Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17)

# Soft and Cost MDD Propagators

**Guillaume Perez** and **Jean-Charles Régin**
Université Nice Sophia Antipolis, CNRS, I3S, France
guillaume.perez06@gmail.com, jcregin@gmail.com

## Abstract

Recent developments of efficient propagators, operations and creation methods for MDDs allow us to directly build efficient MDD-based models, without the need for intermediate data structures. In this paper, we take another step in this direction by improving the propagators of cost MDDs. In addition, we introduce a soft MDD propagator in order to deal with unsatisfiable problems. This directly offers cost and soft versions for table constraints and any constraints which can be represented by an MDD (regular, slide, knapsack...).

## Introduction

Multi-valued decision diagrams (MDDs) are efficient data structures for representing discrete functions. They are implemented in almost all constraint programming solvers and are more and more used to build models (Perez and Régin 2015a; Andersen et al. 2007; Hadzic et al. 2008; Hoda, van Hoeve, and Hooker 2010; Bergman, van Hoeve, and Hooker 2011; Gange, Stuckey, and Szymanek 2011). They can be constructed in several ways, from tables, automata, dynamic programming, etc...; or defined by combining two or more MDDs thanks to operators like intersection, union, or difference.

The MDD associated with a constraint $C$ is an MDD which models the set of tuples satisfying $C$. An MDD propagator of $C$ is an algorithm which removes some inconsistent values of $X(C)$, the variables on which $C$ is defined. In this paper, we consider soft and cost MDD propagators, that are propagators for soft constraints represented by MDDs and cost MDDs.

The cost version of an MDD is an MDD whose arcs have an additional information: the cost of the arc. In a cost-MDD, each path from the top layer to the bottom layer has a cost, and a cost-MDD propagator aims at bounding this cost. Cost-MDDs are useful to model optimization problems (Bergman and Cire 2016; Bergman, van Hoeve, and Hooker 2011). Several cost-MDD propagators already exist (Demassey, Pesant, and Rousseau 2006; Gange, Stuckey, and Van Hentenryck 2013). Recently, MDD4R (Perez and Régin 2014) a new propagator for MDDs has been proposed. For some industrial instances, MDD4R improves on previous propagators by a speed factor of up to 100. Hence, we propose an adaptation of the propagator MDD4R to process cost-MDDs and we will show that such speed up are also observed for cost-MDD4R compared to existing methods.

We also consider propagators of soft constraints, which will allow the violation of some arcs, with respect to a certain amount of violation. We give three original propagators, one dedicated, one based on the cost-MDD propagator and one obtained by intersecting the initial MDD with an MDD expressing a sum constraint.

The paper is organized as follows. The next section presents the motivation for this work. Then, we recall some basics about MDDs, cost-MDDs, and propagators for both of them. Next, we present cost-MDD4R, our cost-MDD propagator which is an adaptation of the MDD4R propagator. The section about soft constraints presents three propagators. In the experimental section, we give some results for the complex text generation problem that motivated our work. We also show by using several random instances that our methods improve the previous results. Finally, we conclude.

## Motivation

The main motivation of this work is the generation of text and music from a corpus while avoiding plagiarism (Papadopoulos, Roy, and Pachet 2014; Perez and Régin 2015a).

The goal of this problem, named `maxOrder`, is to generate sequences of words, where for example, each subsequence of size two belongs to the corpus (Markovian transition) and no subsequence of size 4 belongs to the corpus. Here 4 denotes the maximum plagiarism size.

To handle this problem, we define two types of MDDs (Figure 1). First, we extract the Markov transition matrix from the corpus, then we build the MDD representing the Markov transition constraint named $MDD_M$.

Second we build $MDD_{NP}$, the MDD representing all the sequences of size 4 not belonging to the corpus. To do so, we build the MDD of all the sequences of size 4 which belong to the corpus, then we apply the negation operator (Perez and Régin 2015b). An important remark is that the negation of an MDD is linear in its size.

Then, for each sequence of 2 variables we define an MDD propagator on $MDD_M$ and for each sequence of 4 variables we define one MDD propagator on $MDD_{NP}$ (Figure 2).

