MDDs: Sampling and Probability Constraints

Guillaume Perez and Jean-Charles Régin
guillaume.perez06@gmail.com, jcregin@gmail.com

Université Nice-Sophia Antipolis, I3S UMR 7271, CNRS, France

Abstract. We propose to combine two successful techniques of Artificial Intelligence: sampling and Multi-valued Decision Diagrams (MDDs). Sampling, and notably Markov sampling, is often used to generate data resembling to a corpus. However, this generation has usually to respect some additional constraints, for instance to avoid plagiarism or to respect some rules of the application domain. We propose to represent the corpus dependencies and these side constraints by an MDD and to develop some algorithms for sampling the solutions of an MDD while respecting some probabilities or a Markov chain. In that way, we obtain a generic method which avoids the development of ad-hoc algorithms for each application as it is currently the case. In addition, we introduce new constraints for controlling the probabilities of the solutions that are sampled. We experiments our method on a real life application: the geomodeling of a petroleum reservoir, and on the generation of French alexandrines. The obtained results show the advantage and the efficiency of our approach.

1 Introduction

Multi-valued decision diagrams (MDDs) are a compressing data structure defined over a set of variables and used to store a set of tuples of values. They are implemented in almost all constraint programming solvers and have been increasingly used to build models [24, 21, 1, 9, 10, 3, 8, 5]. 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. They have a high compression efficiency. For instance, an MDD having 14,000 nodes and 600,000 arcs and representing 10^{90} tuples has been used to solve a music synchronization problem [24].

For solving some automatic generation problems, sampling from a knowledge data set is used to generate new data. Often, some additional control constraints must be satisfied. One approach is to generate a vast amount of sequences for little cost, and keep the satisfactory ones. However, this does not work well when constraints are complex and difficult to satisfy. Thus, some works have investigated to integrate the control constraints into the stochastic process.

For instance, in text generation, a Markov chain, which is a random process with a probability depending only on the last state (or a fixed number of them), is defined from a corpus [11, 16, 18]. In this case, a state can represent a word, and such a process will generate sequences of words, or phrases. It can be modeled as a directed graph, encoding the dependency between the previous state and the next state. Then, a random walk, i.e. a walk in this graph where the probability for choosing each successor has been given by the Markov model, will correspond to a new phrase. Such a walk corresponds to a sampling of the solution set while respecting the probabilities given by the Markov chain. This process generates sequences imitating the statistical properties of the corpus. Then, the goal is to be able to incorporate some side constraints defining the type of phrases we would like to obtain. For example, we may want to only produce sequences of words that contain no subsequence belonging to the corpus or longer than a given threshold, in order to limit plagiarism [16].

Such Markov models have long been used to generate music in the style of a composer [7, 13, 16]. The techniques of Markov constraints have been introduced to deal precisely with the issue of generating sequences from a Markov model estimated from a corpus, that also satisfy non Markovian, user defined properties [14, 15, 2, 25].

Hence, there is a real need for being able to sample some solutions while satisfying some other constraints.

The idea of this paper is to represent the corpus dependencies and the additional constraints by an MDD and develop sampling algorithms dealing with the solution set represented by this MDD.

Recently Papadopoulos *et al*. have designed a new algorithm which can be applied to a `regular` constraint [17]. However, the paper is complex because it is a direct adaption of the powerful and general belief propagation algorithm and requires the definition of a regular constraint. In this paper, we propose a conceptually simpler method defined on a more general data structure (the MDD), which may represent any regular constraint, but also different constraints. In addition, we show how to apply it for any kind of samplings and not only on Markov samplings. Thus, instead of developing ad-hoc algorithms or forcing the use of regular constraints, we propose a more general approach that could be used for a large range of problems provided that we have enough memory for representing the MDD.

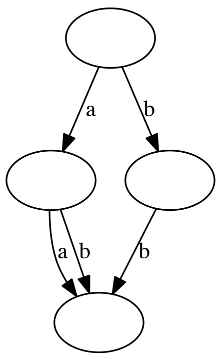

Fig. 1. A simple MDD.

However, combining samplings and MDDs is not an easy task. Consider, for instance, that we have a very simple MDD (Fig. 1) involving only two variables x_1 and x_2 whose values are a and b and that it represents the three solutions $S = \{((x_1, a), (x_2, a)), ((x_1, a), (x_2, b)), ((x_1, b), (x_2, b))\}$. Assume that we want to sample uniformly the solution set. In other words, we want to randomly select one solution with an equal probability for each solution. This can easily be done by randomly selecting a solution in S. Since there are 3 solutions, any solution has a probability of $1/3$ to be selected. The issue with MDDs is that they compress the solution set, so picking a solution with a uniform probability is not straightforward. For instance, if we randomly select the first value of the first variable and if we randomly select the value of the second variable then the selection is not uniform, because we are going to select more often the solution $((x_1, b), (x_2, b))$ than the others. This problem can be solved by computing the local probabilities of selecting a value according to the probabilities of the solutions containing that value.

Furthermore, we study the case where the probabilities of values are not the same and we consider Markov sampling, that is sampling where instead of considering the probability of selecting one value, we consider the probability of selecting a sequence of values.

