Enforcing Meter in Finite-Length Markov Sequences

Pierre Roy and **François Pachet**
Sony CSL
6, rue Amyot
75 005, Paris, France

Abstract

Markov processes are increasingly used to generate finite-length sequences that imitate a given style. However, Markov processes are notoriously difficult to control. Recently, Markov constraints have been introduced to give users some control on generated sequences. Markov constraints reformulate finite-length Markov sequence generation in the framework of constraint satisfaction (CSP). However, in practice, this approach is limited to local constraints and its performance is low for global constraints, such as cardinality or arithmetic constraints. This limitation prevents generated sequences to satisfy structural properties which are independent of the style, but inherent to the domain, such as meter. In this article, we introduce **`meter`**, a constraint that ensures a sequence is 1) Markovian with regards to a given corpus and 2) follows metrical rules expressed as cumulative cost functions. Additionally, **`meter`** can simultaneously enforce cardinality constraints. We propose a domain consistency algorithm whose complexity is pseudo-polynomial. This result is obtained thanks to a theorem on the growth of sumsets by Khovanskii. We illustrate our constraint on meter-constrained music generation problems that were so far not solvable by any other technique.

1 Modeling Style for Content Generation

Statistical style imitation techniques are increasingly used for content generation applications. From a corpus of finite-length sequences considered as representative of the style of an author, a statistical model of the style is built. Then new sequences can be generated from the model that "look" like or "sound" like the originals. In the context of Markov processes, models exploit the Markov hypothesis, which states that the future state of a sequence depends only on the last state, i.e.:

$$p(s_i|s_1, \ldots, s_{i-1}) = p(s_i|s_{i-1}).$$

The Markovian aspects of musical sequences have long been acknowledged, see e.g., (Brooks et al. 1992). Many attempts to model musical style have therefore exploited Markov chains in various ways (Nierhaus 2009), notably sequence generation. The same is true, to some extent, for text, and toy text generators can easily be implemented to illustrate this point.

However, putting these ideas in practice raises difficult control problems: users generally want to enforce specific, domain-dependent properties on the sequences to generate. Unfortunately, Markov models, as most statistical models, do not offer natural handles to enforce such properties. The reason is that semantically interesting properties often establish long-range correlations between non contiguous items in the sequence, which is incompatible with the Markov hypothesis of limited dependency.

Markov Constraints

A general solution to this problem, called *Markov constraints*, was introduced in (Pachet and Roy 2011). It consists in reformulating Markov generation in the context of constraint satisfaction. The sequence to generate is viewed as a sequence of constrained variables, and Markovianity is represented as a constraint holding on consecutive pairs of contiguous variables (for order 1). Control constraints can then be expressed as arbitrary additional constraints, including global constraints. However, such a reformulation is costly as there is no boundary, in principle, on the complexity of the resolution. For interactive content generation applications, better performance is needed.

It was shown recently that for unary and binary constraints, efficient solutions can be found by transforming the initial Markov model (Pachet, Roy, and Barbieri 2011). This result enables the handling of useful constraints. For instance, syntax and rhymes can be represented as unary constraints, which opens the door to fascinating text generation applications (Barbieri et al. 2012).

Markov and Meter

However, many control properties which are natural to express on such sequences require more complex, global constraints. In particular, practical problems in music or text generation often involve constraining *sums of quantities*, a phenomenon known as "meter". Meter consists in recurring temporal patterns associated to the sequence, but which concern an unknown number of items.

For instance, meter in poetry constrains the total duration of an unknown number of words: an Alexandrine verse contains exactly 12 syllables, but the number of words may vary. Similarly in music, it is natural to constrain the total duration of subgroups of continuous notes in the sequence (called

bars) to some predetermined value (say, 4 beats) but not its actual number of notes. For instance, in Figure 3, all the bars have the same duration but different number of notes. Figure 5 shows a melody generated by a simple Markov process trained in the melody of Figure 3 with the same number of notes. It can be seen that the total duration is not the same, and that the meter is not enforced (some notes span over two bars).

Cardinality is yet another example of useful control constraint to enforce on such sequences. In the context of tonal music generation, it may be interesting to control the number of dissonant notes (e.g., an $F^{\#}$ in a C major tonality) by cardinality constraints. In text generation, a phrase that "talks about a subject" can be seen as enforcing a cardinality constraint such as "there is at least one word of a given semantic category".

Sum equal and cardinality have been addressed in the literature of global constraints (e.g., (Régin 1996) and (Zhang and Yap 2000)). However, they do not behave well when they are added to a CSP that also includes Markov constraints. No guarantee is given that solutions (or that the absence of solution) can be found in reasonable time. Additionally, they do not enable the specification of properties on a variable number of variables, a fundamental aspect of meter.

It is therefore interesting to look for a global constraint that enforces *simultaneously* the Markovianity of a sequence as well as meter and cardinality constraints.