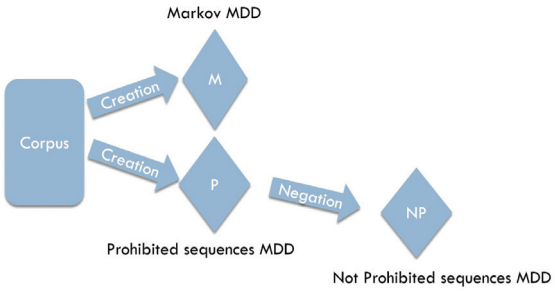


Figure 1: MaxOrder model: Main MDDs

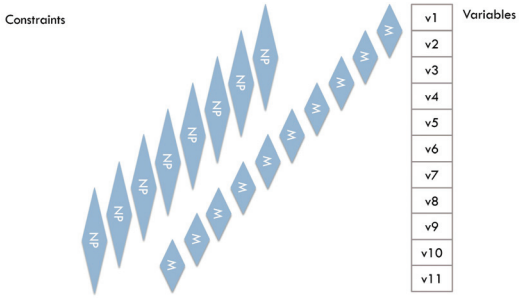


Figure 2: MaxOrder model: Sequences of MDDs

This model gives excellent results in practice and is competitive with an ad-hoc solution.

Only instances with a lot of solutions have been considered. However, when the corpus is too small, or too complex (too many different words compared to the total number of words), there is no solution. In this case, we would like to provide solutions with respect to an amount of violation. Thus, we need to be able to soften either the Markovian transition constraint, or the plagiarism constraint (Figure 3).

The soft MDD propagator can be used in two distinct ways. The first one is for $MDD_M$, the Markov MDD. If a soft MDD propagator is applied to it, then it means that we allow the generation algorithm to create new word transitions (transitions not belonging to the corpus). The second one is for $MDD_{NP}$, the plagiarism MDD. If a soft MDD propagator is applied to it, then this allows the generation algorithm to create sequences containing some plagiarisms. The goal of the solver is then to limit this plagiarism.

## Preliminaries

### Multi-valued decision diagram

A multi-valued decision diagram (MDD) is a data-structure representing discrete functions. It is a multiple-valued extension of BDDs (Bryant 1986). An MDD, as used in CP (Cheng and Yap 2010; Perez and Régin 2014; Andersen et al. 2007; Hadzic et al. 2008; Hoda, van Hoeve, and Hooker 2010; Bergman, van Hoeve, and Hooker 2011; Gange, Stuckey, and Szymanek 2011), is a rooted directed acyclic graph (DAG) used to represent some multi-valued

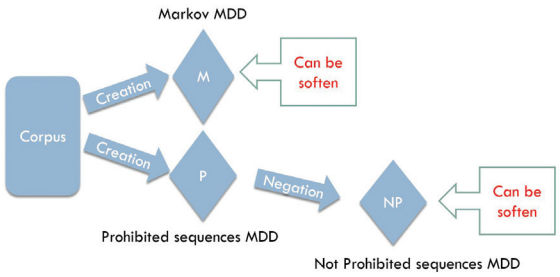


Figure 3: MaxOrder: soft version

function $f : \{0...d-1\}^r \rightarrow \{true, false\}$, based on a given integer $d$. Given the $r$ input variables, the DAG representation is designed to contain $r$ layers of nodes, such that each variable is represented at a specific layer of the graph. Each node on a given layer has at most $d$ outgoing arcs to nodes in the next layer of the graph. Each arc is labeled by its corresponding integer. The arc $(u, v, a)$ is from node $u$ to node $v$ and labeled by $a$. The final layer is represented by the true terminal node (the false terminal node is typically omitted). There is an equivalence between $f(v_1, ..., v_r) = true$ and the existence of a path from the root node to the true terminal node whose arcs are labeled $v_1, ..., v_r$. Nodes without any outgoing arc or without any incoming arc are removed.

**MDD of a constraint.** Let $C$ be a constraint defined on $X(C)$. The MDD associated with $C$, denoted by MDD$(C)$, is an MDD which models the set of tuples satisfying $C$. More precisely, MDD$(C)$ is defined on $X(C)$, such that the labels of arcs of the layer of the variable $x$ correspond to values of $x$, and a path of MDD$(C)$ where $a_i$ is the label of layer $i$ corresponds to a tuple $(a_1, ..., a_n)$ on $X(C)$.

