}, \bar{y}_{m,1}, \dots, y_{m,n}, \bar{y}_{m,n}$ . 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  $y_{i,1}, \dots, \bar{y}_{i,n}$ . 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  $(y_{i,j}, \bar{y}_{i,j})$ , that defines either  $(0, 1)$  or  $(1, 0)$  to this pair. If a variable does not appear in a clause, we represent it using such a diamonds 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 define its first layer  $L(y_{i,1})$  by merging all existing nodes in the layer to a single node  $t$  collecting all incoming edges, and we set  $\text{tag}(t) = \text{‘unsat’}$ . For each node  $u$  in layer  $L(y_{i,j})$  (for  $j = 1, \dots, n$ ), we do the following. If variable  $y_{i,j}$  does not appear in  $c_i$ , or if  $\text{tag}(u)$  is ‘sat’, we create two nodes  $v, v'$  in  $L(\bar{y}_{i,j})$ , one single node  $w$  in  $L(y_{i,j+1})$ , and edges  $(u, v)$  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  $\text{tag}(w) = \text{tag}(u)$ . Otherwise (i.e.,  $\text{tag}(u)$  is ‘unsat’ and  $y_{i,j}$  appears in  $c_i$ ), we create two nodes  $v, v'$  in  $L(\bar{y}_{i,j})$ , two nodes  $w, w'$  in  $L(y_{i,j+1})$ , and edges  $(u, v)$  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  $\text{tag}(w) = \text{‘sat’}$  and  $\text{tag}(w') = \text{‘unsat’}$ . Otherwise ( $c_i$  contains  $\bar{y}_{i,j}$ ), we set  $\text{tag}(w) = \text{‘unsat’}$  and  $\text{tag}(w') = \text{‘sat’}$ .



**Fig. 1.** The MDD corresponding to Example 1.

This procedure will be initialized by a single root node  $\mathbf{T}$  representing  $L(y_{11})$ , and finalized by merging all nodes in the last layer to the single node  $\mathbf{1}$ . By construction, we ensure that only one of  $y_{ij}$  and  $\bar{y}_{ij}$  can be set to 1. Furthermore, the variable assignment corresponding to each path between layers  $L(y_{i,1})$  and  $L(y_{i+1,1})$  will satisfy clause  $c_i$ , and exactly  $n$  literals are chosen accordingly on each such path.

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

Next we need to concatenate the information from one clause to the next, or better, we need to ensure that each variable  $x_j$  will correspond to the same literal  $y_{i,j}$  or  $\bar{y}_{i,j}$  in each clause  $c_i$ . To this end, we impose the constraint

$$\text{sequence}(Y, q = 2n, l = n, u = n, S = \{1\}) \quad (1)$$

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  $\bar{y}_{i,j}$  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, 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 hold for all sub-sequences, all variables  $x_j$  must take the same value in all clauses.  $\square$

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

## 4 Partial MDD Filtering for Sequence

Even though establishing complete MDD consistency for *sequence* is NP-hard, we can still take advantage of the structure captured by *sequence* to improve the MDD filtering. 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 in [12]. However, it is well-known that for classical constraint propagation based on variable domains, the *among* decomposition can be greatly improved by a dedicated domain filtering algorithm for *sequence* [13, 14, 4, 15]. Therefore, our goal in this section is to provide MDD filtering for *sequence* that is stronger than the *among* decomposition. In what follows, we assume that the MDD at hand respects the ordering of the variables in the *sequence* constraint. The proposed filtering algorithm can also be applied to other orderings, but this will most likely weaken its performance.

### 4.1 Cumulative Sums Encoding

Our proposed algorithm extends the original domain filtering algorithm for *sequence* in [13] to MDDs, following the ‘cumulative sums’ encoding as proposed in [4]. This representation takes the following form. For a sequence of variables  $X = x_1, x_2, \dots, x_n$ , and a constraint *sequence*( $X, q, l, u, S$ ), we first introduce variables  $y_0, y_1, \dots, y_n$ , with respective initial domains  $D(y_i) = [0, i]$  for  $i = 1, \dots, n$ . These variables represent the cumulative sums of  $X$ , i.e.,  $y_i$  represents  $\sum_{j=1}^i x_j$  for  $i = 1, \dots, n$ . We now rewrite the *sequence* constraint as the following system of constraints:

