Avoiding Plagiarism in Markov Sequence Generation

Alexandre Papadopoulos, [1,2] **Pierre Roy,** [1] **François Pachet**[1,2]
[1]Sony CSL, 6 rue Amyot, 75005, Paris, France
[2]Sorbonne Universités, UPMC Univ Paris 06, UMR 7606, LIP6, F-75005, Paris, France

Abstract

Markov processes are widely used to generate sequences that imitate a given style, using random walk. Random walk generates sequences by iteratively concatenating states to prefixes of length equal or less than the given *Markov order*. However, at higher orders, Markov chains tend to replicate chunks of the corpus with a size possibly higher than the order, a primary form of plagiarism. The *Markov order* defines a maximum length for training but not for generation. In the framework of constraint satisfaction (CSP), we introduce MAXORDER. This global constraint ensures that generated sequences do not include chunks larger than a given *maximum order*. We exhibit an automaton that recognises the solution set, with a size linear in the size of the corpus. We propose a linear-time procedure to generate this automaton from a corpus and a given max order. We then use this automaton to achieve generalised arc consistency for the MAXORDER constraint, holding on a sequence of size n, in $O(n.T)$ time, where T is the size of the automaton. We illustrate our approach by generating text sequences from text corpora with a maximum order guarantee, effectively controlling plagiarism.

Introduction

Markov chains are a powerful, widely-used technique to analyse and generate sequences that imitate a given style (Brooks et al. 1957; Pinkerton 1956), with applications to many areas of automatic content generation such as music, text, line drawing and more generally any kind of sequential data. A typical use of such models is to generate *novel* sequences that "look" like or "sound" like the original.

From a corpus of finite-length sequences considered as representative of the style of an author, a Markov model of the style is estimated based on the Markov hypothesis which states that the future state of a sequence depends only on the last state, i.e.:

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

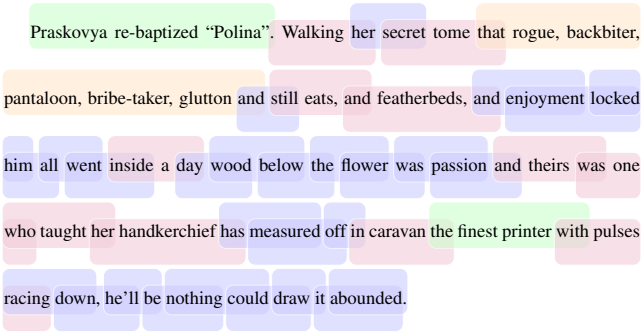
The equation above describes a Markov model of order 1. The definition can be extended to higher orders by considering prefixes of length k greater than 1.

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

In theory, higher order Markov models are equivalent to order 1 models. However, in practice, higher order models offer a better compromise between expressivity and representation cost. *Variable order* Markov models are often used to produce sequences with varying degrees of similarity with the corpus (Begleiter, El-Yaniv, and Yona 2004). Indeed, increasing the Markov order produces sequences that replicate larger chunks of the original corpus, thereby improving the impression of style imitation.

However, it has also been long observed (Brooks et al. 1957) that increasing the order tends to produce sequences that contain chunks of the corpus of size much larger than the Markov order.

We illustrate this phenomenon on a text corpus: Johnston's English translation of Pushkins Eugene Onegin – a reference to Markov, as he used the same corpus (in Russian) for his pioneering studies. Here, an element of the Markov chain is a word of the text or a sentence separator, and a sequence is a succession of such elements. With a Markov order of 1, we obtain the following sequence:



On top of the text, we draw the longest subsequences that appear verbatim the corpus, or chunks, assigning different colours to different lengths. For example, this generated sequence contains the chunk "[...] that rogue, backbiter, pantaloon, bribe-taker, glutton and [...]", which is a subsequence of length 7 from the corpus. The maximum order of a se-

Markov order	median	lower quartile	upper quartile
1	2	2.0	3.0
2	2	2.0	3.0
3	9	6.0	13.0
4	45	28.0	65.0
5	78	77.0	79.0
6	78	78.0	79.0

Table 1: Maximum orders for different Markov orders

quence is the maximum length of its chunks (7, in our example).

If we increase the order to 3, we obtain sequences such as the following one:

Love's frantic torments went on beating and racking with their strain and stress that youthful heart. It all seemed new – for two days only – the estate provides a setting for angry heirs, as one, to admire him – and replies: Wait, I'll present you – but inside a day, with custom, love would fade away. It's right and proper that you transcend in music's own bewitching fashion the foreign words a maiden's passion found for its utterance that night directed his.