**Consistency with MDD$(C)$.** A value $a$ of the variable $x$ is valid iff $a \in D(x)$. An arc $(u, v, a)$ at layer $i$ is valid iff $a \in D(x_i)$. A path is valid iff all its arcs are valid.
Let $path^r_{tt}$(MDD$(C)$) be the set of paths from root $r$ to $tt$ in MDD$(C)$. The value $a \in D(x_i)$ is consistent with MDD$(C)$ iff there is a valid path in $path^r_{tt}$(MDD$(C)$) which contains an arc at layer $i$ labeled by $a$.

**MDD propagator.** An MDD propagator associated with a constraint $C$ is an algorithm which removes some inconsistent values of $X(C)$. The MDD propagator establishes arc consistency of $C$ if and only if it removes all inconsistent values with MDD$(C)$. This means that it ensures that there is a valid path from the root to the true terminal node in MDD$(C)$ if and only if the corresponding tuple is allowed by $C$ and valid.

MDD4 is a MDD propagator which maintains the whole MDD during the search and the following invariants:

*For each node $u \in MDD$, $\omega^+(u)$ contains the valid arcs outgoing from $u$*, and *$\omega^-(u)$ the valid arcs incoming in $u$.*

$\forall x_i \in X, \forall a \in D(x_i)$, *$S(x_i, a)$ contains all the valid arcs* $e$ at layer $i$ s.t. $e = (u, v, a)$.

When a modification occurs in the domain of a variable, MDD4 deletes all arcs in the $S$ lists of the deleted values,

then it deletes the nodes and arcs which do not belong to a valid path of $path^r_{tt}(M)$. To do so, MDD4 performs two BFS, layer by layer, one from the layer of the modified variable to the top, and the other one from the modified variable to the bottom.

MDD4R is an improved version of MDD4 based on the idea of the reset (Perez and Régin 2014). A reset is the operation which consists in clearing and rebuilding a data structure when it needs less operation than updating it. For example, if for a layer we have to remove 90% of the arcs, we should consider rebuilding the layer from the remaining 10% arcs. By maintaining the number of arcs of each layer, we can know exactly if we should remove the inconsistent arcs, or rebuild the layer using the remaining arcs. This idea is the main advantage of MDD4R and leads to orders of magnitude in performance gain.

Note that a regular constraint (Beldiceanu, Carlsson, and Petit 2004; Pesant 2004) can be converted into an MDD by first unfolding the automaton on the variables, which gives an MDD whose nodes are associated to a state from the automaton, and then reducing this MDD (Perez and Régin 2015a). MDD4R is more efficient in practice than Regular propagators.

## Cost-MDD propagator

**Cost-MDD** A cost-MDD is an MDD whose arcs have an additional information: the cost $c$ of the arc. That is, an arc is a 4-tuplet $e = (u, v, a, c)$, where $u$ is the emanating node, $v$ the terminating node, $a$ the label and $c$ the cost. Let $M$ be a cost-MDD and $p$ be a path of $M$. The cost of $p$ is denoted by $\gamma(p)$ and is equal to the sum of the costs of the arcs it contains. A shortest path of $M$ is a path of $M$ whose cost is minimum. A shortest path of an arc $e$ is $p_{min(e)}$, a path such that there is no path of $M$ containing $e$ having a smaller cost.

**Cost-MDD of a constraint.** Let $C$ be a constraint and $f_C$ be a function associating a cost with each value of each variable of $X(C)$. The cost-MDD of $C$ and $f_C$ is denoted by cost-MDD$(C, f_C)$ and is MDD$(C)$ whose the cost of an arc labeled by $a$ at layer $i$ is $f_C(x_i, a)$.

**Cost-MDD propagator.** A cost-MDD propagator associated with $C$, $f_C$, a value $H$, and a symbol $\prec$ (which can be $\leq$ or $\geq$) is an MDD propagator on MDD$(C)$ which ensures that for each path $p$ of cost-MDD$(C, f_C)$ we have $\gamma(p) \prec H$. A cost-MDD propagator establishes arc consistency of $C$ iff each arc of cost-MDD$(C)$ belongs to $p$ a valid path of $path^r_{tt}$(cost-MDD$(C)$) with $\gamma(p) \prec H$. For the sake of clarity, we will consider only the case $\leq$ in this paper. The other case is equivalent.

Existing propagators (Demassey, Pesant, and Rousseau 2006; Gange, Stuckey, and Van Hentenryck 2013) for this constraint are based on the idea of processing and maintaining `up[u]`, the shortest path cost between the *root* node and every node u, and then `dn[u]`, the shortest path cost between each node u and the *tt* node. An arc $e = (u, v, a, c)$ is deleted when $up[u] + dn[v] + c > H$.