This paper proposes the **meter** constraint, that ensures that a sequence is Markovian and also satisfies general meter properties. In order to maximize the scope of our constraint, meter properties are defined by a running predicate, π, which applies on all the partial sequences, i.e., subsequences from the first item to any other item. This predicate enables the definition of meter as well as cardinality, to some extent. We propose an algorithm that enforces arc-consistency for **meter**. Its complexity is shown to be in pseudo-polynomial time, thanks to a theorem on the growth of sumsets by Khovanskii. We illustrate the constraint with examples in melody generation.

Related Works

Several global constraints have been proposed that address similar problems, but they are not applicable to meter. (Petit, Beldiceanu, and Lorca 2011) propose a GAC algorithm for a class of counting constraints with polynomial complexity, **seqbin**, which counts the number of consecutive times a given binary constraint is satisfied in a sequence. It is not possible to express meter with **seqbin**, as meter involves summing up quantities on all subsequences, from the first item to any other item in the sequence. Additionally, our constraint can enforce a global **sum-equals** on the sequence, which is impossible to achieve with **seqbin**.

Our constraint deals with cumulative sums of costs along the sequence. The **cumulative** constraint (Aggoun and Beldiceanu 1993) and its many variations deal with task assignment problems having an *upper cost limit*, so its scope is different from ours, because meter is made up of recurring temporal patterns.

The constraint **regular** (Pesant 2004) and its cost-variant, **cost-regular** (Demassey, Pesant, and Rousseau 2006), were proposed for the generation of sequences that belong to a given regular language. Although these constraints bear similarities with **meter**, they are not naturally adapted to our problem. Handling meter with a **regular** constraint requires to define a specific regular automata whose size may be very large. This point is further illustrated in Section 2. The **cost-regular** constraint is not suited either to our problem because it does not guarantee domain consistency for the sequence variables. Domain consistency is an essential aspect of our proposition as it ensures a *backtrack-free* generation of sequences constrained by a single **meter** constraint, which is crucial for real-time interactive applications. Besides, as explained in Section 9, a backtrack-free generation is necessary to represent a posteriori the probabilities of the original Markov model.

2 Running Example

As an illustration, consider the corpus S consisting of the two sequences $s_1 = (1, 2, 3, 4, 3, 2, 1)$ and $s_2 = (1, 2, 4, 2, 1)$. A Markov model M_S is estimated from S, whose transition probabilities are:

$$\begin{pmatrix} 0 & 1 & 0 & 0 \\ 1/2 & 0 & 1/4 & 1/4 \\ 0 & 1/2 & 0 & 1/2 \\ 0 & 1/2 & 1/2 & 0 \end{pmatrix}$$

We consider the problem (P) of generating from M_S finite sequences $s = (x_1, \ldots, x_n)$ such that:

- (P1) : $\exists i_1 < n$ such that $\sum_{i=1}^{i_1} x_i = 6$;
- (P2) : $\exists i_2 < n$ such that $\sum_{i=1}^{i_2} x_i = 12$;
- (P3) : $\sum_{i=1}^{n} x_i = 18$.

For instance, $\sigma_1 = (1, 2, 3, 4, 2, 3, 2, 1)$ is such a sequence as: 1) all the transitions exist in the corpus sequences, and 2) σ_1 is the concatenation of the three subsequences $(1, 2, 3)$, $(4, 2)$, and $3, 2, 1$ whose sum is 6. On the contrary, $\sigma_2 = (1, 2, 4, 3, 2, 3, 2, 1)$ is not a correct sequence as it can not be decomposed into subsequences of cost 6. Sequence $\sigma_3 = (1, 2, 3, 4, 3, 4, 2)$ is not correct either as its sum is not 18, but 19.

The sequences that satisfy properties (P3) have at most 18 elements. Therefore, the problem can be stated as a global constraint $\mathcal{C}_S$ on 18 variables $V_1, \ldots, V_{18}$, each with domain $D = \{0, 1, \ldots, 4\}$.

$\mathcal{C}_S$ states that, for every valuation $(x_1, \ldots, x_{18}), x_i \in D$ of its variables:

- $\forall i, M_S(x_i, x_{i+1}) > 0$ or $x_{i+1} = 0$;
- $\forall i, x_i = 0 \implies x_{i+1} = 0$;
- $\forall k \leq n, \pi(\sum_{i=1}^{k-1} x_i, x_k, k)$, where π is the predicate defined by:

$$\pi(c, x, k) = (c + x = 18 \vee k < 18) \wedge \left(\left\lfloor \frac{c}{6} \right\rfloor = \left\lfloor \frac{c+x}{6} \right\rfloor \vee c \equiv 0 \pmod 6 \right)$$

The running sum, $\sum_{i=1}^{j} x_i$, will go from 0 up to 18. We need to trace when the partial sum passes multiples of 6. That happens for those k at which we have

$$\left\lfloor \frac{\sum_{i=1}^{k-1} x_i}{6} \right\rfloor < \left\lfloor \frac{\sum_{i=1}^{k} x_i}{6} \right\rfloor .$$

For these k, it must be enforced that the partial sum on the left, $\sum_{i=1}^{k-1} x_i$, is indeed a multiple of 6, which is done via the condition $c \equiv 0 \pmod 6$ in the definition of π.