This sequence makes arguably more sense than the one generated with order 1. However, its maximum order is 20 (i.e. it contains a 20-word-long subsequence copied verbatim from the corpus). To any reader familiar with the corpus, this would read like blatant plagiarism.

To illustrate this phenomenon in a more general way, we generated a few hundreds of sequences of varying order (from to 1 to 6). Table 1 shows the maximum order observed for each Markov order: this value increases to values much higher than the Markov order. Markov order affects *training*: when estimated from a corpus, the Markov model learns all continuations of sequences of k states or less. However, this parameter k does not limit the maximum order of the generated sequence. In particular, the whole corpus itself is trivially a valid Markov sequence of order k.

This paper addresses precisely the problem of generating Markov sequences with an imposed maximum order. Such sequences cannot be obtained with greedy approaches like random walk. However, the maximum order problem can be seen as a specific instance of a constrained Markov problem. Introduced by Pachet and Roy (2011), Markov constraints consist in reformulating the generation of Markov sequences as a combinatorial problem (CSP). By enforcing arc-consistency of the Markov constraints, we can solve such a CSP in a backtrack-free manner, thus simulating a greedy random walk generation procedure.

As opposed to greedy approaches, the Markov constraint approach guarantees a complete exploration of the space of sequences. The variables of this CSP represent the elements of the sequence to generate. Markov constraints enforce that variables in the sequence should have valid Markov continuations. Sequence generation is obtained by CSP resolution. Additional constraints can be easily introduced to control the generation of the sequence. In practice, the problem boils down to the identification of efficient arc-consistency algorithms for the given constraints.

Under this framework, we introduce the MAXORDER global constraint. It holds on all the variables of the CSP, and states that 1) the sequence is a valid order k Markov sequence, and 2) no L consecutive variables form a sequence that belongs to the training corpus. Following Pesant (2004) and Beldiceanu, Carlsson, and Petit (2004), we enforce generalised arc-consistency by feeding a constraint such as REGULAR an automaton that accepts the set of such sequences. However, we show that canonical methods are not satisfactory for building this automaton. The main contribution of this paper is a linear-time algorithm that builds this automaton.

Running Example

We consider the corpus made of ABRACADABRA, where each element is one of the symbols A, B, C, D or R. With $k = 1$, the Markov chain estimated on this corpus is given by the following transition matrix:

$$\begin{array}{c} \\ A \\ B \\ C \\ D \\ R \end{array} \begin{array}{c} \begin{array}{ccccc} A & B & C & D & R \end{array} \\ \begin{pmatrix} 0 & 0.5 & 0.25 & 0.25 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

During a training phase, these probabilities are estimated according to their frequency in the corpus. Here, in the four continuations for A in the corpus, two are with B, one with C and one with D. A sequence is a Markov sequence, according to an order k Markov chain, if every k-gram of the sequence has a continuation with a non-zero probability. For example, ABRADABRACA is a valid Markov sequence, but ABRACADABA is not a valid Markov sequence, because the probability of having A after B is zero.

Automaton Representation

Following the works on language automata and global constraints, we can encode a set of Markovian sequences using an automaton. Global constraints such as REGULAR can then take this automaton as input to generate sequences.

Definition 1 (Automata). A deterministic finite-state automaton, or, simply, automaton, is a quintuple $\mathcal{A} = \langle Q, \Sigma, \delta, q_0, F \rangle$, where:

- Q is a finite non-empty set of states;
- Σ is the alphabet – a finite non-empty set of symbols;
- $q_0 \in Q$ is the initial state of the automaton;
- δ is the transition function $Q \times \Sigma \to Q$, which maps a state to its successors for a given symbol;
- $F \subseteq Q$ is the set of final, or accepting, states.

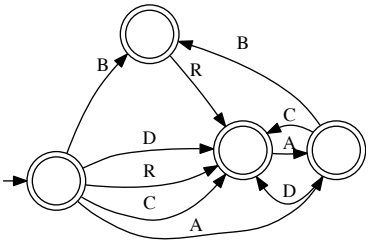


Figure 1: A Markov automaton for the ABRACADABRA corpus, with $k = 1$

Definition 2 (Accepted language). An automaton recognises, or accepts, a set of words, called a language, defined as follows.