These algorithms use a modified version of their own MDD propagator to handle this new deletion and propagation on the cost-MDD. Basically, when a variable of X is modified, the arcs labeled by the deleted values have to be removed, and, if a node lost all its incoming or outgoing arcs, it has to be removed. When a node is removed, all its remaining arcs have to be removed. If a value `up[u]` or `dn[u]` of a node $u$ is modified, then the new value has to be propagated.

We propose Cost-MDD4R a modified version of the efficient MDD4R algorithm for establishing the arc consistency of a constraint $C$ represented by cost-MDD$(C)$.

In order to deal with costs, cost-MDD4R adds and maintains for each node u, the value `up[u]` and `dn[u]` as previously defined. While a propagator dealing with an MDD only has to manage the variables modifications, a propagator dealing with cost-MDD also needs to handle cost modifications. We detail these operations:

**Variable Modification.** When the domain of a variable is modified, cost-MDD4R performs the same work as MDD4R and maintains for each node $u$ the values `up[u]` and `dn[u]`. To do so, it marks the modified nodes and, between the work made layer by layer, it updates the cost of the modified nodes, and deletes the arcs that just became invalid. Then, it continues this cost propagation, layer by layer, even if no more arcs have to be deleted by MDD4R.

To avoid unnecessary work, we can only mark nodes whose value is equal to the value brought by the deleted arc. For example, if we remove the arc $e = (u, v, a, c)$, we mark $u$ only if `dn[u]` $= c +$ `dn[v]` and we mark $v$ only if `up[v]` $= c +$ `up[u]`.

If a reset is performed (i.e. fewer valid arcs than invalid arcs), then the values `up` and `dn` are recalculated while putting back the arcs.

**Modification of the cost value.** Let $e = (u, v, a, c)$ be an arc, the Minimal Path Cost of $e$ denoted by $MPC(e)$ is $MPC(e) = c +$ `up[u]` $+$ `dn[v]` $= \gamma(p_{min(e)})$.

**Proposition 1** *Let $C$ be an arc consistent constraint with a cost value $H = k+i$. If $H$ is reduced to $k$, then removing all arcs of cost-MDD $(C)$ such that $MPC(e) > k$ is sufficient for $C$ to be arc consistent.*

**Proof:** Any arc $\epsilon$ with $MPC(\epsilon) > k$ is not consistent by definition, and so can be safely deleted. Let $e$ be any arc with $MPC(e) \leq k$. Then, for every arc $e' \in p_{min(e)}$ there is $p'$ a path such that $\gamma(p') \leq \gamma(p_{min(e)}) = MPC(e)$, so $MPC(e') \leq MPC(e) \leq k$. Thus, no arc of $p_{min(e)}$ has been deleted and $p_{min(e)}$ is a valid path of $path^r_{tt}(M)$ . Hence, $e$ is consistent with $C$.

So, when the value $H$ is reduced from $k + i$ to $k$, cost-MDD4R establishes arc consistency by performing a BFS, and for each layer, it simply removes the arcs such that $MPC > k$. This can be efficiently done by maintaining the arcs sorted by their $MPC$.

Here again, we can use the reset idea. When there are less arcs with $MPC \leq k$ than arcs with $MPC > k$, then cost-MDD4R will choose to clear the data structures and put back the arcs with $MPC \leq k$. This is an important part of the algorithm because the bound propagation can be costly. This idea avoids deleting almost all the MDD when only few arcs are still valid.

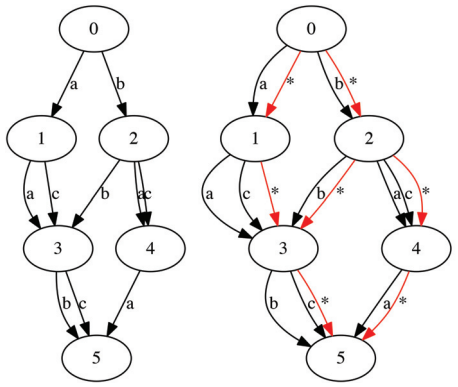


Figure 4: Soft MDD and $scMDD$

## Soft-MDD propagator

A soft constraint is a constraint which allows some violations.