In addition, it is sometimes interesting to define some constraints on the sampling. For instance, the problem of generating the sequences with the maximum probability in the Markov chain estimated from the corpus satisfying other constraints has been studied by Pachet and Roy [14]. Hence we propose some constraints for imposing that the probabilities of the solutions belong to a given range of probabilities.

This paper is mainly a paper about modeling and the advantage of having general methods for dealing with different kinds of problems occurring in Artificial Intelligence. As an example of this advantage, we apply our method to the transformation of classical texts written in French into alexandrine texts. This means that we try to express the same idea as the original text with the same style but by using only sentences having twelve syllables. The generation of the text in the same style as an author uses a Markov chain that is extracted from the corpus. An MDD is defined from the corpus and ensures that each sentence will have exactly twelve syllables. Then, probabilities implementing the Markov chain are associated with arcs of the MDD, and a random walk procedure is used for sampling the solutions. Thus, the model of this problem is conceptually simple and easy to implement. In addition, thanks to the existence of efficient propagators for MDDs and the algorithms we propose for computing local probabilities, it gives good results in practice.

We also test our approach on a real world application mainly involving convolutions which are expressed by knapsack constraints (i.e. $\sum \alpha_i x_i$) in which the probability of a value to be taken by a variable is defined by a probability mass function. In addition outliers are not allowed. We show how solutions can be efficiently sampled.

Note that the problem we consider is different from the work of Morin and Quimper on the Markov transition constraint which proposes to compute the distribution of the states of a Markov chain [12].

The paper is organized as follows. First we recall some definitions about probability distribution, Markov chain and MDDs and their use in constraint programming. Then, we propose some algorithms for sampling the solution set of an MDD while respecting the probabilities given by a distribution, that can be a probability mass function or a Markov chain. Next, we introduce two constraints ensuring that any solution of an MDD associated with some probability distribution belongs to a given probability interval. Afterwards, we present some experiments on the geomodelling of a petroleum reservoir and on the generation of French alexandrines based on the famous La Fontaine's fables. Finally we conclude.

2 Preliminaries

2.1 Probability distribution

We consider that the probability distribution is given by a probability mass function (PMF), which is a probability density function for a discrete random variable. The PMF gives for each value v, the probability $P(v)$ that v is taken:

Given a discrete random variable Y taking values in $Y = \{v_1, ...v_m\}$ its probability mass function P: $Y \rightarrow [0, 1]$ is defined as $P(v_i) = Pr[Y = v_i]$ and satisfies the following condition: $P(v_i) \geq 0$ and $\sum_{i=1}^{m} P(v_i) = 1$.

Property 1 *Let f_P be a PMF and consider $\{x_i\}$ a set of n discrete integer variables independent from a probabilistic point of view and associated with f_P that specifies probabilities for their values. Then, the probability of an assignment of all the variables (i.e. a tuple) is equal to the product of the probabilities of the assigned values. That is $\forall i = 1..n$, $\forall a_i \in D(x_i)$ $P(a_1, a_2, ..., a_n) = P(a_1)P(a_2)...P(a_n)$.*

2.2 Markov chain

A Markov chain[1] is a stochastic process, where the probability for state X_i, a random variable, depends only on the last state X_{i-1}. A Markov chain produces sequence $X_1, ..., X_n$ with a probability $P(X_1)P(X_2|X_1)...P(X_n|X_{n-1})$.

Property 2 *Let P_M be a Markov chain and consider a set of n discrete integer variables associated with P_M that specifies probabilities for their values. Then, $\forall i = 1..n$, $\forall a_i \in D(x_i)$ $P(a_1, a_2, ..., a_n) = P(a_1)P(a_2|a_1)...P(a_n|a_{n-1})$.*

Several methods can be used to estimate the Markov chain from a corpus, like the maximum likehood estimation [11]. This paper is independent from such methods and considers that the Markov chain is given.

Sampling a Markov chain can be simply and efficiently done by a random walk (i.e. a path consisting of a succession of random steps) driven by the distribution of the Markov chain. If we need to build a finite sequence of length k, then we perform a random walk of k iterations using the given distribution.

[1] Order k Markov chains have a longer memory: the Markov property states that $P(X_i|X_1, ..., X_{i-1}) = P(X_i|X_{i-k}, ..., X_{i-1})$. They are equivalent to order 1 Markov chains on an alphabet composed of k-grams, and therefore we assume only order 1 Markov chains.[17]

\	a	b
a	0.9	0.1
b	0.1	0.9

Tuple	Probability
aa	0.54
ab	0.06
ba	0.04
bb	0.36

Fig. 2. Markov chain for two variables. The starting probabilities are 0.6 for a and 0.4 for b.

Example. Consider M, the Markov chain in Fig. 2 and an initial probability of 0.6 for a and 0.4 for b. If we apply M on two variables x_1 and x_2, then the probability of the tuple (a, a) is $P(x_1, a)P((x_2, a)|(x_1, a)) = 0.6 \times 0.9 = 0.54$. The probabilities of the four possible tuples are given in Fig. 2. The sum of the probabilities is equal to 1.