- A *word* $w \in \Sigma^*$ is a sequence $a_1 \dots a_p$ of symbols $a_i \in \Sigma$.
- The word w is accepted by $\mathcal{A}$ if and only if there exists a sequences $q_0, \dots, q_p$ of states, such that $\delta(q_{i-1}, a_i) = q_i$, for all i, $1 \leq i \leq p$, and $q_p \in F$.
- $\mathcal{L}(\mathcal{A}) = \{w \in \Sigma^* | w \text{ is accepted by } \mathcal{A}\}$ is the accepted language of $\mathcal{A}$.

In order to represent order k Markov transitions, we create an alphabet Σ where each symbol corresponds to a unique k-gram of the corpus. Then, a valid order k Markov transition is represented by two symbols, such that their two corresponding k-grams overlap on their common $k-1$ symbols. A valid Markov sequence of length n is represented by a word of length $n-k+1$ on this alphabet.

For example, for $k = 2$, the sequence ABRA corresponds to a sequence of three 2-grams $\langle A, B\rangle$, $\langle B, R\rangle$, $\langle R, A\rangle$. We can assign three symbols $a_1, a_2, a_3 \in \Sigma$ to those three 2-grams in their respective order. The Markov transition $A, B \rightarrow R$ is represented by the word $a_1 a_2$, and the sequence ABRA by the word $a_1 a_2 a_3$.

Definition 3 (Markov Automaton). A Markov automaton associated to a corpus is an automaton $\mathcal{A}$ such that $\mathcal{L}(\mathcal{A}) = \{a_1 \dots a_n \in \Sigma^* | a_i a_{i+1} \text{ is a Markov transition}, \forall i, 1 \leq i \leq n\}$

Figure 1 shows a Markov automaton for the corpus ABRACADABRA with $k = 1$.

We can now represent the set of Markov sequences satisfying the maximum order property as the following automaton.

Definition 4 (Maximum Order Automaton). Let $\mathcal{M}$ be a Markov automaton for a given corpus. For a given no-good $a_1 \dots a_L \in \mathcal{N}$, let $\mathcal{A}(a_1 \dots a_L)$ be an automaton such that $\mathcal{L}(\mathcal{A}(a_1 \dots a_L)) = \{w \in \Sigma^* | a_1 \dots a_L \text{ is a factor of } w\}$, i.e. the language of words containing at least one occurrence of the no-good. An automaton $\mathcal{MO}$ is a maximum order automaton for $\mathcal{C}$ and $\mathcal{N}$ if:

$$\mathcal{L}(\mathcal{MO}) = \mathcal{L}(\mathcal{M}) \cap \bigcap_{a_1 \dots a_L \in \mathcal{N}} \overline{\mathcal{L}(\mathcal{A}(a_1 \dots a_L))}$$

Figure 2 shows a maximum order automaton for the corpus ABRACADABRA, with $k = 1$ and $L = 4$. The labels in the states correspond to prefixes of forbidden no-goods.

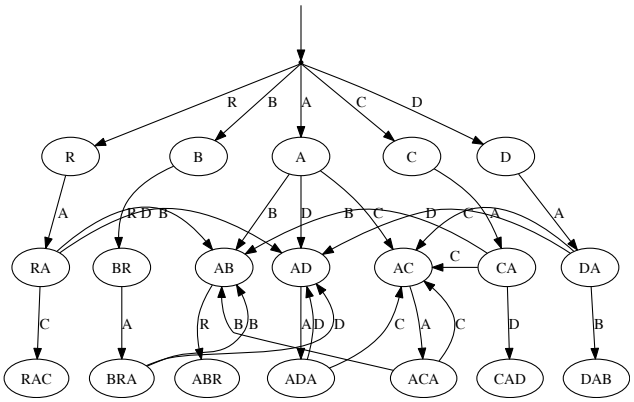


Figure 2: A maximum order automaton for the ABRACADABRA corpus, with $k = 1$ and $L = 4$

The MAXORDER Constraint

The purpose of MAXORDER is to generate Markov sequences with a guaranteed maximum order. The constraint takes two parameters $\mathcal{C}$ and $\mathcal{N}$; $\mathcal{C} \subseteq \Sigma^2$ denotes a set of valid Markov transitions, each represented by an element $a_1 a_2 \in \mathcal{C}$ corresponding to a valid pair of k-grams; $\mathcal{N} \subseteq \Sigma^L$ denotes a set of forbidden sequences, or no-goods.

In practice, each no-good of size $L = L' - k + 1$ corresponds to a sequence of size L' that appears in the corpus, where k is the Markov order, and L' is the maximum order.

MAXORDER is defined as follows.