In this paper, we consider only the variable based violation cost (Petit, Régin, and Bessière 2001). Precisely, for a given assignment $A$ of valid values of a constraint $C$, the cost of the violation of $C$ by $A$ is defined as the minimal number of values of $A$ that should be changed in order to satisfy $C$. In other words, it corresponds to the minimum of the Hamming distance between $A$ and any tuple of $C$. We denote by Hamming$(A, C)$ this distance. So, we consider that a value $a \in D(x)$ is consistent with a soft constraint $C$ associated with an integer $H$ if and only if there exists $A$, an assignment of valid values involving $(x, a)$, such that Hamming$(A, C) \leq H$.

A soft-MDD propagator of a soft constraint $C$ associated with an integer $H$ is an algorithm which removes some values inconsistent with $C$ and $H$.

MDD propagators (Cheng and Yap 2010; Perez and Régin 2014) fail when no solution exists. For example, consider the top left MDD of Figure 4. If all the arcs of any path are valid, then the constraint is not violated. But if values a and b are deleted from the domain of the first variable (i.e. the variable of the first layer) then the propagation of this deletion will remove all the nodes and arcs of the MDD. This shows the need for new propagators in order to deal with the amount of violation.

We introduce three methods to propagate a soft constraint $C$ represented by MDD$(C)$. The first one is a simple propagator, which does not modify MDD$(C)$ and uses some properties on the shortest path to establish arc consistency of $C$. The second one transforms MDD$(C)$ into a cost-MDD and uses a cost-MDD propagator on it. The last one builds an MDD explicitly dealing with the violation cost variable and intersects it with MDD$(C)$ and apply on the resulting MDD an MDD propagator.

### Dedicated Propagator

The first propagator does not modify MDD$(C)$ and is easy to implement.

Consider $\mu(p)$ the function which counts the number of arcs of the path $p$ that are not valid.

While filtering, only assignments involving values in the domains of the variables are considered, so the Hamming distance between any assignment and a tuple of the MDD corresponding to the path $p$ is at least $\mu(p)$. Let $e$ be an arc of the MDD. If $e$ is not involved in a path $p$ of $path^r_{tt}(M)$ with $\mu(p) \leq H$ then it means that no path containing $e$ may support a value, so $e$ can be safely deleted. Let $p$ be a path of $path^r_{tt}(M)$. If $\mu(p) = H$ then it means that $p$ supports any value of a variable corresponding to the layer of a non valid arc of $p$, because in an assignment each value belongs to the domain of its variable. If $\mu(p) < H$ then it means that $p$ supports any value of any variable, because we can have one more change in the tuple of the path to correspond to the assignment.

We can design a propagator. First, we search for the shortest path of the MDD according to Function $\mu$. If this path has a cost strictly lower than the maximum cost, then all the values are supported. Otherwise, we delete all arcs $e$ with $MPC(e)) > H$. The resulting MDD can now be handled by any MDD propagator. Using two BFS, we can determine the shortest path cost of all arcs and remove all impossible paths. This method establishes arc consistency of the soft constraint.

Note that a classical cost-MDD cannot handle this constraint by considering $\mu$ as cost function because the cost of an arc depends on the domain of the variables.

This dedicated propagator can be expensive. This is why we propose some other propagators.

### Transformation into a cost-MDD

The conversion of MDD$(C)$ into a cost-MDD uses a method initially created for the regular constraint (Van Hoeve, Pesant, and Rousseau 2006).

The idea is to add, for each two nodes which have at least one arc between them, an additional arc labeled by *, with a cost of 1. The cost of all the other arcs is set to 0. An arc labeled by * is an arc which supports any value of the variable. We call *-arcs such arcs and we denote by $scMDD$ the resulting cost-MDD and $f_{SC}$ the cost function we have defined. Then, we define a cost-MDD propagator on $scMDD$ with $f_{SC}$, $H + 1$ and $<$.

For instance, in Figure 4 the *-arcs (in red) are created only between connected nodes. Nodes 2 and 3 are connected by an arc labeled by b, so we create the *-arc, but 1 and 4 are not connected, so we do not create the *-arc between them. Now, assume that the values a and b are deleted from the domain of the variable x1. The resulting MDD is the MDD in Figure 5. We can see that only 2 arcs have been deleted, and, unlike in MDD$(C)$, the nodes are not deleted, thanks to the *-arcs. It is easy to see that the shortest path cost in this MDD is 1 because all paths contain at least one *-arc.

