Building efficient soft and cost MDD constraints

Guillaume Perez and Jean-Charles Régin

Université Nice-Sophia Antipolis, I3S UMR 6070, CNRS, 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 of intermediate data structures. In this paper, we make another step in this direction by defining efficient methods to handle cost and soft versions of the MDD constraint. First, we improve the existing algorithms, then we propose to perform several graph operations that reformulate both constraints as a classical MDD constraint. This directly offers cost and soft versions for table constraints and any constraints which can be viewed as an MDD (regular, slide, knapsack, grammar...).

1 Introduction

Multi-valued decision diagrams (MDDs) have a significant place in today's solver, they are implemented in almost all of them, and are more and more used to build models [13, 1, 9, 10, 3, 7]. They can be constructed by several ways, from table, automata, dynamic programming, etc. Another reason of their use is the existence of several operators allowing to build MDDs by combining two or more MDDs.

In this paper, we consider cost-MDD, the cost version of an MDD, which is an MDD whose edges have an additional information: the cost of the edge. In a cost-MDD, each path from the top layer to the bottom layer has a cost, and a cost-MDD constraint aims at bounding this cost. Cost-MDDs are useful to model optimisation problems [2, 3]. Several algorithms already exist in Constraint Programming for the cost-MDD constraint [6, 8], they are an adaptation of existing algorithms for MDD constraints. We propose an adaptation of MDD4R [12] to process to cost-MDD constraint, because MDD4R is one of the best MDD propagators, for which we have observed speed up factor of 100 for some industrial applications. We will show that such speed up are also observed to cost-MDD4R compared to existing methods.

We also consider the soft version of an MDD constraint, which will allow to violate some edges, with respect to a certain amount of violation. Cost and soft version of propagators add flexibility for the modeler.

The main motivation of this work is the generation of text and music from corpus avoiding plagiarism [11, 13]. This problem has been focused on when a lot of solutions exist. However when the corpus is too small, or too complex (too many different words compared to the total number of words), classical algorithms do not find any solution. In this case, we would like to provide solutions with respect to an amount of violation. An important remark is that, if the corpus size grows, then the problem becomes satisfiable.

If it grows again, then it becomes useless to apply a plagiarism constraint because it becomes exponentially improbable to build a sequence containing plagiarism.

The paper is organized as follows. The next section recalls some basics about MDDs, cost-MDD, constraints for both of them and soft constraint. Section 3 presents the related work and cost-MDD4R, our cost-MDD propagator which is an adaptation of the MDD4R propagator. Section 4 presents three methods to process the soft MDD constraint, notably a transformation of the soft-MDD constraint into a classical MDD constraint. In the Experimental section, we show how we can modify the model of the text generation problem in order to deal with violation, and show our experimental results. We also show by using several random instances that our methods are very efficient for both of these problems. Finally, we conclude.

2 Preliminaries

2.1 Definitions

MDD. Multi-valued decision diagram (MDD) is a data-structure for representing discrete functions. It is a multiple-valued extension of BDDs [4]. An MDD, as used in CP [5, 12, 1, 9, 10, 3, 7], is a rooted directed acyclic graph (DAG) used to represent some multi-valued 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 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.

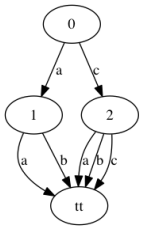

Fig. 1. An MDD representing the tuple set {(a,a),(a,b),(c,a),(c,b),(c,c)}

MDD constraint. In an MDD constraint, the MDD models the set of tuples satisfying the constraint, such that every path from the root to the true terminal node corresponds to an allowed tuple. Each variable of the MDD corresponds to a variable of

the constraint. An arc associated with an MDD variable corresponds to a value of the corresponding variable of the constraint. An example of MDD is given in Figure 1. It represents the tuples (a,a), (a,b), (c,a), (c,b) and (c,c). For each tuple, there is a path from the root node (node 0) to the terminal node (node tt) whose are labeled by the tuple values.

For convenience, we will denote by d the maximum number of values in the domain of a variable; and an arc from u to v labeled by a will be denoted by (u, v, a). We denote by $path(M)$ the set of paths from $root$ to tt of the MDD M.