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

$$y_{i+q} - y_i \geq l \quad \forall i \in \{0, \dots, n - q\}, \quad (3)$$

$$y_{i+q} - y_i \leq u \quad \forall i \in \{0, \dots, n - q\}, \quad (4)$$

where  $\delta_S : X \rightarrow \{0, 1\}$  is the indicator function for the set  $S$ , i.e.,  $\delta_S(x) = 1$  iff  $x \in S$  and  $\delta_S(x) = 0$  iff  $x \notin S$ . In [4], it is shown 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 information captured by the variables  $y_0, \dots, y_n$  can be naturally represented by the *nodes* of the MDDs. 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 **T-1** paths containing  $v$ . Let us denote this information for each node  $v$  explicitly as the interval  $[\text{lb}(v), \text{ub}(v)]$ , and we will refer to it as the ‘node domain’ of  $v$ . Following the approach of [12], we can compute this information in linear time by one top-down pass, by using equation (2), as follows:

$$\begin{aligned} \text{lb}(v) &= \min_{e \in E(u,v)} \{\text{lb}(u) + \delta_S(e)\}, \\ \text{ub}(v) &= \max_{e \in E(u,v)} \{\text{ub}(u) + \delta_S(e)\}, \end{aligned} \tag{5}$$

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

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  $\mathcal{A}_{\mathbf{T}} = [\mathbf{T}, \emptyset, \dots, \emptyset]$ , and apply the recursion

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

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  $O(qW^2)$  operations, where  $W$  is the maximum width of the MDD. 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  $\mathcal{D}_{\mathbf{1}} = [\mathbf{1}, \emptyset, \dots, \emptyset]$ .

Alternatively, we can approximate the information from the ancestor and descendant nodes by only maintaining a lower and upper bound on the node domains of  $\mathcal{A}_v$ , resp.,  $\mathcal{D}_v$ , resulting in a recursive step of  $O(qW)$  operations per node.

## 4.2 Processing the Constraints

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

Starting with the ternary constraints of type (2), we remove an edge  $e \in E(u, v)$  if  $\text{lb}(u) + \delta_S(e) > \text{ub}(v)$ . Updating  $[\text{lb}(u), \text{ub}(u)]$ , and  $[\text{lb}(v), \text{ub}(v)]$  is done similar to the rules (5) above:

$$\begin{aligned} \text{lb}(v) &= \max \{ \text{lb}(v), \min_{e \in E(u,v)} \{ \text{lb}(u) + \delta_S(e) \} \}, \\ \text{ub}(v) &= \min \{ \text{ub}(v), \min_{e \in E(u,v)} \{ \text{ub}(u) + \delta_S(e) \} \}, \end{aligned}$$

In fact, the resulting algorithm is a special case of the MDD consistency equality propagator of [7], 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  $L_{i+1}$  ( $i = 0, \dots, n$ ). 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:

$$\begin{aligned} y_i &\geq l + y_{i-q}, \\ y_i &\leq u + y_{i-q}, \\ y_i &\leq y_{i+q} - l, \\ y_i &\geq y_{i+q} - u. \end{aligned} \tag{6}$$

Observe that we can apply these constraints to filter *only* the node domain  $[\text{lb}(v), \text{ub}(v)]$  corresponding to  $y_i$ . Namely, the node domains corresponding to the other variables  $y_{i-q}$  and  $y_{i+q}$  may be connected to nodes in layer  $L_{i+1}$  other than  $v$ . We update  $\text{lb}(v)$  and  $\text{ub}(v)$  according to equations (6):

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

The resulting algorithm is a specific instance of the generic MDD consistent binary constraint propagator presented in [12], and again we inherit the MDD consistency for these constraints.

We can choose to apply the filtering algorithms repeatedly until a fixed point is reached, similar to classical domain propagation. Moreover, we can strengthen the propagation by applying an extension of singleton bounds consistency to MDDs, which we will refer to as ‘singleton bounds MDD consistency’. That is, for each adjacent pair of nodes  $(u, v)$ , respectively for each node  $v$ , we determine the feasibility of  $\min E(u, v)$ ,  $\max E(u, v)$ , respectively  $\text{lb}(u)$ ,  $\text{ub}(v)$ , by restricting the MDD to that particular edge, respectively node domain value, and applying the fixed point constraint propagation on the resulting MDD. If the propagation leads to an empty domain, we can tighten the corresponding bound by one unit, and continue. Note that during the fixed point computation, we must regularly update  $A_v$  and  $D_v$  for each node  $v$ , as the filtering algorithms typically remove edges from the MDD. We have the following result.