The sequences that satisfy $\mathcal{C}_S$ are the sequences that satisfy (P1)-(P3) completed with 0 elements. Note that σ_2 violates $\mathcal{C}_S$ because π does not hold at position $k = 3$: $((1+2)\ \mathbf{div}\ 6) = 0 \neq ((1+2+4)\ \mathbf{div}\ 6) = 1$ but $(1+2) \not\equiv 0\ (6)$, i.e., there is no partial sequence of σ_2 whose sum is 6.

Note that this example could be formulated with a **`regular`** constraint. To capture the information in the definition of π, the corresponding deterministic finite automaton (DFA) would have one state for each (*value*, *partial sum modulo 6*) pair. Therefore, the automaton graph would consist of 6 states for each value.

In general, for a given predicate π, the relation between π and the corresponding DFA for **`regular`** is not known. In particular, there is no guarantee on the size of the DFA and therefore on the complexity of the **`regular`** constraint.

3 The `meter` Constraint

The constraint $\mathcal{C}_S$ defined in the previous section is an example of **`meter`** constraint. The transition probabilities and the transitions of order higher than 1 are not represented by Markov Meter constraints. We will show in Section 9 that transition probabilities and higher-order transitions can be taken into account during the sequence generation.

The general formulation of **`meter`** constraints is the following. Given a set of values $X = \{x_1, \ldots, x_m\}$, a Markov model M on X, a cost function $c\colon X \to \mathbb{N}$, a predicate π on $\mathbb{N} \times X \times \mathbb{N}$, and a maximum size n for the sequences to generate, we define the **`meter`** constraint $\mathcal{C}_{X,M,c,\pi,n}$ as follows. ($\mathcal{C}_{X,M,c,\pi,n}$ is abbreviated by $\mathcal{C}$ in the following.)

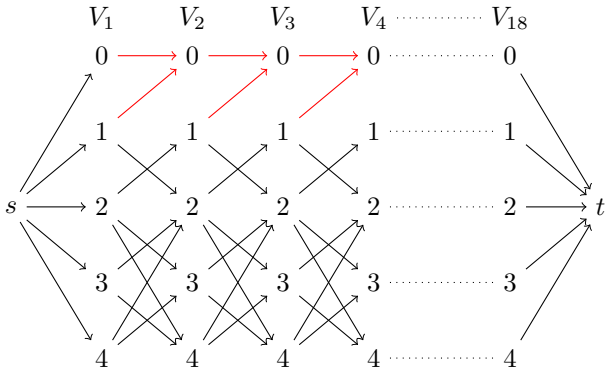
Let $D = X \cup \{x_0\}$, where x_0 is a dummy value that is added to the domains of the variables to allow the generation of sequences of variable length. We extend c to $X \cup \{x_0\}$ by stating that $c(x_0) = 0$. We define variables $V_1, \ldots, V_n$, each with domain D, and the constraint $\mathcal{C}$ holds on $\{V_1, \ldots, V_n\}$ and states that:

- $\forall i < n, M(x_i, x_{i+1}) > 0$ or $x_{i+1} = 0$;
- $x_i = 0 \implies x_{i+1} = 0$;
- $\forall k \leq n, \pi(\sum_{i=1}^{k-1} c(x_i), x_k, k)$.

Note that the first two properties can be condensed to the simpler property $\forall i < n, M(x_i, x_{i+1}) > 0$ if we add x_0, the dummy value of cost 0, to the Markov model M. This can be achieved by adding the following transition probabilities to M:

- $M(x_0, x_0) = 1$;
- $M(x_i, x_0) = 1/m, \forall i = 1, \ldots, m$.

Figure 1: A graph representation of the Markov Meter constraint of the running example. The vertices and arcs in red are added to the Markov model to deal with the variable number of values.



In this case, $\mathcal{C}$ simply states that:

- $\forall i < n, M(x_i, x_{i+1}) > 0$;
- $\forall k \leq n, \pi(\sum_{i=1}^{k-1} c(x_i), x_k, k)$.

In the next sections, we propose a pseudo-polynomial algorithm that enforces arc-consistency for this constraint.

4 Graph Representation

The arc-consistency algorithm is based on a graph structure G whose *vertices* correspond to the *values* of the domains and whose *arcs* correspond to the *transitions* of the underlying Markov model. The vertex that corresponds to value x_i for variable V_k is denoted by x_i^k.

We add starting vertex s and an ending vertex t to the graph. There are arcs from s to every vertex that represents a value of V_1 and from every vertex that represents a value of V_n to t.

Every Markov sequence of M is uniquely represented by a path from s to t.

Figure 1 shows the graph built from the example problem (P). To each vertex x_i^k is associated a set C_i^k of costs of paths supporting x_i^k, computed by algorithm described in the next section.

5 Filtering Algorithm

In this section, we present an arc-consistency algorithm for $\mathcal{C}$. This algorithm is based on the computation of the costs of the paths from s to t along which the predicate π holds.

The number of paths from s to t grows exponentially (roughly in m^n) with the size of the graph. However, the algorithm considers the *cost* of each path, not the path itself. We will show later that there is a limited number of costs, which allows the algorithm to run in pseudo-polynomial time.