Definition 1 *Let $M = (N, E)$ be an MDD, X be a set of variable. Let $p = \{e_1, e_2, ..., e_n\}$.*

$$valid(p) \iff \forall e_i = (u_i, v_i, a_i) \in p, a_i \in dom(x_i) \wedge p \in path(M). \quad (1)$$

Definition 2 *Let X be a set of variables and M be an MDD.*
*The constraint **MDDConstraint(X, M)** ensures that:*

$$\exists p, valid(p). \quad (2)$$

Definition 3 *Let X be a set of variables and M be an MDD.*
*The constraint **MDDConstraint(X, M)** is Arc-Consistent (AC) iff:*

$$\forall x_i \in X, \forall a \in dom(x_i), \exists p = \{e_1, e_2, ..., e_n\}, e_i = (u_i, v_i, a) \wedge valid(p). \quad (3)$$

Cost-MDD. A cost-MDD is an MDD whose edges have an additional information: the cost c of the edge. That is, edges are 4-uplet $e = (u, v, a, c)$, where u is the emanating node, v the terminating node, a the label and c the cost.

Definition 4 *Let $M = (N, E)$ be a cost-MDD, let $p = \{e_1, e_2, ..., e_n\}$ be a path of M with $e_i = (u_i, v_i, a_i, c_i)$. C is a path cost function, defined by*

$$C(p) = \sum_{i=1}^{n} c_i \quad (4)$$

Definition 5 *Let $M = (N, E)$ be a cost-MDD. A shortest path of M with respect to C, is a path noted $p_{min(M)}$ such that:*

$$\nexists\, p' \in path(M), p' \neq p_{min(M)} \wedge C(p') < C(p_{min(M)}). \quad (5)$$

Definition 6 *Let $M = (N, E)$ be a cost-MDD. A shortest path of an edge $e \in E$ is a path noted $p_{min(e)}$ such that:*

$$e \in p_{min(e)}, \nexists\, p' \in path(M), e \in p' \wedge p' \neq p_{min(e)} \wedge C(p') < C(p_{min(e)}). \quad (6)$$

Definition 7 *Let $M = (N, E)$ be a cost-MDD. Let $\underline{C}(M)$ (resp. $\overline{C}(M)$) be the minimal (resp. maximal) shortest path cost in M:*

$$\underline{C}(M) = min_{e \in E}(C(p_{min(e)})) \quad (7)$$

$$\overline{C}(M) = max_{e \in E}(C(p_{min(e)})) \quad (8)$$

Cost-MDD constraint.

Definition 8 *Let X be a set of variables, M = (N, E) be a cost-MDD and H be an integer value. The constraint* ***cost-MDDConstraint(X, M, H)*** *ensures that:*

$$\exists p, valid(p) \wedge C(p) \leq H. \tag{9}$$

Definition 9 *Let X be a set of variables, M = (N, E) be a cost-MDD and H be an integer value. The constraint* ***cost-MDDConstraint(X, M, H)*** *is AC iff:*

$$\forall x_i \in X, \forall a \in dom(x_i), \exists p = \{e_1, e_2, ..., e_n\}, e_i = (u_i, v_i, a, c_i) \wedge valid(p) \wedge C(p) \leq H. \tag{10}$$

Several algorithms exist for propagating the cost-MDD constraint [6, 8], they will be introduced in the related work section.

Soft constraint. Soft-constraints are constraints that can be violated, with respect to a violation variable which measures the violation. The violation measurement can be different depending on the constraint. For example, for a table, we can consider the minimal Hamming distance to a tuple from the table. In general the (weighted) sum of violation variables is minimized, because a good solution that minimizes the number of violation.

Soft MDD constraint A soft MDD constraint is an MDD constraint which allows the violation of $valid(p)$. We assume that the amount of violation is the number of value of the path p not in the domain of the variable.

Definition 10 *Let e be an edge. Let $cost_d()$ be a cost function.*

$$cost_d(e) = \begin{cases} 0 \text{ if } label(e) \in dom(var(e)) \\ 1 \text{ otherwise} \end{cases}$$

Definition 11 *Let X be a set of variables, M be an MDD and H be an integer value. The constraint* ***softMDDConstraint(X, M, H)*** *ensures that:*

$$\exists p = \{e_1, e_2, ..., e_n\}, \sum_{i=1}^{n} cost_d(e_i) \leq H. \tag{11}$$

Definition 12 *Let X be a set of variables, M be an MDD and H be an integer value. The constraint* ***softMDDConstraint(X, M, H)*** *is AC iff:*

$$\forall x_i \in X, \forall a \in dom(x_i), \exists p = \{e_1, e_2, ..., e_n\}, \sum_{i=1}^{n} cost_d(e_i) \leq H. \tag{12}$$

3 Cost-MDD constraint.

3.1 Related work

Existing propagators [6, 8] for the cost-MDD 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 edge $e = (u, v, a, c)$ is deleted when $up[u] + dn[v] + c > H$.