**Lemma 1.** *Singleton bounds MDD consistency establishes domain consistency for sequence for MDDs of width 1, assuming that the MDD follows the variable ordering of the sequence constraint.*

*Proof.* When applied to an MDD of width 1 (following the variable order in the sequence constraint), the singleton bounds MDD consistency algorithm performs singleton bounds consistency on the cumulative sums decomposition, which was shown in Brand et al. [4] to establish domain consistency on sequence.  $\square$

Note that for MDDs of general width, however, we cannot expect our polynomial-time algorithm to establish MDD consistency, given Theorem 1.

In practice, establishing a fixed point for the MDD propagation of the cumulative sums encoding will typically be too computationally expensive. Therefore, in our experiments we apply only one round of propagation, that is, each constraint is treated once in a top-down pass (considering the ancestors  $\mathcal{A}_v$  for each node  $v$ ), and once in a bottom-up pass (considering its descendants  $\mathcal{D}_v$ ), through the MDD.

*Remark.* Observe that the proposed algorithm can be applied immediately to the more general *gen-sequence* constraints in which each *among* constraint has its individual  $l, u$  and  $q$ , as the cumulative sums encoding can be adjusted in a straightforward manner to represent these different values.

## 5 Computational Results

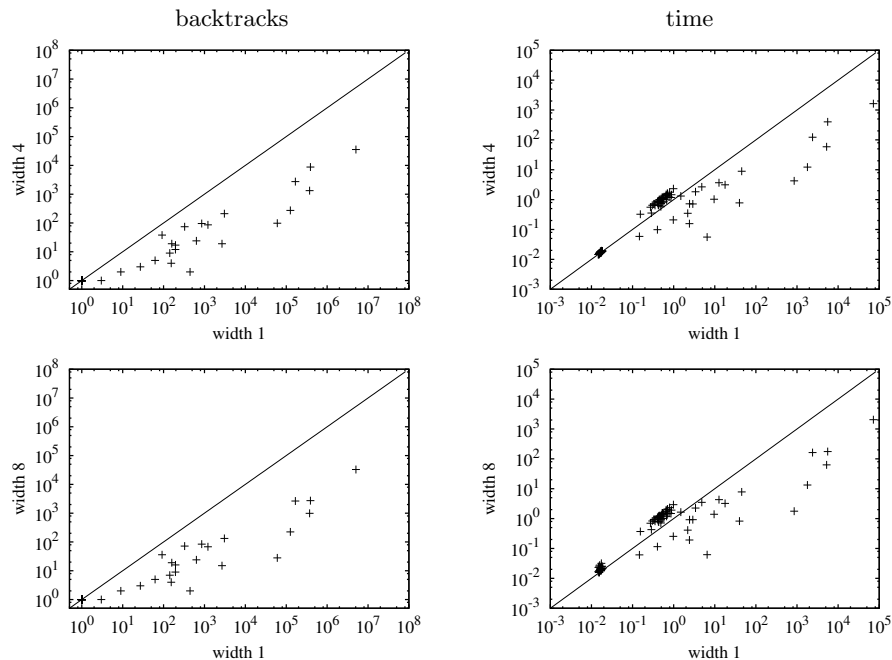
The purpose of our computational results is to evaluate empirically the filtering strength of the *sequence* propagator. This is done through three experiments. First, we evaluate the filtering strength of the *sequence* propagator for various widths of the MDD, to better understand its behavior. Second, we compare the filtering strength of the *sequence* propagator to the filtering strength of the individual *among* propagators, being the best MDD approach for *sequence* so far [12]. Third, we compare the MDD propagator to a state-of-the-art domain propagator for *sequence*.

We have implemented our MDD propagator for *sequence* inside the MDD-based CP solver presented in [11]. The model that uses the *among* decomposition has also been implemented in this solver, as described in [12]. The domain propagator for *sequence* has been implemented in IBM-ILOG CP Optimizer 1.6. All three approaches apply the same search strategy, being the smallest domain size as variable selection heuristic (breaking ties lexicographically), and a fixed lexicographic value ordering. Observe that the different levels of filtering established by the various methods may impact the dynamic variable selection heuristic. All experiments are performed using a 2.33GHz Intel Xeon machine with 8GB memory.