2.3 Multi-valued decision diagram (MDD)

An MDD is a data-structure representing discrete functions. It is a multiple-valued extension of BDDs [4]. An MDD, as used in CP [5, 20, 1, 9, 10, 3, 8, 24], is a rooted directed acyclic graph (DAG) used to represent some multi-valued function $f : \{0...d - 1\}^n \rightarrow \{true, false\}$. Given the n input variables, the DAG representation is designed to contain $n+1$ 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. Each arc is labeled by its corresponding integer. The arc (u, v, a) is from node u to node v and labeled by a. All outgoing arcs of the layer n reach tt, the true terminal node (the false terminal node is typically omitted). There is an equivalence between $f(a_1, ..., a_n) = true$ and the existence of a path from the root node to the true terminal node whose arcs are labeled $a_1, ..., a_n$. The number of nodes of an MDD is denoted by V, the number of edges by E and d is the largest domain size of the input variables.

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. MDD(C) is defined on $X(C)$, such that layer i corresponds to the variable x_i and the labels of arcs of the layer i correspond to values of x_i, 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). 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. The value $a \in D(x_i)$ is consistent with MDD(C) iff there is a valid path in MDD(C) from the root node to tt 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 $X(C)$. It 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.

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-uplet $e = (u, v, a, c)$, where u is the head, v the tail, 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.

Cost-MDD of a constraint [8, 6]. 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 $f_C(x_i, a)$.

3 Sampling and MDD

We aim at sampling the solution set of an MDD while respecting the probabilities given by a distribution, that can be a PMF or a Markov chain.

Let M be an MDD whose n variables are associated with a distribution that specifies the probabilities of their values. For sampling the solutions of M, we propose to associate with each arc a probability, such that a simple random walk from the root node to tt according to these probabilities will sample the solution set of M while respecting the probabilities of the distribution of M.

First, we consider that the distribution of M is given by a PMF and that the variables of M are independent from a statistical point of view. Then, we will consider that we have a Markov chain for determining the probability of a value to be selected.

3.1 PMF and Independent variables

If the distribution associated with M is defined by a PMF f_P and if the variables of M are independent from a statistical point of view, then we propose to associate with each arc e a probability $P(e)$. From Property 1 we know that the probability of a solution $(a_1, ..., a_n)$ must be equal to $\Pi_{i=1}^{n} P(a_i)$.

We could be tempted to define $P(e)$ as the value of $f_P(label(e))$ where $label(e)$ is the label (i.e. value) associated with e. However, this is not exact because the MDD usually does not contain all possible combinations of values as solutions. For instance, consider the example of Fig. 1 with a uniform distribution. If all probabilities are equivalent then each solution must be able to be selected with the same probability, which is $1/3$ since there are three solutions (a, a), (a, b) and (b, b). Now, if we do a random walk considering that the probability of each arc is $1/2$ then we will choose with a probability $1/2$ the solution (b, b) which is incorrect. The problem stems from the fact that the probabilities of the higher layers are not determined according to the probabilities of solutions that they can reach while it should be the case. The choice (x_1, a) allows to reach 2 solutions and (x_1, b) one. So, with a uniform distribution the probability of choosing a for x_1 should be $2/3$ while that of choosing b should be $1/3$.

Definition 1 *The partial solutions that can be reached from a node n in an MDD are defined by the paths from n to tt.*

In order to compute the correct values, we compute for each node n the sum of the original probabilities of the partial solutions that we can reach from n. Then, we renormalize these values in order to have these sums equal to 1 for each node. For instance, for the node reached by traversing the first arc labeled by a in Fig. 1, the sum of the original probabilities is $1/2 + 1/2 = 1$, so the original probabilities are still valid. However, for the node reached by traversing the arc from the root and labeled by b, the sum of the original probabilities is $1/2$, so half of the combinations are lost. This probability is no longer valid and new values must be computed.

The sum of the original probabilities of the partial solutions that can be reached from a node is defined as follows:

Property 3 *Let M be an MDD defined on X and f_P a PMF associated with M. Let n be any of node of the MDD and A be any partial instantiation of X reaching node n. The sum of the original probabilities of the partial solutions that can be reached from n is $v(n) = \sum_{s \in S(n)} P(s|A)$, where $S(n)$ is the set of partial solutions that we can reach from n and $P(s|A)$ is the probability of s under condition A. The probability of any arc $e = (n', n, a)$ is defined by $P(e) = f_P(a) \times v(n)$.*

proof: By induction from tt. Assume this is true at layer $i+1$. Let n' be a node of layer i, n a node in layer $i+1$ and $e = (n', n, a)$ an arc. We have $P(e) = f_P(a) \times v(n)$, that is $P(e) = f_P(a) \times \sum_{s \in S(n)} P(s|A)$, where A is any partial instantiation reaching node n. So for node n' we have:
$v(n') = \sum_{e \in \omega^+(n')} P(e)$, where $\omega^+(n')$ is the set of outgoing arcs of n'
$v(n') = \sum_{e \in \omega^+(n')} f_P(label(e)) \times \sum_{s \in S(n)} P(s|A)$. Note that A is any partial instantiation reaching node n, so it can go through e. So we have
$v(n') = \sum_{s \in S(n')} P(s|A')$ where A' is any partial instantiation reaching node n'. $\square$