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

3.2 Cost-MDD4R Propagator

In this section, we propose a modified version of the MDD4R algorithm [12] to handle a cost-MDD constraint. This is motivated by the fact that MDD4R is a very efficient MDD propagator and is already implemented in several CP solvers. First, we will recall some basics about MDD4R, then we will introduce cost-MDD4R, the cost-MDD propagator.

MDD4 is a propagator for the MDDConstraint(X, M) with X a set of variables and $M = \{N, E\}$ an MDD. MDD4 maintains the whole MDD during the search. It maintains the three following invariants:

$\forall u \in N, \omega^+(u)$ *contains all the valid edges outgoing from* u.

$\forall u \in N, \omega^-(u)$ *contains all the valid edges incoming in* u.

$\forall x \in X, \forall a \in dom(x), S(x, a)$ *contains all the valid edges* e s.t. $e = (u, v, a)$.

When a modification occurs in the domain of a variable, MDD4 deletes all edges in the ω lists of the deleted values, then it deletes the nodes and edges which do not belong to a valid path. To do so, MDD4 performs two BFS, layer by layer, one from the layer of the modified variable until 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. A reset [12] is an 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 edges, we should consider rebuilding the layer from the 10% remaining edges. So by using this idea and just by maintaining the number of edges of each layer, we can know exactly if we should remove the inconsistent edges, or rebuild the layer using the remaining edges. This idea is the main advantage of MDD4R and leads to orders of magnitude in performances gain.

Introduction of costs. MDD4R does not deal with any cost. To do so, cost-MDD4R adds and maintains for each node u, the value `up[u]` and `dn[u]` as defined in the related work section.

Definition 13 *Let $e = (u, v, a, c)$ be an edge, the Minimal Path Cost denoted by $MPC(e)$ is $MPC(e) = c + up[u] + dn[v] = C(p_{min(e)})$.*

Cost-MDD4R. While the MDD propagator only has to handle one callback for the variables, a cost-MDD propagator needs to handle cost modifications. These two points are the subject of the next two paragraphs.

Variable propagation. When a modification occurs in the domain of a variable, cost-MDD4R can perform the same work as MDD4R, but it has to maintain for each node u the values of `up[u]` and `dn[u]`. To do so, it can mark the modified nodes and, between the work made layer by layer, update the cost of the modified nodes, and delete the edges that just become invalid. Cost-MDD4R may have to continue its cost propagation, layer by layer, even if no more edges have to be deleted by MDD4R.

If a reset is performed (i.e. less valid edges than invalid edges), then the values `up` and `dn` are recalculated while putting back the edges, there is no need of a recalculation between the layer, only the cost propagation part is required for the nodes whose value has been modified.

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

Modification of the cost value. When the upper-bound of the cost H is reduced from $k + i$ to k, cost-MDD4R has to remove all the edges e such that $MPC(e) > k$

Property 1 $\forall e, \forall e' \in p_{min(e)}, MPC(e) \geq MPC(e')$

Corollary 1 *Let $M = (N, E)$ be a cost-MDD, $\forall e, MPC(e) = k \implies \forall e'$ such that $MPC(e') < k$, $MPC_M(e') = MPC_{M \setminus \{e\}}(e')$*

Corollary 2 *Let C be an AC cost-MDD constraint with a cost-variable upper-bound of $k + i$. If the cost upper-bound is reduced to k, then removing all edges such that $MPC_M(e) > k$ is sufficient for C to be AC.*

Proof: The edges with $MPC(e) > k$ are not consistent by definition. And so they have to be deleted. From Corollary 1, the other edges remain consistent, because their costs do not change.

Using these properties, when the value H is reduced from $k + i$ to k, cost-MDD4R performs a BFS, and for each layer, it will simply remove the edges such that $MPC > k$. This can be efficiently done by maintaining the edge sorted by their MPC.

Here again, we can use the reset idea. When there is less edges whose $MPC \leq k$ than edges with $MPC > k$, then cost-MDD4R will chose to clear the data structures and put back the edges 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 edges are still valid.