Definition 1. *A* partial path *is a path from s to a vertex v of G.*

Definition 2. *A partial path* $(s, x^1_{i_1}, \ldots, x^k_{i_k})$ *is* consistent *if* $\pi\left(\sum_{j=1}^{l} c(x^j_{i_j}), x^l_{i_l}, l\right)$ *holds true* $\forall l \leq k$.

For a vertex $x^k_i \in G$, we define P^k_i, the set consisting of all paths $p = (s, x^1_{i_1}, \ldots, x^n_{i_n}, t)$ from s to t, such that $\pi\left(\sum_{l=1}^{k-1} c(x^l_{i_l}), x^k_{i_k}, k\right)$ holds $\forall k = 1, \ldots, n$. We define C^k_j the set consisting of the costs of all paths of P^k_j.

Algorithm 1 computes the C^i_j for every vertex of G.

Algorithm 1 Computing the C^k_i cost sets.

for $i = 1, \ldots, m$ **do** ▷ initialization
  if $\pi(0, x^1_i, 1)$ **then**
    $C^1_i \leftarrow \{c(x^1_i)\}$
  else
    $C^1_i \leftarrow \emptyset$
for $k = 2, \ldots, n$ **do** ▷ propagation
  for $i = 1, \ldots, m$ **do**
    $C^k_i \leftarrow \emptyset$
    for $x^{k-1}_j : M(x^{k-1}_j, x^k_i) > 0$ **do**
      $C^k_i \leftarrow C^k_i \cup \{c + c(x^k_i) : c \in C^{k-1}_j$ and $\pi(c, x^k_i, k)\}$
for $k = n-1, \ldots, 1$ **do** ▷ back-propagation
  for $i = 1, \ldots, m$ **do**
    $C' \leftarrow \emptyset$
    for $x^{k+1}_j : M(x^k_i, x^{k+1}_j) > 0$ **do**
      for $c \in C^{k+1}_j$ **do**
        if $\pi(c - c(x^{k+1}_j) - c(x^k_i), x^k_i, k)$ **then**
          $C' \leftarrow C' \cup \{c - c(x^{k+1}_j)\}$
    $C^k_i \leftarrow C^k_i \cap C'$

Note that there may exist consistent partial paths that cannot be extended to a longer consistent partial path. For a vertex x^k_i of G, it may be the case that all the consistent partial path form s to x^k_i of cost c cannot be extended to a longer consistent partial path. In this case, c should be removed from C^k_i. This is what the back-propagation phase of Algorithm 1 does.

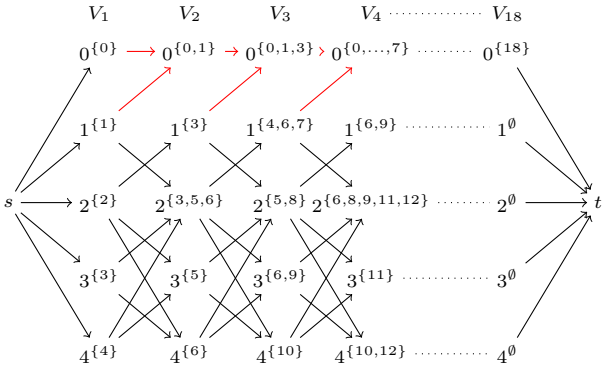
For instance, in the example (P), we mentioned that the sequence $\sigma_3 = (1, 2, 3, 4, 3, 4, 2)$ does not satisfy the constraint. In fact, the subsequence $(1, 2, 3, 4, 3, 4)$ cannot be extended to a correct sequence, and, more generally, any subsequence of sum 17 whose last element is 4 cannot be extended to a solution, as $18 - 17 = 1$ and there is no transition if M_C from 4 to 1. Hence, during the back-propagation phase, the value 17 is removed from $C^k_4, \forall k \leq n$.

The following property shows that Algorithm 1 is an arc-consistency algorithm for $\mathcal{C}$.

Property 1. $C^k_i \neq \emptyset \iff x^k_i$ *is arc-consistent with* $\mathcal{C}$.

Proof. Let $k \in \{1, \ldots, n\}$ and let $i_k \in \{1, \ldots, m\}$. Assume $x^k_{i_k}$ is arc-consistent for $\mathcal{C}$. Therefore, $\exists (x^1_{i_1}, \ldots, x^{k-1}_{i_{k-1}}, x^{k+1}_{i_{k+1}}, \ldots, x^n_{i_n}) \in V_1 \times \ldots \times V_{k-1} \times V_{k+1} \times V_n$ such that $\mathcal{C}(x^1_{i_1}, \ldots, x^n_{i_n})$ holds. It is obvious that $\forall k = 1, \ldots, n$, $\sum_{l=1}^{k} c(x^l_{i_l}) \in C^l_i$. Conversely, let $c \in C^k_i$. By the propagation phase of Algorithm 1, $\exists i_{k-1} \in \{1, \ldots, m\}$ such that $c - c(x^k_i) \in L^{k-1}_{i_{k-1}}$. Similarly, by the back-propagation phase of Algorithm 1, $\exists i_{k+1} \in \{1, \ldots, m\}$ such that $c + c(x^{k+1}_{i_{k+1}}) \in L^{k+1}_{i_{k+1}}$. By applying the same process iteratively to $k - 2, \ldots, 1$ on the left and to $k + 1, \ldots, n$ on the right, we obtain indices $i_1, \ldots, i_n$ (in which $i = i_k$). It is easy to show that $\mathcal{C}(x_{i_1}, \ldots, x_{i_k})$ holds. □

Figure 2: An part of the graph of the running example with the costs-sets computed by Algorithm 1.