We note that for single *sequence* or *gen-sequence* constraints, our MDD propagator for *sequence* can provably obtain domain consistency already for MDDs of width one, provided that the ‘singleton bound consistency’ algorithm is applied. We have confirmed this behavior in our experiments; even when a single round of filtering is applied, single *sequence* or *gen-sequence* constraints are trivially solved using our algorithm. Therefore, in our experiments we focus on problems consisting of multiple *sequence* constraints.

### 5.1 MDD Sequence Filtering for Various Widths

We first consider systems of multiple independent *sequence* constraints similar to [4]. We randomly generate instances consisting of two *sequence* constraints,



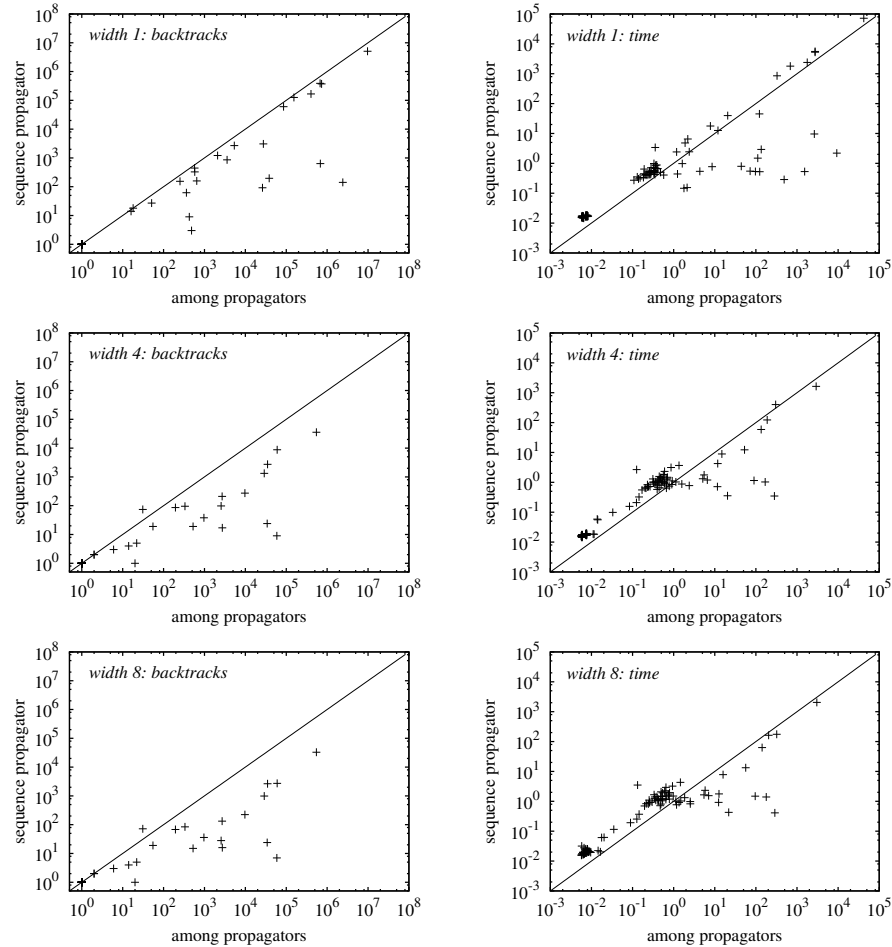
**Fig. 2.** Scatter plots comparing the behavior of the *sequence* propagator on MDDs of width 1 versus width 4 and 8 (from top to bottom) in terms of backtracks (left) and computation time in seconds (right) on multiple *sequence* problems.

defined on  $n = 50$  variables. For each *sequence* constraint, we set  $q = 14$ , while  $u - l = 1$  by uniform-randomly selecting a lower bound  $l$  from  $[0, n - 1]$  and setting  $u = l + 1$ . We generated 100 such instances, and solved each instance using our *sequence* propagator, for various maximum widths.

The results, presented in Figure 2, compare the impact of increasing the width on the performance of the *sequence* propagator in terms of search tree size (backtracks) and solution time (in seconds). In these scatter plots, we compare an MDD of maximum width 1 to widths 4 and 8. Clearly, increasing the width improves the performance of the propagator (we note that increasing the width beyond 8 did not pay off in terms of computation time for these instances). These plots demonstrate the power of MDD-based constraint propagation, being able to achieve several orders of magnitude speedups when compared to the traditional domain propagation (represented by the MDD of width 1).