### Intersection of MDDs

The recent development of efficient operators between MDDs (Perez and Régin 2015a; 2016) allows us to perform several operations between MDDs. In general these operations aim at combining MDDs according to the label of their

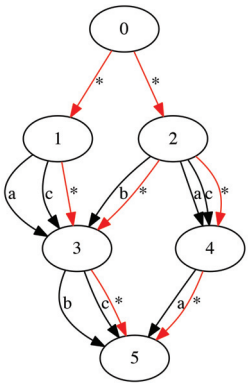


Figure 5: Soft MDD propagation

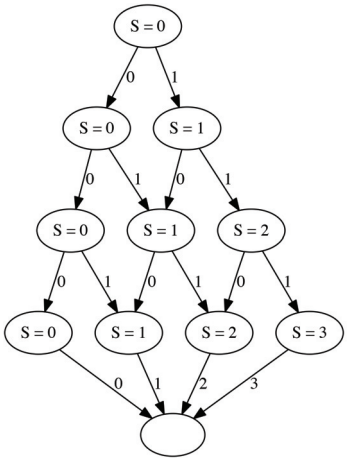


Figure 6: $MDD_{\Sigma\{0,1\}}$ on 3 variables

arcs. In this section, we use these operations in another way. Instead of applying operators on the label of the arcs, we apply them on their cost.

Let $MDD_{\Sigma\{0,1\}}$ (Figure 6) be the MDD representing the sum of $n$ variables with the set $\{0, 1\}$ as a domain. The last variable represents the possible values of this sum. As seen in the previous section, a soft constraint can be expressed as $scMDD$, a cost-MDD in which the cost of the arcs are either 0 or 1. If we perform the intersection between $scMDD$ and $MDD_{\Sigma\{0,1\}}$, then we obtain an MDD whose last layer corresponds to the violation cost. Then, the constraint represented by this MDD is similar to the soft constraint. Thus, any MDD constraint propagator can be used.

For example, if we take the right MDD from Figure 4, and we intersect it with the one from Figure 6, then we obtain the MDD of Figure 7. In the resulting MDD, the node `(1 S=0)` represents the copy of the node `1` from the first MDD having an incoming cost of 0. We can observe that the outgoing arcs of node `(1 S=X)` are still directed to a node labeled by `(3 S=X)`.

We can compute the size of the newly created MDD. Let M be any MDD, we denote by $|M.layer(i)|$ the number of nodes at the layer $i$ and by $|M|$ the total number of nodes of $M$. The maximum number of nodes at a layer i of the intersection of $scMDD$ and $MDD_{\Sigma\{0,1\}}$ is bounded by $|scMDD.layer(i)| * |MDD_{\Sigma\{0,1\}}.layer(i)|$. The sum of all the layers is bounded by $|scMDD| * \max_i(|MDD_{\Sigma\{0,1\}}.layer(i)|)$. In our case, the maximum for a layer of $MDD_{\Sigma\{0,1\}}$ is the number of variables plus one, that is $n + 1$. So the resulting MDD has a maximum size of $(n + 1)|scMDD|$

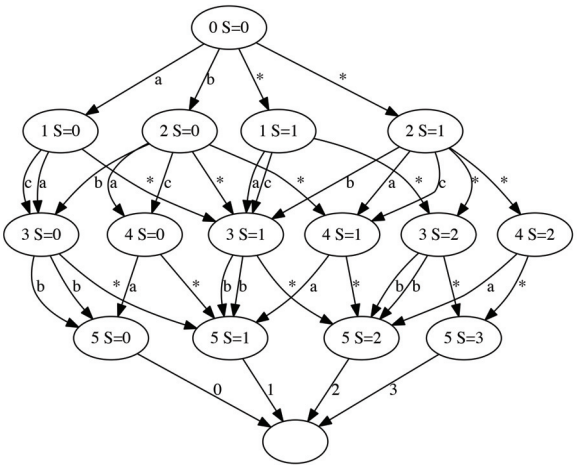


Figure 7: MDD resulting from the intersection of the right MDD of Figure 4 and the one from Figure 6