4 soft-MDD constraint

MDD constraint propagators [5, 12] fail when no solution exists. A soft-MDD constraint is used to handle the distance of the satisfaction. In this paper, for a soft-MDD constraint, we consider the distance of the satisfaction as the minimum distance with a path of the MDD.

For example, consider the top left MDD from the Figure 2. If all the edges from a 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 edges of the MDD. This shows that we need to introduce new propagators in order to deal with the amount of violation.

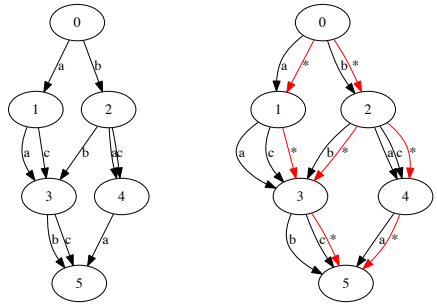

Fig. 2. Soft MDD creation and uses

We will now introduce three methods to handle soft-MDD constraints. The first one is a simple propagator, which does not modify the MDD and use some properties on shortest path to valid the values. The second one is a transformation of the MDD in order to allow a cost-MDD constraint to handle the soft-MDD constraint. The last one is another transformation of the MDD, in order to allow a classical MDD propagator to handle a soft-MDD constraint.

4.1 First propagator

The first propagator we introduce for the soft-MDD constraint does not modify the MDD and is really easy to implement. The $cost_d$ cost function is the one from definition 10. This propagator is based on the following properties:

Property 2 *The lower Bound of the soft constraint is the shortest path from the root node to the tt node using the $cost_d$ cost function.*

Property 3 *If it exists a path of cost k, then it exists a path of cost $\in$ [k-1, k+1] for any value of any variable.*

Proof: The k+1 value is given by the replacement of one of the values of the path of cost 0 by any other value. The cost of any other value is either 0 or 1. So the cost will be k or k+1. The same idea for k-1 applies.

Corollary 3 *A path with cost strictly lower than H supports all the values of all the variables.*

A first remark is that a classical cost-MDD cannot handle this constraint because the cost of an edge depends on the domain of the variables.

This propagator searches for the shortest path of the MDD according to the $cost_d$ cost function. If a path as a cost strictly lower than the maximum cost, then all the values are supported. Otherwise, we delete all edges 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 edges and removing all impossible paths.

This method can be expensive. This is why we propose a second method for the soft MDD constraint. This method uses two steps, the first step converts the soft-MDD constraint into a cost-MDD constraint. The second step converts the cost-MDD constraint into a simple MDD constraint.

4.2 soft-MDD as a cost-MDD

The conversion of a soft-MDD constraint into a cost-MDD constraint uses the method defined in [15] initially created for the regular constraint.
The idea is to add, for each two nodes which have at least one edge between them, an additional edge labeled by *, with a cost of 1. The cost of all the other edges is set to 0. An edge labeled by * is an edge which supports any value of the variable and we call them *-edges. Algorithm 1 describes the method. We can now apply a cost-MDD constraint, or convert this cost-MDD constraint into a simple MDD constraint using the next step.

As we can see in the top right MDD in figure 2, the *-edges (dotted) are created only between connected nodes. Nodes 2 and 3 are connected by an edge labeled by b, so we can create the *-edge, but 1 and 4 are not connected, so we cannot create the *-edge between them.
For instance, consider the variables x1, x2 and x3 having the set $\{$a,b,c$\}$ as domain. We can create a cost-MDD constraint taking the top-right MDD of figure 2 and the variables x1, x2 and x3 in this order. If the values a and b are deleted from the domain of the variable x1, the resulting MDD is the MDD in the figure 3. We can see that only 2 edges have been deleted, and contrary to basic MDD constraints, the nodes are not deleted, thanks to the *-edges. It is easy to see that the shortest path cost in this MDD is 1 because all the path contain at least one *-edge.

4.3 soft-MDD as an MDD via intersection

The recent development of efficient operations between MDDs [13, 14] allows us to perform several operations between MDDs. In general these operations aim at combining MDDs according to the label of their edges. In this section, we use these operations

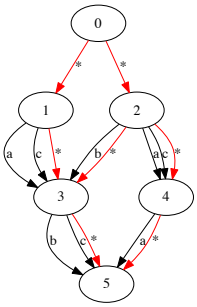

Fig. 3. Soft MDD propagation

Algorithm 1: *-edges creation in an MDD.

```
*-EDGES_CREATION(mdd)
    for each n ∈ mdd.V do
        for each n' in neighbors+(n) do
            CREATE_EDGE(n, n', *, 1)
```

in another way. Instead of applying operators on the label of the edges, we apply them on their cost.