## 5.2 Comparing MDD Filtering for Sequence and Among

The second set of results, presented in Figure 3, compares the *among* propagators with the *sequence* propagator, again in terms of search tree size and solution time, for maximum MDD widths of 1, 4, and 8. These experiments are



**Fig. 3.** Scatter plots comparing the *among* and *sequence* propagators for widths 1, 4 and 8 (from top to bottom) in terms of backtracks (left) and computation time in seconds (right) on multiple *sequence* problems.

performed on the same set of instances as were used in the previous section. The plots indicate that the *sequence* propagator can indeed outperform the *among* propagators, both in terms of search tree size (backtracks) and solving time. The *sequence* propagator can be particularly effective on the harder instances, with improvements of several orders of magnitude for some instances.

In addition to the reported results, we remark that we have experimented with various other settings. Interestingly, for smaller values of  $q$  (say below 5), the performance of the *among* and *sequence* constraints was comparable. That is, the individual MDD propagators for *among* are already quite strong for instances

instance	domain propagator		MDD width 1		MDD width 4		MDD width 8		MDD width 16	
	BT	CPU	BT	CPU	BT	CPU	BT	CPU	BT	CPU
$n$										
40	121,767	4.63	34,108	251.06	75	1.67	28	0.99	28	1.00
50	121,777	5.67	34,108	487.25	75	3.06	29	1.86	29	1.90
60	121,782	6.51	34,108	796.21	75	5.00	30	3.09	30	3.12
70	121,787	6.99	34,108	1,110.88	75	6.96	28	4.30	28	4.33
80	121,792	7.48	34,108	1,492.33	75	9.38	28	5.97	28	5.88

**Table 1.** Comparing the domain propagator and the MDD propagator 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.

with a small  $q$ . When  $q$  was increased, we noticed that the *sequence* propagator quickly started to pay off, as witnessed by the reported results.

### 5.3 Comparing MDD Filtering and Domain Filtering for Sequence

In our last set of experiments, we compare the MDD propagator with the domain propagator for *sequence*. There exist several domain consistency algorithms for *sequence* [13, 14, 4, 15], the theoretically most efficient one running in  $O(n^2)$  time [15]. In practice, the fastest algorithms are the algorithm of [15] and the cumulative-sums encoding of [4]. For our purposes, the most insightful comparison will be with the cumulative-sums encoding. Namely, that domain propagator is mimicked by our MDD propagator for MDDs of width 1, albeit in a weaker form (we don't run our propagator until a fixed point is reached, and also we do not perform singleton bounds consistency). This allows us to measure the amount of computational overhead induced by the MDD, as well as the quality of the MDD representation in terms of its maximum width.

For this experiment, we consider a more structured problem domain inspired by nurse rostering problems, similar to [14, 4]. 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 demand that the nurse works at least 22 day or evening shifts every 30 consecutive days, while in addition having between 1 and 4 days off every 7 consecutive days. This problem is modeled by introducing a variable  $x_i$  for each day  $i = 1, \dots, n$ , with domain  $D(x_i) = \{O, D, E, N\}$ . The constraints are modeled as *sequence*( $X, q = 30, l = 22, u = 30, S = \{D, E\}$ ) and *sequence*( $X, q = 7, l = 1, u = 4, S = \{O\}$ ), where  $X = x_1, x_2, \dots, x_n$ . We consider instances with  $n = 40, 50, 60, 70$ , and 80.

The results are presented in Table 1. The columns for 'domain propagator' show the total number of backtracks (BT) and solving time in seconds (CPU) for the domain propagator. The subsequent columns show these numbers for the MDD propagator, for MDDs of maximum width 1, 4, 8, and 16. There are two

important observations to be made. First, the total number of backtracks can be dramatically reduced already for MDDs of very small width (i.e., width 4 or 8), on average with about four orders of magnitude. Second, our current prototype implementation carries significant computational overhead; the savings in terms of computation time are respectable, with at most a factor 4.7 for these instances, but they illustrate that maintaining and manipulating the MDDs comes with a computational cost. Nevertheless, these results demonstrate the potential of MDD-based propagators with respect to a state-of-the-art domain store propagator for *sequence*. We expect that a more efficient implementation of our propagator will yield a more pronounced impact on computation time as well.