The correct probabilities can be computed by a bottom-up algorithm followed by a top-down algorithm. First, we consider the second to last layer and we define the probability P of an arc labeled by a as $f_P(a)$. Then, we directly apply Property 3 from the bottom of the MDD to the top: once the layer $i+1$ is determined, we compute for each node n' of the layer i the value $v(n') = \sum_{e \in \omega^+(n')} P(e) = \sum_{e \in \omega^+(n')} f_P(label(e)) \times v(n)$. Once the bottom-up part is finished, we normalize the computed values P in order to have $v(n) = 1$ for each node n. We use a simple top-down procedure for computing these values. Fig. 3 details this process. The left graph simply contains the probability of the arc labels. The middle graph shows the bottom-up procedure. For instance, we can see that the right arc outgoing from the source has a probability equal to $1/2 \times 1/2 = 1/4$. Thus a normalization is needed for the root because the sum of the probabilities of the outgoing arcs is $1/2 + 1/4 = 3/4 < 1$. The right graph is obtained after normalization.

Note that the normalization consists of computing the probability according to the sum of the probabilities. If $P(e)$ is the current value for the arc $e = (u, v, a)$ and T is the sum of the probability of the outgoing arcs from u, then the probability of e becomes $P(e)/T$.

This step can be avoided in practice by computing such normalized values only when needed.

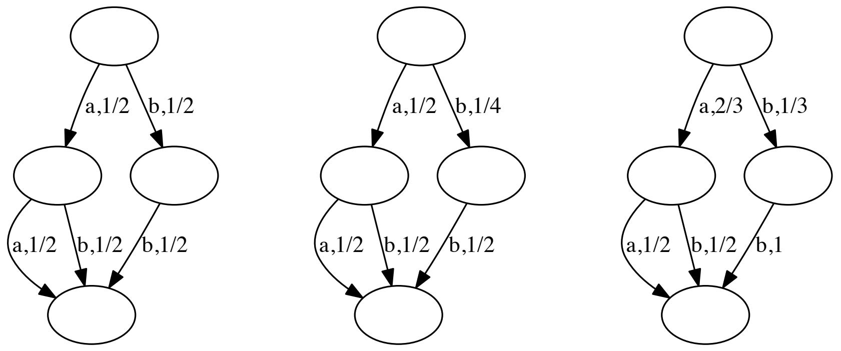

Fig. 3. Sampling from a simple MDD. The probability of a and b are 1/2.

Algorithm COMPUTEMDDPROBABILITIES can be described as follows:

1. Set $v(tt) = 1$; For each node $v \neq tt$, in a Breadth First Search (BFS) in bottom-up fashion:
   (a) Compute $v(n)$ the sum of the original probabilities of the outgoing arcs of n.
   (b) Define the probability of each incoming arc e of n labeled by a as $P(e) = f_P(a) \times v(n)$
2. For each node in a BFS top-down fashion, normalize the probabilities of the outgoing arcs.

During this algorithm, each sum is calculated once for each node during the bottom up processing, and the normalization is performed once for each arc. The final complexity is $O(|E| + |V|)$.

Fig. 4 gives an example of the running of this algorithm when the probabilities are not uniform.

3.2 Markov chain

As in the previous section, our goal is to associate each arc with a probability and then sample the solution set by running a simple random walk according to these probabilities. The method we obtain is equivalent to the one proposed by Papadopoulos *et al.* for the `regular` constraint [17]. However, their method is complex and the propagation of the regular constraint costs more memory than the one of an MDD [20]. We claim that our method is conceptually simpler.

It is more difficult to apply a Markov chain than a PMF because in a Markov chain the probability of selecting a value depends on the previous selected value, that is, probabilities must be defined in order to satisfy Property 2. More precisely, in an MDD, a node can have many incoming arcs, and these different incoming arcs can have different labels. Since the Markov probability depends on the previous value, the outgoing arcs of that node may have different probabilities depending on which was the incoming arc

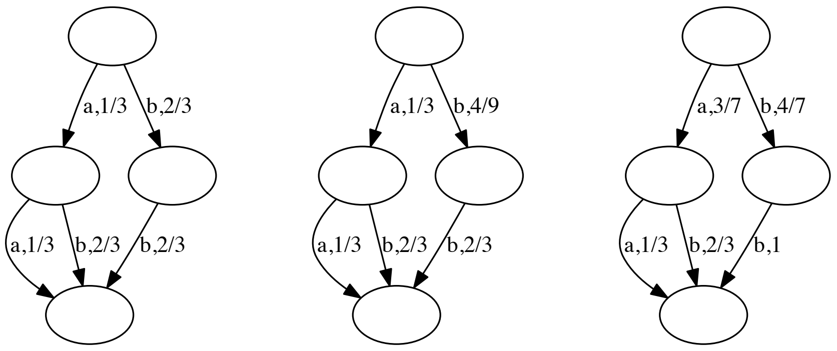

Fig. 4. Sampling from a simple MDD. The probability of a is 1/3 and it is 2/3 for b.

label. Thus, for an arc e, we need to have several probability values depending on the previous arc that has been used.

There are two possible ways to deal with a Markov chain. Either we transform the MDD by duplicating nodes in order to be able to apply an algorithm similar as COMPUTEMDDPROBABILITIES or we directly deal with the original MDD and we design a new algorithm.