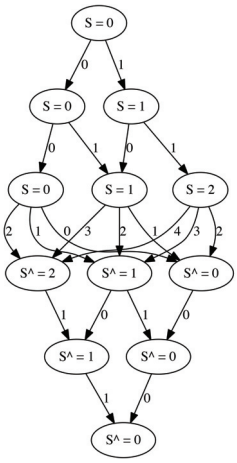


Figure 8: $MDD_{\Sigma\{0,1\}}$ on 4 variables

Note that we can reduce the size of $|MDD_{\Sigma\{0,1\}}|$, by modeling $\sum_i x_i = S$, the sum constraint, in a different way. Instead of ordering the variables from $x_1$ to $x_n$ and to $S$ the final sum variable,we can order the variable from $x_1$ to $x_{\frac{n}{2}}$ then $S$ and then from $x_{\frac{n}{2}+1}$ to $x_n$. This leads to an MDD having half as many nodes for its largest layer (Figure 8).

| Algo | Markov | | | Plagiarism | | |
|---|---|---|---|---|---|---|
| size | 18 | 20 | 22 | 18 | 20 | 22 |
| inter | 5,5 | 104,8 | 111,7 | **4,7** | **8,1** | **9,3** |
| cost-MDD4R | **5,3** | **86,5** | **94,9** | 23,7 | 44,6 | 67,9 |
| ev-mdd | 11,1 | 361,9 | 355,5 | 26,2 | 58,5 | 78,0 |

Table 1: Times needed to build the sequences with minimum of violations (Time out 1800s).

| | Markov | | Plagiarism | |
|---|---|---|---|---|
| | #nodes | #arcs | #nodes | #arcs |
| original | 73 | 168 | 261 | 21.5k |
| arc * | 73 | 380 | 261 | 22.1k |
| intersection | 147 | 590 | 783 | 54.6k |

Table 2: Size of the MDDs. 60 different words.

## Discussion

The intersection based method could also be applied to any constraint represented by a cost-MDD. However, in general the sum is not bounded by a small number. In the best representation of $MDD_{\Sigma[0,k]}$, the number of nodes of the middle layer is the number of different values reachable by summing the numbers between 0 and k, which is $nk/2$. The intersection of an MDD $M$ with $MDD_{\Sigma[0,k]}$ leads to an MDD having at most $nk|M|/2$ nodes. Note that this number is an upper bound, because the number of time a node is duplicated in the intersection is equal to the number of different values of the sums reaching this node. If this number is not too large, then applying this transformation is a good idea because MDD propagators are faster than cost-MDD propagators.

## Experiments

We compare cost-MDD4R with *ev-mdd*, the incremental algorithm presented in (Gange, Stuckey, and Van Hentenryck 2013) and with *inter*, the intersection method we proposed.

### Sequence generation

We consider the problem detailed in Section Motivation. We have tested both ways of softening the constraint and they both gives pertinent results. For the experimentation, we used `"The fables of Jean de La Fontaine"` because they contain several sentences, not too many words and often produce funny results.

An important remark is that, if the corpus size grows, then the `maxOrder` constraint becomes satisfiable. If it grows again, then it becomes useless to apply a `maxOrder` constraint because it becomes exponentially improbable to build a sequence containing plagiarism. That's why we focus on corpus like fables and short texts.

Table 1 gives the time results (in seconds) and Table 2 gives the size of the MDDs. Note that the model also contains an alldifferent constraint. `Markov` means that we apply the soft constraint on the Markovian transition, `Plagiarism` is for the plagiarism part. The creation time is similar for both MDDs, and insignificant compared to the search time. These tables show that both methods are useful, and that our algorithms clearly outperform the existing methods. The MDD representing the Markov is smaller than the MDD of the Plagiarism. We can see that both MDDs are at least twice as big after the intersection.

### Random instances

We test the propagators on several random instances, in order to detect the location of the best performances of each propagator. We select randomly a certain number of tuples and build an MDD from this tuple set. We associate each arc with a random cost between 0 and 10. This implies the use of $MDD_{\Sigma[0,10]}$ instead of $MDD_{\Sigma\{0,1\}}$ for the intersection method. We have a constraint of arity 18 and an allDifferent constraint. Table 3 shows that the intersection method can be very efficient in practice. Intuitively, the intersection method "precomputes" some operations, that are recomputed each time by the cost-MDD propagator. However, when the MDD grows up to reach our memory limit (around 1.7GB), then cost-MDD4R become faster than the intersection method. This comes from memory swaps.

| #tuples | cost-MDD4R | ev-mdd | inter |
|---|---|---|---|
| 50 | 35,89 | 59,23 | **2,55** |
| 150 | 19,15 | 33,98 | **1,97** |
| 500 | 19,61 | 35,38 | **2,77** |
| 1k | 19,97 | 37,37 | **4,15** |
| 2k | 32,04 | 66,22 | **8,23** |
| 5k | 32,27 | 71,43 | **14,22** |
| 10k | 44,26 | 83,58 | **19,12** |
| 25k | 101,57 | 189,30 | **49,84** |
| 50k | 201,94 | 378,533 | **150,316** |
| 100k | **478,296** | 755,570 | 1668,508 |

Table 3: search for the best solution (construction time is included, arity 18, domain size 18).