6 Propagators

Algorithm 2 updates the C^k_i sets upon removal of a single value in the domain of a variable. Note that the predicate π is not used by the propagators, π is evaluated only during the initialization of the costs sets by Algorithm 1.

Let $P^k_i = \{x^{k-1}_j : x^{k-1}_j \rightarrow x^k_i\}$, the set of values that precede x^k_i in the multi-partite graph, and let $S^k_i = \{x^{k+1}_j : x^k_i \rightarrow x^{k+1}_j\}$, the set of values that follow x^k_i.

When value x^k_i is removed from the domain of variable V_k, we apply the following (left-propagation): $\forall x^{k-1}_j \in P^k_i$,

$$C^{k-1}_j \leftarrow C^{k-1}_j \bigcap \left(\bigcup_{x^k_l \in S^{k-1}_j \setminus \{x^k_i\}} (C^k_l - c(x^k_l)) \right)$$

If as a result, the cost set C^{k-1}_j is emptied, then, the corresponding value, namely x^{k-1}_j, is removed from the domain of variable V_{k-1}. This value removal is in turn propagated to the left, by applying the same procedure. Note that this removal event does not need to be propagated to the right.

Similarly, we apply the following propagation to the right: $\forall x^{k+1}_j \in S^k_i$,

$$C^{k+1}_j \leftarrow C^{k+1}_j \bigcap \left(\bigcup_{x^k_l \in P^{k+1}_j \setminus \{x^k_i\}} (C^k_l + c(x^k_l)) \right)$$

As above, if as a result, the cost set C^{k+1}_j is emptied, then, the corresponding value, namely x^{k+1}_j, is removed from the

domain of variable V_{k+1}. This value removal is in turn propagated to the right, by applying the same formulas. Note that this removal event does not need to be propagated to the left.

The domains of the variables V_{k-1} and V_{k+1} may be reduced by the application of the two formulas above. The value removal events corresponding to those variables have to be propagated to the left (for V_{k-1}) and to the right (for V_{k+1}).

Besides, some cost sets are also reduced even though the corresponding values are not removed. These cost removal event have to be propagated as well, and only in one direction, i.e., the cost removal events for values of $V_{k-1)}$ and for V_{k+1} are propagated to the left and to the right respectively.

Assume that cost c was removed from the cost set C_i^k (the cost set corresponding to the value x_i^k of variable V_k). Then $\forall x_j^{k+1} \in S_i^k$, if $\forall x_l^k \in P_j^{k+1}$, $c \notin C_l^k$ value $c + c(x_j^{k+1})$ is removed from C_j^{k+1}.

Algorithm 2 is the complete propagator for the value removal events.

7 Complexity

The worst-case complexity of Algorithm 1 is $\mathcal{O}(n \times m \times |nC|)$, where C denotes the set of costs of elements of X, i.e., $C = \{c(x) : x \in X\}$, and the sumset nC consists of all sums of n elements of C. A theorem in additive number theory by Khovanskii (Khovanskii 1992) shows that there is a polynomial $p(n)$ such that $|nC| = p(n)$, and that the degree of p is less than $|C|$. Therefore, the worst-case complexity of Algorithm 1 is $\mathcal{O}(n.m^2.p(n))$, which is $\mathcal{O}(n^{|C|}.m^2)$, i.e., pseudo-polynomial.

This pseudo-polynomial complexity is obtained because we consider the cost of paths of G instead of considering the paths themselves. This is visible in the algorithm and in the definition of the constraint $\mathcal{C}$: the predicate π is defined on the cost of the partial sequences, not on the sequences themselves.

8 Cardinality Constraints

A cardinality constraint controls the number of times a given value is taken. Let (P') be the problem defined as (P) with the additional cardinality constraint:

- (P4) : $|\{x_i \colon x_i = 4, i = 1, \ldots, n\}| = 2$

The sequence $\sigma_4 = (2, 4, 2, 4, 2, 1, 2, 1)$ is a solution. The sequence $\sigma_1 = (1, 2, 3, 4, 2, 3, 2, 1)$ is not a solution of (P') as value 4 appears only once.