Duplication of nodes We can note that the matrix of the Markov chain represents a compression of nodes. Thus, if we duplicate each node according to its incoming arcs then we obtain a new MDD for which the probabilities become independent. More precisely, for each node n we split the node n in as many nodes as there are different values incoming. This means that each node n has only incoming arcs having the same label, and so only one value a incoming. Thus, the probability of each outgoing arc of the duplicated nodes of n can be determined directly by the Markov matrix.

For instance, consider the probabilities of Fig 2 and that we have a node n with two incoming arcs: one labeled by a an the other labeled by b; and with two outgoing arcs: one labeled by a an the other labeled by b (Fig. 5). The node n is split into two nodes n_a and n_b. Node n_a has only incoming arcs labeled by a, and n_b has only incoming arcs labeled by b ((c) in Fig 5). In this case, we can define the probabilities as if we had independent variables. The probability of the arc (n_a, x, a), is defined by $P(a|a) = 0.9$, the probability of the arc (n_a, x, b) is $P(b|a) = 0.1$, the probability of the arc (n_b, x, a) is $P(a|b) = 0.1$, the probability of the arc (n_b, x, b) is $P(b|b) = 0.9$. Fig. 5 shows the duplication of a node. Note that when the node x will be split into two nodes x_a and x_b, then each of them will have two incoming arcs having the same label, a for x_a and b for x_b ((d) in Fig. 5).

Let $P_C(e)$ be the computed probability of any edge e computed by the duplication process. We can establish a Property similar as Property 3

\	a	b
a	0.9	0.1
b	0.1	0.9

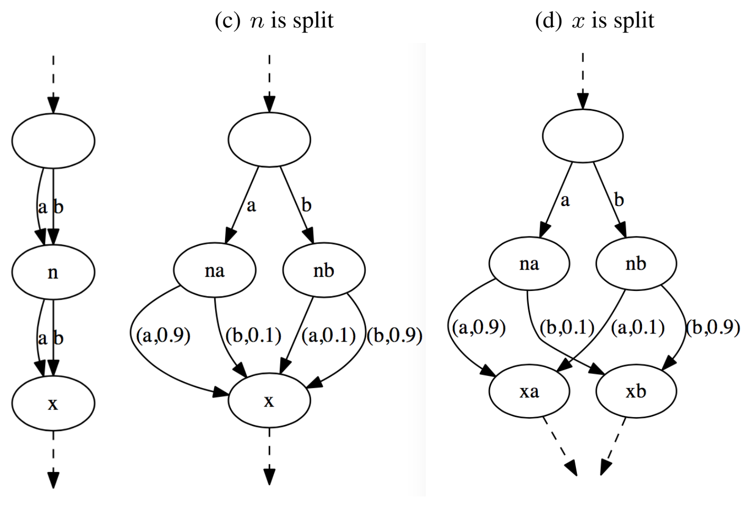

Fig. 5. Duplication of a node and computation of probabilities.

Property 4 *Let M be an MDD defined on X and P_C a probability associated with each arc. Let n be any of node of the MDD and A be any partial instantiation of X reaching node n. The sum of the original probabilities of the partial solutions that can be reached from n is $v(n) = \sum_{s \in S(n)} P(s|A)$, where $S(n)$ is the set of partial solutions that we can reach from n and $P(s|A)$ is the probability of s under condition A. The probability of any arc $e = (n', n, a)$ is defined by $P(e) = P_C(e) \times v(n)$.*

proof: Similar as for Property 3. □

From this property we can design an algorithm similar as COMPUTEMDDPROBABILITIES by using $P_C(e)$ instead of $f_P(label(e))$ for each arc e. The drawback of this method is that it can multiply the number of nodes by at most d, the greatest cardinality domain of variables and also increases the number of edges which slowdowns the propagators. The next section presents another method avoiding this duplication.

A new algorithm In order to deal with the fact that the probability of an outgoing arc depends on the label of the incoming arc without duplicating nodes, we associate each node with a probability matrix whose row depends on the incoming arc label. We denote these matrices by P_M^n for the node n. For efficiency, we only have one vector by incoming value instead of the full matrix, and each vector contains only the probability of the possible outgoing arcs labels. Then, the same reasoning as previously can be applied. We just need to adapt the previous algorithm by using matrices instead of duplicating nodes:

Algorithm COMPUTEMDDMARKOVPROBABILITIES can be described as follows:

1. For each node n, build the P_M^n matrix by copying the initial Markov probabilities.
2. For each node n, in BFS in bottom-up fashion:

(a) Build the vector $vv(n)$ whose size is equal to the number of different incoming labels[2]. Each cell contains the sum of the probabilities of the row of the corresponding label in the P_M^n matrix.
(b) Multiply each incoming arc probability by the cell of $vv(n)$ corresponding to its label.

3. For each node in a BFS top-bottom fashion, normalize the probability of the outgoing arcs.

Example. Consider the MDD of Fig. 6.a, if we reuse the Markov distribution of Fig. 2 and apply the step 1 of the method, we obtain the MDD in Fig. 6.b.