## Conclusion

This paper makes another step in the direction of building advanced constraint programming models using decision diagrams. We have introduced soft and cost propagators. We have adapted the efficient MDD4R propagator to handle costs and we have proposed two efficient ways of dealing with a soft constraint represented by an MDD. We have shown that our methods are efficient in practice and that it can be worthwhile to intersect MDDs and use MDD4R on the intersection instead of using a dedicated or a cost-MDD propagator.

## References

Andersen, H. R.; Hadzic, T.; Hooker, J. N.; and Tiedemann, P. 2007. A constraint store based on multivalued decision diagrams. In *CP*, 118–132.

Beldiceanu, N.; Carlsson, M.; and Petit, T. 2004. Deriving filtering algorithms from constraint checkers. In *CP'04*, 107–122.

Bergman, D., and Cire, A. A. 2016. Decomposition based on decision diagrams. In *International Conference on AI and OR Techniques in Constriant Programming for Combinatorial Optimization Problems*, 45–54. Springer International Publishing.

Bergman, D.; van Hoeve, W. J.; and Hooker, J. N. 2011. Manipulating mdd relaxations for combinatorial optimization. In *CPAIOR*, 20–35.

Bryant, R. E. 1986. Graph-based algorithms for boolean function manipulation. *Computers, IEEE Transactions on* 100(8):677–691.

Cheng, K., and Yap, R. 2010. An mdd-based generalized arc consistency algorithm for positive and negative table constraints and some global constraints. *Constraints* 15:265–304.

Demassey, S.; Pesant, G.; and Rousseau, L. 2006. A cost-regular based hybrid column generation approach. *Constraints* 11(4):315–333.

Gange, G.; Stuckey, P.; and Szymanek, R. 2011. Mdd propagators with explanation. *Constraints* 16:407–429.

Gange, G.; Stuckey, P. J.; and Van Hentenryck, P. 2013. Explaining propagators for edge-valued decision diagrams. In *Principles and Practice of Constraint Programming*, 340–355. Springer.

Hadzic, T.; Hooker, J. N.; O'Sullivan, B.; and Tiedemann, P. 2008. Approximate compilation of constraints into multivalued decision diagrams. In *CP*, 448–462.

Hoda, S.; van Hoeve, W. J.; and Hooker, J. N. 2010. A systematic approach to mdd-based constraint programming. In *CP*, 266–280.

Papadopoulos, A.; Roy, P.; and Pachet, F. 2014. Avoiding plagiarism in markov sequence generation. In *Proceeding of the Twenty-Eight AAAI Conference on Artificial Intelligence*, 2731–2737.

Perez, G., and Régin, J.-C. 2014. Improving GAC-4 for table and MDD constraints. In *Principles and Practice of Constraint Programming - 20th International Conference, CP 2014, Lyon, France, September 8-12, 2014. Proceedings*, 606–621.

Perez, G., and Régin, J.-C. 2015a. Efficient operations on mdds for building constraint programming models. In *International Joint Conference on Artificial Intelligence, IJCAI-15*, 374–380.

Perez, G., and Régin, J.-C. 2015b. Efficient operations on MDDs for building constraint programming models. *International Joint Conference on Artificial Intelligence, IJCAI-15, Argentina*.

Perez, G., and Régin, J.-C. 2016. Constructions and in-place operations for MDDs based constraints. In *Integration of AI and OR Techniques in Constraint Programming*. Springer International Publishing. 279–293.

Pesant, G. 2004. A regular language membership constraint for finite sequences of variables. In *Proc. CP'04*, 482–495.

Petit, T.; Régin, J.-C.; and Bessière, C. 2001. Specific filtering algorithms for over-constrained problems. In *Proceedings CP'01*, 451–465.

Van Hoeve, W.-J.; Pesant, G.; and Rousseau, L.-M. 2006. On global warming: Flow-based soft global constraints. *Journal of Heuristics* 12(4-5):347–373.