In the example (P), the cost function c is implicitly the identity (each value is its own cost). However, if we change the cost function to be defined as:

$$c'(x) = \begin{cases} n \times max(X) \times meter, & \text{if } x = 4 \\ x, & \text{otherwise} \end{cases}$$

Algorithm 2 Complete propagation when value x_i^k is removed from the domain of V_k.

```
1: function PROPAGATE-REMOVE(x_i^k)
2:     LEFTREMVAL(x_i^k)
3:     RIGHTREMVAL(x_i^k)
4:
5: function LEFTREMVAL(x_i^k)
6:     for x_j^{k-1} ∈ P_i^k do
7:         U ← ∅
8:         for x_l^k ∈ S_j^{k-1} \ {x_i^k} do
9:             U ← U ∪ {c − c(x_l^k) : c ∈ C_l^k}
10:        if C_j^{k-1} ∩ U = ∅ then
11:            LEFTREMVAL(x_j^{k-1})
12:        else
13:            for c ∈ C_j^{k-1} \ U do
14:                LEFTREMCOST(c, j, k − 1)
15:        C_j^{k-1} ← C_j^{k-1} ∩ U
16:
17: function RIGHTREMVAL(x_i^k)
18:    for x_j^{k+1} ∈ S_i^k do
19:        U ← ∅
20:        for x_l^k ∈ P_j^{k+1} \ {x_i^k} do
21:            U ← U ∪ {c + c(x_l^k) : c ∈ C_l^k}
22:        if C_j^{k+1} = ∅ then
23:            RIGHTREMVAL(x_j^{k+1})
24:        else
25:            for c ∈ C_j^{k+1} \ U do
26:                RIGHTREMCOST(c, j, k + 1)
27:        C_j^{k+1} ← C_j^{k+1} ∩ U
28:
29: function LEFTREMCOST(c, i, k)
30:    for x_j^{k-1} ∈ P_i^k do
31:        if ∀x_l^k ∈ S_j^{k-1}, c ∉ C_l^k then
32:            C_j^{k-1} ← C_j^{k-1} \ {c − c(x_i^k)}
33:            if C_j^{k-1} = ∅ then
34:                LEFTREMVAL(x_j^{k-1})
35:            else
36:                LEFTREMCOST(c − c(x_i^k), j, k − 1)
37:
38: function RIGHT-REMOVE-COST(c, i, k)
39:    for x_j^{k+1} ∈ S_i^k do
40:        if ∀x_l^k ∈ P_j^{k+1}, c ∉ C_l^k then
41:            C_j^{k+1} ← C_j^{k+1} \ {c + c(x_j^{k+1})}
42:            if C_j^{k+1} = ∅ then
43:                RIGHTREMVAL(x_j^{k+1})
44:            else
45:                RIGHTREMCOST(c + c(x_j^{k+1}), j, k + 1)
```

$(n \times max(X) \times meter = 18 \times 4 \times 6 = 432.)$ If we also change the predicate π defined in (1) to:

$$\pi'(c, x, k) = (c + c'(x) = 18 + (2 \times 432) \vee k < 18) \wedge \left(\left\lfloor \frac{c}{6} \right\rfloor = \left\lfloor \frac{c+x}{6} \right\rfloor \vee c \equiv 0 \pmod 6 \right)$$

The new **meter** constraint $\mathcal{C}_{\pi',c'}$ is equivalent to the problem (P'). Such a modification of the cost function and of π' used to control the cardinality of several values of the domain (akin to **gcc**). This method is of course limited to max costs that can be represented by integers.

9 Generation

We have presented an efficient arc-consistency algorithm for the **meter** constraint. Applying this algorithm to a problem consisting of one **meter** constraint ensures that the problem becomes backtrack-free. For this kind of problems, Algorithm 1 can straightforwardly be turned into a greedy generation procedure.

- Probabilities

The transition probabilities of the Markov model are ignored in the definition of the **meter** constraint. However, they can be introduced in the generation process as shown in (Pachet, Roy, and Barbieri 2011). A simple right-to-left normalization of the transition probabilities can be applied. After this normalization procedure, a greedy procedure will generate the solution sequences with the same probability distribution as that of the original model, provided that values are chosen with respect to the normalized probabilities.

The **meter** constraint $\mathcal{C}$ build for a Markov model M may be seen as a Markov model statistically equivalent to M, but that generates only those sequences that satify $\mathcal{C}$.

- Higher orders

Our **meter** constraints addresses order 1 Markov models. However, there is no restriction to order 1 for generation. Because the CSP is arc-consistent, we can ensure that no solution at order 1 is lost, which implies that no solution at higher orders is lost either. Therefore, any greedy strategy can be implemented as a value choice heuristics. For instance, so-called variable-order strategy can be implemented to favor longest possible prefixes. During the greedy generation, after k values $(x_1, \ldots, x_k)$ have been selected (for the first k variables), the $k+1$ value will be selected from the set of values x_{k+1} such that $M(x_{k-1}, x_k, x_{k+1}) > 0$. If no such value is available, the selection is among all the x_{k+1} such that $M(x_k, x_{k+1}) > 0$, i.e., at order 1.

10 Application to Melody Generation

Rhythm and meter are notoriously difficult to learn using statistical models. In particular, meter can be used to establish long-term dependencies in the events making up a melody, which escape all statistical models known so far. (Paiement et al. 2008) propose a distance measure between rhythms to predict rhythm continuations but their model does not apply readily to melodic learning and generation.