Now from the MDD in Fig. 6.b, we perform step 2, first (step 2.a) we process the sum of the outgoing probabilities for each node. For example for node 5 its probability is $0.1 + 0.9 = 1$ and for node 3 the sum is 0.9. For these two nodes the sum does not depend on the incoming arc label because there is only one. This is not the case for node 4 which has a sum of 0.1 for the incoming arc labeled by a and 0.9 for the incoming arc labeled by b. Now we apply step 2.b: we multiply the probability of the incoming arcs by the sum associated to their label in their destination node. Consider the arc from node 1 to node 3 and labeled by a, its probability was 0.9 and the sum of probabilities in its destination node is 0.9, then its new probability is 0.81. The arc from node 1 to node 4 is labeled by b; its probability was 0.1. For node 4, the sum is 0.9 for the incoming arc labeled by b, so the new probability of the $(1, 4, b)$ is $0.1 \times 0.9 = 0.09$. The MDD in Fig. 7.a is labeled with the resulting global probabilities.

Finally, from the MDD in Fig. 7.a, we normalize the outgoing arc probability of each node (step 3). For the root node 0, the outgoing probabilities sum is $0.54 + 0.364 = 0.904$. For its arc labeled by a and directed to node 1, the probability become $0.54/0.904 = 0.597$, this value has been rounded to 3 digits for readability. For its arc labeled by b and directed to node 2, the probability becomes $0.364/0.904 = 0.403$ (rounded). Thus, the outgoing sum of probabilities emanating from node 0 becomes $0.597 + 0.403 = 1$. The MDD from Figure 7.b shows the normalized probabilities.

Complexities. The complexities of COMPUTEMDDMARKOVPROBABILITIES algorithm are the following. The number of matrices is $|V|$, in the worst case the number of columns and rows is d, so the global memory complexity is $O(|V| \times d^2)$. The complexity of each of the operations of this method are all linear over the matrices, so the overall time complexity is $O(|V| \times d^2)$. Since the number of columns of the matrix of a node is equal to the number of outgoing arcs of this node, a more realistic complexity for space and time is $O(|V| + |E| \times d)$, knowing that in a MDD, $|E| \leq |V| \times d$. Note that, for a given layer, nodes can be processed in parallel.

3.3 Incremental modifications.

If some modifications occur in the MDD, then instead of reprocessing all the probabilities we can have an incremental approach. From Step 2 of algorithms COMPUTEMDDPROBABILITIES or COMPUTEMDDMARKOVPROBABILITIES, which performs a BFS in bottom-up, we perform the BFS only from the modified nodes since they are the only ones that can trigger modifications of the probabilities.

[2] $vv(n)$ represents a vector of $v(n)$.

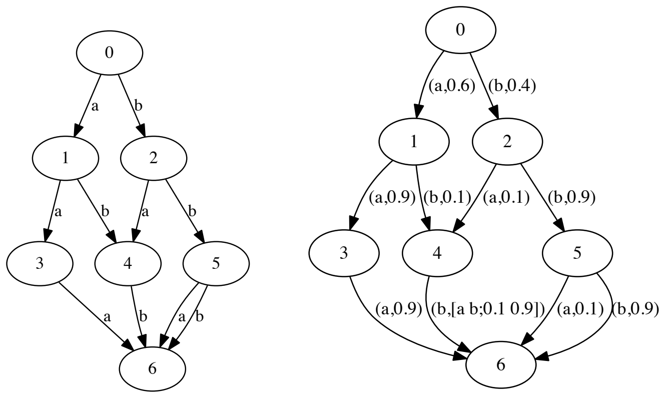

Fig. 6. (a) left: an MDD. (b) right: the MDD whose arcs have their probability set thanks to the Markov distribution.

The reset principle used in MDD4R [20] can also be applied in this case. In other words, when there is less remaining arcs than deleted arcs, it is worthwhile to recompute from scratch the values.

4 MDDs and Probabilities based constraints

For some reasons, like security or for avoiding outliers, some paths of MDDs can be unwanted, because they have only very little chance to be selected or because they contain almost only values having the strongest probability to be selected. In other words, we accept only paths whose probability is in a certain interval.

We define constraints for this purpose. One, named the `MDDProbability`, considered that the MDD is associated with a PMF and independent variables and the other, named `MDDMarkovProcess`, that the MDD is associated with a Markov chain.

Definition 2 *Given M an MDD defined on $X = \{x_1, x_2, ..., x_n\}$ that are independent from a probabilistic point of view and associated with f_P a probability mass function , P_{min} a minimum probability and P_{max} a maximum probability. The constraint `MDDProbability`$(X, f_P, M, P_{min}, P_{max})$ ensures that every allowed tuple $(a_1, a_2, ...a_n)$ is a solution of the MDD and satisfies $P_{min} \leq \Pi_{i=1}^{n} f_P(a_1) \leq P_{max}$.*

This constraint can be easily transformed into a cost-MDD constraint. The cost associated with an arc labeled by a is $\log(f_P(a))$, and the logarithms of P_{min} and P_{max} are considered for dealing with a sum instead of a product[3]. Thus, any cost-MDD propagator can be used [22].

[3] We can also directly deal with products if we modify the costMDD propagator accordingly.