Let $MDD_{\Sigma\{0,1\}}$ (Figure 4) be the MDD representing the sum of n variable with the set $\{0, 1\}$ as domain. It is easy to prove that the maximum of this sum is n. The last variable represents the possible value of this sum. As seen in the previous section, a soft-MDD constraint can be expressed as a cost-MDD constraint, the cost of the edges are either 0 or 1. If we perform the intersection between our cost-MDD with $MDD_{\Sigma\{0,1\}}$, we will obtain an MDD with the cost var at its last layer. Now any MDD propagator can handle this MDD.

For example, if we take the right MDD of figure 2, and we intersect this MDD with the one of figure 4, then we will obtain the MDD of the figure 5. 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 edges of nodes `(1 S=X)` arc still directed to a node labeled by `(3 S=X)`.

For convenience, we denote by $mdd.layer(i)$ the set of nodes at the layer i, by $|mdd.layer(i)|$ the number of nodes at the layer i and by $|M|$ the number of nodes of the MDD M.

About the size of this new MDD, the maximum number of time a node in a soft-MDD can be duplicated is bounded by its $depth + 1$. The proof is that in an intersection of two MDDs, the maximum number of nodes at a layer i is bounded by $|mdd.layer(i)| * |MDD_{\Sigma\{0,1\}}.layer(i)|$. Knowing that $|MDD_{\Sigma\{0,1\}}.layer(i)| = i$,

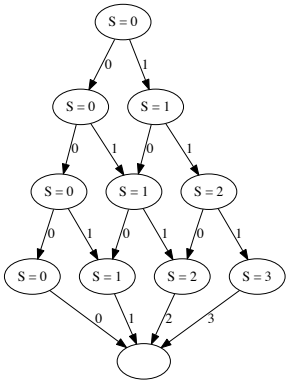

Fig. 4. $MDD_{\Sigma\{0,1\}}$ of size 3

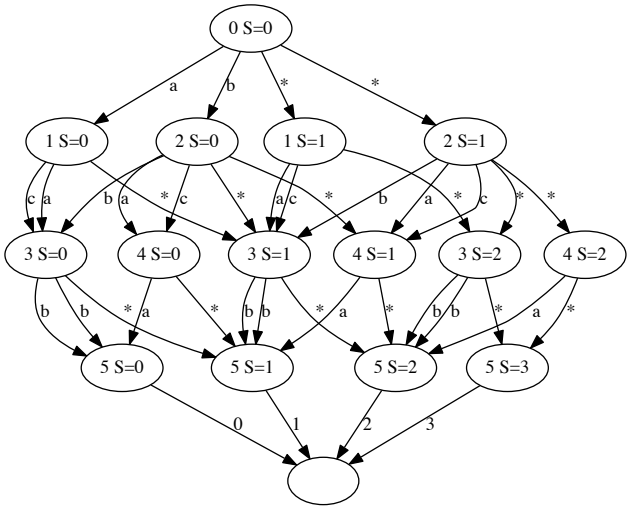

Fig. 5. MDD resulting from the intersection of the right MDD of the figure 2 and the one from figure 4

we can say that the total number of nodes is bounded by $n * |mdd|$. The size of our new MDD is bounded by $n * |M|$. Here M is the transformation previously defined of a soft-MDD into a cost-MDD.

This method could also be applied to any cost-MDD constraint. However, in general the sum is not bounded by a small number. When it is, applying this transformation is a good idea because MDD propagators are faster than cost-MDD propagators. At last, this intersection method gives a way to have an AC propagator on the cost-variable of the cost-MDD constraint, contrary to the other propagators.

5 Experiments

In this section we compare cost-MDD4R with the algorithm presented in [8] (ev-mdd) and with the intersection method from section 4.3 (inter).

Sequence generation We focus on a recent work on sequence generation based on corpus [11, 13] named `maxOrder`. The goal 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, the authors perform graph operations to build a structure representing the solutions of this problem. When no solution exists, they obtain an empty graph.

If one wants to obtain solutions with respect to some violations, one should handle the problem differently. We propose to split the problem, in order to be able to soften either the Markovian transition constraint, or the plagiarism constraint. So we will create two distinct MDDs.