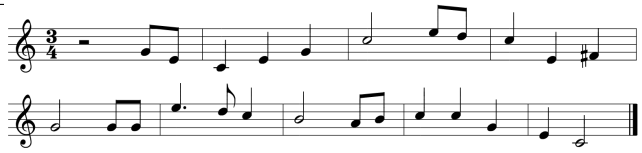


Figure 3: The melody of Star Spangled Banner. Note that all bars have the same duration, but a different number of notes.

Figure 4: The transition probabilities for the pitches of the melody of Figure 3

	C_3	E_3	$F_3^\#$	G_3	A_3	B_3	C_4	D_4	E_4
C_3	0	1	0	0	0	0	0	0	0
E_3	1/2	0	1/4	1/4	0	0	0	0	0
$F_3^\#$	0	0	0	1	0	0	0	0	0
G_3	0	1/3	0	1/3	0	0	1/6	0	1/6
A_3	0	0	0	0	0	1	0	0	0
B_3	0	0	0	0	1/2	0	1/2	0	0
C_4	0	1/5	0	1/5	0	1/5	1/5	0	1/5
D_4	0	0	0	0	0	0	1	0	0
E_4	0	0	0	0	0	0	0	1	0

In the context of Markov modelling, several approaches have been followed to "learn about meter". One consists in building appropriate viewpoints for the data to be learned. Indeed, there is a great variability of the choice of the particular viewpoint used to build the Markov models from input data. If we consider sequences of notes, many different viewpoints can be considered (Conklin and Witten 1995). It is therefore possible to associate to each note in the corpus its metric position (say, in a quantized time with 16 different positions per bar). A possible viewpoint could then be the couple pitch, metric position. The problem with this approach is that the learned model will be sparse, since there are very few alternative for a given couple (pitch, position). Another approach, followed by (Davismoon and Eccles 2010) is to use heuristics such as stochastic optimization to attempt to find solution during the search. These approaches may work in practice, especially when the model is dense, but without any guarantee whatsoever.

Our approach consists in exploring the whole search space, including all the combinatorial possibilities offered by meter, therefore offering a guarantee that all solutions are found, in reasonable (pseudo polynomial) time. This is, to our knowledge, the first approach that solves this problem properly.

In the examples below, we use a simple viewpoint (only pitch), to create a rich (though not complete) Markov model. By definition, this model does not capture meter, rhythm or any long-term strutural information. We then enforce **meter** constraints to re-establish this information in the generated sequences.

More precisely, we build a Markov model from the melody of Figure 3. The transition probabilities of the model are shown on Figure 4.

Without any other constraint, we obtain melodies such as the one in Figure 5. It can be observed that no metric is

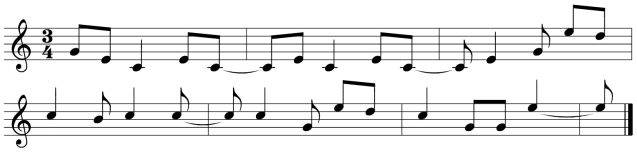


Figure 5: A 25-note melody generated from the Markov model of Star Spangled Banner with no constraint on meter or on the global duration.

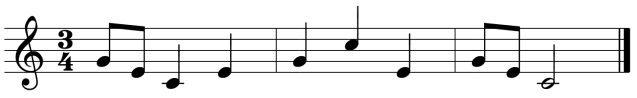


Figure 6: A 3-bar melody generated from the Markov model of Star Spangled Banner with a **meter** constraint.

enforced (total duration is not a integer number of bars, and several notes span over two bars).

We then add a **meter** constraint, ensuring that there is 1) a total duration of 3 bars, and 2) no note spans across bars. We obtain a total of 4467 solutions with no backtrack. An example is shown in Figure 6.

We can further refine our CSP by adding a cardinality constraint stating that we want exactly two $F_3^{\#}$ pitches in the melody. This is an interesting constraint in our context since $F_3^{\#}$ is the most dissonant note in the C major tonality. We obtain 35 solutions only. Two examples are shown in Figure 7 and Figure 8. Figure 7 is particularly interesting rhythmically as a new rhythmical pattern (slightly syncopated) is created that was not in the training corpus, but which is nevertheless valid musically. Such a rhythm could in turn be filtered out by enforcing a stricter **meter** constraint that forbids syncopations.

It is interesting to look at the probabilities of all these sequences. Among all Markov sequences with a total duration of three bars, the 4467 sequences with a correct meter have a cumulated probability of .30304. Among these, those with two $F_3^{\#}$ have a total probability of .0043. They are therefore quite "rare" and thus hard to find using heuristic approaches.

These short examples illustrate the semantics of the **meter** constraint applied to melody generation. The constraint can be used on much larger sequences covering all practical needs in music composition, typically up to a few thousands of notes (representing several minutes of music), in real-time.

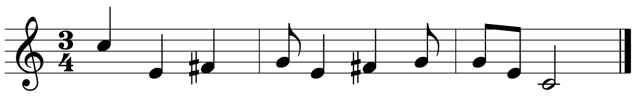


Figure 7: A 3-bar melody generated from the Markov model of Star Spangled Banner with a **meter** constraint and a cardinality constraint stating that two $F_3^{\#}$ must appear in the melody.