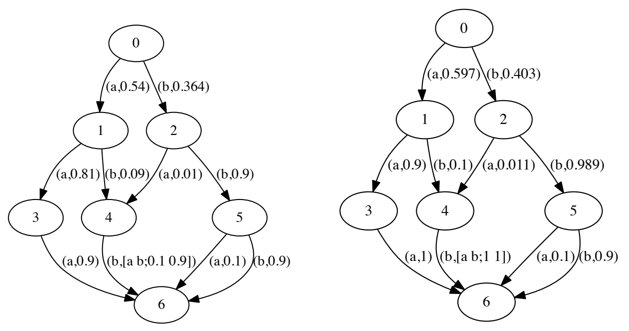

Fig. 7. (a) left: MDD from Fig. 6.b whose arcs probability has been multiplied by the sum of the probabilities of the outgoing arcs from their destination node. (b) right: the MDD with renormalized probabilities.

Definition 3 *Given M an MDD defined on $X = \{x_1, x_2, ..., x_n\}$ and associated with P a Markov chain, P_{min} a minimum probability and P_{max} a maximum probability. The constraint* `MDDMarkovProcess`$(X, P, M, P_{min}, P_{max})$ *ensures that every allowed tuple $(a_1, a_2, ...a_n)$ is a solution of the MDD and satisfies*
$P_{min} \leq P(a_1)P(a_2|a_1)...P(a_n|a_{n-1}) \leq P_{max}$.

As we have seen, with a Markov chain, the probability for selecting an arc depends on the previous selected arc. Thus, each arc of the MDD is associated with several probabilities. So we cannot directly use a cost-MDD propagator as for the `MDDProbability` constraint. However, if we accept to duplicate the nodes as proposed in the previous section then we can immediately transforms the constraint into a simple cost-MDD constraint by considering logarithms of probabilities and any cost-MDD propagator can be used. Since the number of time a node can be duplicated is bounded by d, the overall complexity of this transformation is $O(d \times (|V| + |E|))$.

5 Evaluation

The experiments were run on a macbook pro (2013) Intel core i7 2.3GHz with 8 GB of memory. The constraint solver used is or-tools. MDD4R [20] is used as MDD propagator and cost-MDD4R as cost-MDD propagator [23].

5.1 PMF constraint and sampling

The data come from a real life application: the geomodeling of a petroleum reservoir [19]. The problem is quite complex and we consider here only a subpart. Given a seismic image we want to find the velocities. Velocities values are represented by a probability mass function (PMF) on the model space. Velocities are discrete values of variables. For each cell c_{ij} of the reservoir, the seismic image gives a value s_{ij} from which we define a sum constraint $C_{ij} : \sum_{k=1}^{22} \alpha_k log(x_{i-11+k-1}j) = s_{ij} \pm \epsilon$, where α_k are defined from the given seismic wavelet. Locally, that is, for each sum, we have to avoid outliers w.r.t. the PMF for the velocities. The problem is huge (millions of variables) so we consider here only a very small part.

We recall that the MDD of the constraint $\sum_{x_i \in X} f(x_i) \in I$, with $I = [a, b]$ is denoted by MDD$(\Sigma_{f,I}(X))$ and defined as follows. For the layer i, there are as many nodes as there are values of $\sum_{k=1}^{i} f(x_k)$. Each node is associated with such a value. A node n_p at layer i associated with value v_p is linked to a node n_q at layer $i+1$ associated with value v_q if and only if $v_q = v_p + f(a_i)$ with $a_i \in D(x_i)$. Then, only values v of the layer $|X|$ with $a \leq v \leq b$ are linked to tt. The reduction operation is applied after the definition and delete invalid nodes [21]. The construction can be accelerated by removing states that are greater than b or that will not permit to reach a.

Each constraint C_{ij} is represented by MDD$(\Sigma_{f,I}(X))$ where $f(x_i) = \alpha_i x_i$ and I is the tight interval representing $[s_{ij} - \epsilon, s_{ij} + \epsilon]$. Outliers are avoided thanks to an `MDDProbability` constraint defined from the PMF for the velocities. P_{min} is defined by selecting only values having the 10% smaller probabilities, P_{max} is defined by selecting only values having the 10% greater probabilities. This constraint is represented by a cost-MDD constraint, as explained in the Section 4. Then, we intersect it with MDD$(\Sigma_{f,I}(X))$.

We consider 20 definitions of C_{ij}. We repeat the experiments 20 times and take the mean of the results.

For each constraint C_{ij}, the resulting MDD has in average 116,848 nodes and 1,239,220 edges. More than 320s are needed to compute it. Only 8 ms are required by COMPUTEMDDPROBABILITIES algorithm in average. When a modification occurs the time to recompute the values are between a negligible value when the modifications are close to the root of the MDD and 8 ms when another part is modified.

For sampling 100,000 solutions we need 169 ms with the rand() function and 207 ms with the Mersenne-Twister random engine in conjunction with the uniform generator of the C++ standard library. Note that the time spends within the rand() function is 15 ms, whereas it is 82 ms with the second function. Therefore, the sampling procedures require less than 3 times the time spent in the random function.