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_p, 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 [13]. An important remark is that the negation of an MDD is linear in its size.

Here the soft MDD constraint can be used in two distinct ways. The first one is for MDD_M, the Markov MDD, if a soft MDD constraint is applied for it, then we allow the generation algorithm to create new word transition (transition not belonging to the corpus).

The second one is for MDD_p, the plagiarism MDD, this allows the generation algorithm to create sequences containing some plagiarism. The goal of the solver is then to limit this plagiarism.

We have tested both of these soft ways, 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 results. (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 close for both MDDs, and insignificant compared to the search time. This table shows that both methods are useful, and that our algorithms clearly outperform the existing methods.

Table 1. Times needed to build the sequences with minimum of violation (T-0 1800s).

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

Random instances In this part, we test our algorithms with several random instances, in order to detect the location of the best performance of each algorithm. In our experiments, we associate each edge with a random cost between 0 and 10. We have a cost-MDD constraint of arity 18, several other constraints like allDifferent etc... The first experiment in Table 2 shows that the intersection method can be very efficient in practice.However, when the MDD grows up to reach our memory limit (around 1.7GB), then cost-MDD4R become faster than the intersection method

Table 2. search for the best solution (construction time is included, arity 18, domain size 18).

#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**

6 Conclusion

This paper makes another step in the direction of building advanced constraint programming models using decision diagrams. We have introduced methods that can be used

by several constraint programming solvers which have already implemented an MDD package. We have also shown how to model a text generation problem using this soft MDD. Finally, we have shown that our methods are efficient in practice.

7 Acknowledgments

We would like to thank A. Papadopoulos and the members of Sony CSL for the help during the construction of the benchmarks.

References

1. Henrik Reif Andersen, Tarik Hadzic, John N. Hooker, and Peter Tiedemann. A constraint store based on multivalued decision diagrams. In *CP*, pages 118–132, 2007.
2. David Bergman and Andre A Cire. Decomposition based on decision diagrams. In *International Conference on AI and OR Techniques in Constriant Programming for Combinatorial Optimization Problems*, pages 45–54. Springer International Publishing, 2016.
3. David Bergman, Willem Jan van Hoeve, and John N. Hooker. Manipulating mdd relaxations for combinatorial optimization. In *CPAIOR*, pages 20–35, 2011.
4. Randal E Bryant. Graph-based algorithms for boolean function manipulation. *Computers, IEEE Transactions on*, 100(8):677–691, 1986.
5. K. Cheng and R. Yap. An mdd-based generalized arc consistency algorithm for positive and negative table constraints and some global constraints. *Constraints*, 15, 2010.
6. Sophie Demassey, Gilles Pesant, and Louis-Martin Rousseau. A cost-regular based hybrid column generation approach. *Constraints*, 11(4):315–333, 2006.
7. G. Gange, P. Stuckey, and Radoslaw Szymanek. Mdd propagators with explanation. *Constraints*, 16:407–429, 2011.
8. Graeme Gange, Peter J Stuckey, and Pascal Van Hentenryck. Explaining propagators for edge-valued decision diagrams. In *Principles and Practice of Constraint Programming*, pages 340–355. Springer, 2013.
9. Tarik Hadzic, John N. Hooker, Barry O'Sullivan, and Peter Tiedemann. Approximate compilation of constraints into multivalued decision diagrams. In *CP*, pages 448–462, 2008.
10. Samid Hoda, Willem Jan van Hoeve, and John N. Hooker. A systematic approach to mdd-based constraint programming. In *CP*, pages 266–280, 2010.
11. A. Papadopoulos, P. Roy, and F. Pachet. Avoiding plagiarism in markov sequence generation. In *Proceeding of the Twenty-Eight AAAI Conference on Artificial Intelligence*, pages 2731–2737, 2014.
12. Guillaume Perez and Jean-Charles Régin. 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*, pages 606–621, 2014.
13. Guillaume Perez and Jean-Charles Régin. Efficient operations on mdds for building constraint programming models. *International Joint Conference on Artificial Intelligence, IJCAI-15, Argentina*, 2015.
14. Guillaume Perez and Jean-Charles Régin. Constructions and in-place operations for mdds based constraints. In *Integration of AI and OR Techniques in Constraint Programming*, pages 279–293. Springer International Publishing, 2016.
15. Willem-Jan van Hoeve. *Operations Research Techniques in Constraint Programming*. PhD thesis, CWI / ILLC, University of Amsterdam, 2005.