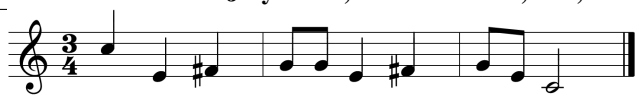


Figure 8: Another solution (see Figure 7). This solution does not have the syncopated rhythm of Figure 7.

These examples enforce meter at only one level (the *bar* level). The same technique can be used to enforce meter at a hierarchical level (e.g., strong/weak beat, bar, phrase, section).

The same holds for text generation: the constraint can be used to generate sentences or paragraphs with metrical properties, such as song lyrics or poetry. For instance, a typical text corpus such as Proust's "A la Recherche du Temps Perdu" contains about $50,000$ phrases and $20,000$ unique words. Our current experiments show that our constraint is able to generate Alexandrines (12-syllable-verses) from this corpus in less than a second.

11 Conclusion

Our contribution is two-fold. First, we introduced **meter**, a global constraint that ensures meter properties on text or music sequences. Those properties are defined by a single predicate π which is enforced on all subsequences from the first item to any other item. This formulation enables the definition of many constraints such as **sum-equals**, music meter, cardinality, and many others. Its complexity is pseudo-polynomial, which is sufficient for our text and music applications.

There may be relations between our constraint and other global constraints, but if there are, they are not obvious to highlight. Recent work by N. Beldiceanu on automata reformulations of global constraints (Beldiceanu et al. 2005) may provide alternative implementations for **meter**. However, since **sum-equals** is known to be NP-hard (Bordeaux et al. 2011) it is unlikely that much better filtering procedures exist for **meter**.

In practice **meter** is a crucial element to build a new generation of authoring tools in which users post constraints interactively to control the generation of sequences in a given style. This is the goal of the Flow Machines project, which aims at applying these ideas in music composition and literary text writing.

Acknowledgment: This research is conducted within the Flow Machines project which received funding from the European Research Council under the European Union's Seventh Framework Programme (FP/2007-2013) / ERC Grant Agreement n. 291156. The authors would like to thank Gilles Pesant for helpful comments about **regular** and **cost-regular**.

References

Aggoun, A., and Beldiceanu, N. 1993. Extending CHIP in order to Solve Complex Scheduling and Placement Problems. *Mathl. Comput. Modelling* 17(7):57–73.

Barbieri, G.; Pachet, F.; Roy, P.; and Esposti, M. D. 2012. Markov constraints for generating lyrics with style. In *ECAI*, 115–120.

Beldiceanu, N.; Carlsson, M.; Debruyne, R.; and Petit, T. 2005. Reformulation of global constraints based on constraints checkers. *Constraints* 10(4):339–362.

Bordeaux, L.; Katsirelos, G.; Narodytska, N.; and Vardi, M. Y. 2011. The complexity of integer bound propagation. *J. Artif. Intell. Res. (JAIR)* 40:657–676.

Brooks, Jr., F. P.; Hopkins, Jr., A. L.; Neumann, P. G.; and Wright, W. V. 1992. *An experiment in musical composition*. Cambridge, MA, USA: MIT Press. 23–40.

Conklin, D., and Witten, I. H. 1995. Multiple viewpoint systems for music prediction. *Journal of New Music Research* 24:51–73.

Davismoon, S., and Eccles, J. 2010. Combining musical constraints with Markov transition probabilities to improve the generation of creative musical structures. *EvoApplications* 2:361–370.

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

Khovanskii, A. 1992. Newton polyhedron, Hilbert polynomial, and sums of finite sets. *Functional Analysis and Its Applications* 26(4):276–281.

Nierhaus, G. 2009. *Algorithmic Composition, Paradigms of Automated Music Generation*. Springer-Verlag.

Pachet, F., and Roy, P. 2011. Markov constraints: steerable generation of Markov sequences. *Constraints* 16(2).

Pachet, F.; Roy, P.; and Barbieri, G. 2011. Finite-length Markov processes with constraints. In *IJCAI*, 635–642.

Paiement, J.-F.; Bengio, S.; Grandvalet, Y.; and Eck, D. 2008. A generative model for rhythms. In *Neural Information Processing Systems, Workshop on Brain, Music and Cognition*.

Pesant, G. 2004. A regular language membership constraint for finite sequences of variables. In Wallace, M., ed., *CP*, volume 3258 of *Lecture Notes in Computer Science*, 482–495. Springer.

Petit, T.; Beldiceanu, N.; and Lorca, X. 2011. A generalized arc-consistency algorithm for a class of counting constraints. In Walsh, T., ed., *IJCAI*, 643–648. IJCAI/AAAI.

Régin, J.-C. 1996. Generalized arc consistency for global cardinality constraint. In *AAAI/IAAI, Vol. 1*, 209–215.

Zhang, Y., and Yap, R. H. C. 2000. Arc consistency on *n*-ary monotonic and linear constraints. In Dechter, R., ed., *CP*, volume 1894 of *Lecture Notes in Computer Science*, 470–483. Springer.