## 6 Conclusion

In this work, we have studied constraint propagation of *sequence* constraints for limited-width MDDs. As the first main contribution, we have shown that establishing MDD consistency for *sequence* is NP-hard. Our second contribution is a partial MDD filtering algorithm for *sequence* by introducing the new concept of node domain filtering in the context of MDD-based constraint programming. We have presented an experimental evaluation of our MDD propagator, demonstrating that our MDD propagator for *sequence* can outperform the current best MDD approach that applies the decomposition of *sequence* into *among* constraints, in some cases by several orders of magnitude. This implies that additional filtering based on global constraints can also be worthwhile in the context of MDD-based constraint propagation. Finally, we have experimentally shown the potential of our MDD-based propagator with respect to a state-of-the-art domain store propagator, in terms of search tree size as well as solving time.

## Bibliography

- [1] H.R. Andersen, T. Hadzic, J.N. Hooker, and P. Tiedemann. A Constraint Store Based on Multivalued Decision Diagrams. In *Proceedings of CP*, volume 4741 of *LNCS*, pages 118–132. Springer, 2007.
- [2] N. Beldiceanu and E. Contejean. Introducing global constraints in CHIP. *Journal of Mathematical and Computer Modelling*, 20(12):97–123, 1994.
- [3] D. Bergman, W.-J. van Hove, and J. N. Hooker. Manipulating MDD Relaxations for Combinatorial Optimization. In *Proceedings of CPAIOR*, volume 6697 of *LNCS*, pages 20–35. Springer, 2011.
- [4] S. Brand, N. Narodytska, C.G. Quimper, P. Stuckey, and T. Walsh. Encodings of the sequence constraint. In *Proceedings of CP*, volume 4741 of *LNCS*, pages 210–224. Springer, 2007.
- [5] K. Cheng and R. Yap. Maintaining Generalized Arc Consistency on Ad Hoc r-Ary Constraints. In *Proceedings of CP*, volume 5202 of *LNCS*, pages 509–523. Springer, 2008.
- [6] M.R. Garey and D.S. Johnson. *Computers and Intractability - A Guide to the Theory of NP-Completeness*. Freeman, 1979.

- [7] T. Hadzic, J.N. Hooker, B. O’Sullivan, and P. Tiedemann. Approximate Compilation of Constraints into Multivalued Decision Diagrams. In *Proceedings of CP*, volume 5202 of *LNCS*, pages 448–462. Springer, 2008.
- [8] T. Hadzic, J.N. Hooker, and P. Tiedemann. Propagating Separable Equalities in an MDD Store. In *Proceedings of CPAIOR*, volume 5015 of *LNCS*, pages 318–322. Springer, 2008.
- [9] T. Hadzic, E. O’Mahony, B. O’Sullivan, and M. Sellmann. Enhanced Inference for the Market Split Problem. In *Proceedings of ICTAI*, pages 716–723. IEEE, 2009.
- [10] P. Hawkins, V. Lagoon, and P.J. Stuckey. Solving Set Constraint Satisfaction Problems Using ROBDDs. *JAIR*, 24(1):109–156, 2005.
- [11] S. Hoda. *Essays on Equilibrium Computation, MDD-based Constraint Programming and Scheduling*. PhD thesis, Carnegie Mellon University, 2010.
- [12] S. Hoda, W.-J. van Hove, and J.N. Hooker. A Systematic Approach to MDD-Based Constraint Programming. In *Proceedings of CP*, volume 6308 of *LNCS*, pages 266–280. Springer, 2010.
- [13] W.-J. van Hove, G. Pesant, L.-M. Rousseau, and A. Sabharwal. Revisiting the Sequence Constraint. In *Proceedings of CP*, volume 4204 of *LNCS*, pages 620–634. Springer, 2006.
- [14] W.-J. van Hove, G. Pesant, L.-M. Rousseau, and A. Sabharwal. New Filtering Algorithms for Combinations of Among Constraints. *Constraints*, 14:273–292, 2009.
- [15] M. Maher, N. Narodytska, C.-G. Quimper, and T. Walsh. Flow-Based Propagators for the SEQUENCE and Related Global Constraints. In *Proceedings of CP*, volume 5202 of *LNCS*, pages 159–174. Springer, 2008.
- [16] J.-C. Régin. A Filtering Algorithm for Constraints of Difference in CSPs. In *Proceedings of AAAI*, volume 1, pages 362–367. AAAI Press, 1994.
- [17] J.-C. Régin and J.-F. Puget. A Filtering Algorithm for Global Sequencing Constraints. In *Proceedings of CP*, volume 1330 of *LNCS*, pages 32–46. Springer, 1997.