5.2 Markov chain and sampling

We evaluate our method for generating French alexandrines. That is, sentences containing exactly twelve syllables. The goal is to transform an existing text into a text having the same meaning but using only alexandrines. From the corpus we define a Markov chain and an MDD representing the sentences having the right number of syllables. The sampling procedure we define generates solutions of the MDD associated with the Markov chain, that is, sentences hopefully resembling those of the corpus and having exactly 12 syllables. This model is simple and easy to implement. Note that we are not able to model this problem with any other technique, even the one proposed by Papadopoulos *et al*, because we need to deal only with sentences having 12 syllables and we do not know how to integrate this constraint into their model.

First, we use a corpus defined by one of the famous La Fontaine's fables. Here is the result we obtain for the fable: La grenouille qui veut se faire aussi grosse que le boeuf

(The Frog and the Ox). We have underlined the syllables that must be pronounced when it is unclear:

La grenouille veut se faire aussi grosse que le bœuf

Grands seigneurs Tout bourgeois veut bâtir comme un Bœuf
Plus sages Tout marquis veut bâtir comme un œuf
Pour égaler l'animal en tout M'y voila
Voici donc Point du tout comme les grands seigneurs
Chétive Pécore S'enfla si bien qu'elle creva
Seigneurs Tout petit prince a des ambassadeurs

The generation of the MDD with the correct probabilities, that is just before the random walk, can be performed in negligible computational time.

We also considered a larger corpus: "A la recherche du temps perdu" of Proust, which contains more than 10,000 words. In this case, the results are less pertinent and some more work must be done about the meaning of the sentences. However, the method is efficient in term of computing performance because only 2 seconds are needed to create the MDD with the correct probabilities.

6 Conclusion

We have presented two methods for sampling MDDs, one using a probability mass function and another one using a Markov chain. These methods require the definition of probabilities for each arc and we have given algorithms for performing this task. We have also proposed propagators for constraining these probabilities. Thanks to these algorithms and MDD propagators we can easily model and implement complex problems of automatic music or text generations having good performances in practice. We have experimented our method on a real life application: the geomodeling of a petroleum reservoir and on the problem of the transformation of French texts into alexandrines. We have shown how it is easy to define the model and to generate solutions.

7 Acknowledgments

This research is conducted within the Flow Machines project which received funding from the European Research Council under the European Unions Seventh Framework Programme (FP/2007-2013) / ERC Grant Agreement n. 291156. We would like to thank F. Pachet and P. Roy, who gave us the idea of this article.

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. Gabriele Barbieri, François Pachet, Pierre Roy, and Mirko Degli Esposti. Markov constraints for generating lyrics with style. In *ECAI 2012 - 20th European Conference on Artificial Intelligence.*, pages 115–120, 2012.
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. *IEEE Trans. Computers*, 35(8):677–691, 1986.
5. Kenil C. K. Cheng and Roland H. C. Yap. An mdd-based generalized arc consistency algorithm for positive and negative table constraints and some global constraints. *Constraints*, 15(2):265–304, 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. Brooks F., Hopkings A., Neumann P., and Wright W. An experiment in musical composition. 3(6):175–182, 1957.
8. Graeme Gange, Peter J Stuckey, and Pascal Van Hentenryck. Explaining propagators for edge-valued decision diagrams. In *CP*, 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. Dan Jurafsky and James H Martin. *Speech and language processing*. Pearson, 2014.
12. Michael Morin and Claude-Guy Quimper. The markov transition constraint. In *International Conference on AI and OR Techniques in Constriant Programming for Combinatorial Optimization Problems*, pages 405–421. Springer, 2014.
13. G Nierhaus. *Algorithmic composition: paradigms of automated music generation*. Springer, 2009.
14. François Pachet and Pierre Roy. Markov constraints: steerable generation of markov sequences. *Constraints*, 16(2):148–172, 2011.
15. François Pachet, Pierre Roy, and Gabriele Barbieri. Finite-length markov processes with constraints. In *IJCAI 2011*, pages 635–642, 2011.
16. 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.
17. Alexandre Papadopoulos, François Pachet, Pierre Roy, and Jason Sakellariou. Exact sampling for regular and markov constraints with belief propagation. In *International Conference on Principles and Practice of Constraint Programming*, pages 341–350. Springer, 2015.
18. Alexandre Papadopoulos, Pierre Roy, Jean-Charles Régin, and François Pachet. Generating all possible palindromes from ngram corpora. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*, pages 2489–2495, 2015.
19. Wayne D. Pennington. Reservoir geophysics. 66(1), 2001.
20. G. Perez and J-C. 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.
21. G. Perez and J-C. Régin. Efficient operations on mdds for building constraint programming models. In *International Joint Conference on Artificial Intelligence, IJCAI-15*, pages 374–380, Argentina, 2015.

22. G. Perez and J-C. Régin. Soft and cost mdd propagators. In *The Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17)*, 2017.
23. G. Perez and J-C. Régin. Soft and cost mdd propagators. In *Proc. AAAI'17*, 2017.
24. P. Roy, G. Perez, JC. Régin, A. Papadopoulos, F. Pachet, and M. Marchini. Enforcing structure on temporal sequences: The allen constraint. In *CP 2016*, pages 786–801, 2016.
25. Pierre Roy and François Pachet. Enforcing meter in finite-length markov sequences. In *AAAI 2013*, 2013.