Definition 5 (MAXORDER constraint). The constraint MAXORDER$(\mathcal{C}, \mathcal{N}, X_1, \dots, X_n)$ holds iff:

- for each i, $1 \leq i < n$, $X_i X_{i+1} \in \mathcal{C}$;
- for each i, $1 \leq i < n - L' + 1$, $X_i \dots X_{i+L'-1} \notin \mathcal{N}$.

For example consider the ABRACADABRA corpus, with $k = 1$ and $L = 4$. There are 7 no-goods: ABRA, BRAC, RACA, ACAD, CADA, ADAB, DABR. The Markovian sequence ABRADABRACA does not satisfy the maximum order property: it contains the no-goods ABRA, ADAB, DABR, BRAC, RACA. The Markovian sequence RADADACAB does not contain any no-good of size 4, and so satisfies the maximum order property for $L = 4$. In fact, any satisfying sequence is a substring of a string following the pattern $BRA(DA)^+(CA)^+BR$.

Propagating the MAXORDER Constraint

There is a canonical way to propagate MAXORDER by considering the maximum order automaton. Then, we can enforce the MAXORDER constraint by imposing a sequence of variables to form a word recognised by this automaton, using the REGULAR constraint (Pesant 2004). If T is the size of the automaton (the number of transitions), generalised arc-consistency on a sequence of n variables is achieved in $O(n.T)$ time.

The maximum order automaton can be built using standard automata theory operations that implement Definition 4. Initially, we build a Markov automaton (we provide

in this paper an algorithm for doing this). Then, for each no-good, this automaton is intersected with the negation of the automaton recognising sequences containing the no-good.

However, the complexity of this procedure is dominated by the complexity of intersecting a number of automata. If $O(t)$ is the size of any of the automata, and $N = |\mathcal{N}|$ is the number of no-goods, the complexity is $O(t^N)$. It is unlikely that an algorithm with a better complexity exist (Karakostas, Lipton, and Viglas 2000; 2003). Furthermore, this method does not give any bound on the size of the final automaton (other than $O(t^N)$). In the following section, we propose a linear-time algorithm to build this automaton.

Algorithms

In this section, we build the maximum order automaton in time linear in the size of the input, the set of no-goods. As a corollary, we show that the size T of this automaton is linear in the size of the input, and, therefore, that propagating the MAXORDER is polynomial too. To this end, we introduce two algorithms. The first algorithm builds the minimal Markov automaton. The second algorithm builds a trie with the no-goods, computes their overlaps, and uses it to remove from the Markov automaton all sequences containing a no-good.

Markov automaton

The first algorithm consists in building an automaton recognising all Markovian sequences, i.e. sequences where any two successive k-grams correspond to a $k+1$-gram of the corpus, for a Markov order of k. This automaton, when viewed as a labelled directed graph, is essentially a syntactic rewrite of the Markov chain, with a different semantics attached to its nodes and edges. In this automaton, each transition is labelled by a symbol corresponding to a Markov state. A notable property of a Markov automaton is that all transitions labelled with a given symbol point to the same state.

Definition 6 (State labels). We use the following notation to relate states and the labels of its ingoing transitions.

- Let q be a state, $a(q) = \{a \in \Sigma | \exists q' \in Q, \delta(q', a) = q\}$ is the set of labels of the transitions pointing to q.
- Let a be a symbol of the alphabet, $Q(a)$ is the *unique* state q such that $a \in a(q)$.

The interpretation of a Markov automaton $\mathcal{A}$ can be stated formally as follows. Let $a_1, a_2 \in \Sigma$ be two symbols. A Markov transition between the k-grams corresponding to a_1 and a_2 is represented in $\mathcal{A}$ by a transition between $Q(a_1)$ and $Q(a_2)$, labelled by a_2. Intuitively, when at state $q = Q(a_1)$, we can observe any one of the Markov states in $a(q)$. If we observe a_1, we can generate a_2. Since the automaton accepts sequences of arbitrary length, all states are accepting.

We build the Markov automaton using Algorithm 1. First a state is created that observes any Markov state (Lines 2 to 7). Then, each Markov transition is inserted iteratively. When inserting (a_1, a_2), a new state q is created using the `separate()` function. This function creates a new state, which has the same outgoing transitions as q_1, and redirects all ingoing a_1-labelled transitions from q_1 to the newly created state. An a_2-labelled transition can be added. Minimality is incrementally maintained by assuming the invariant that, before the call to `separate()`, all states are pairwise non-equivalent. The creation of q does not affect the equivalency of any other state (in particular, not the predecessors of q and q_1). After we add an a_2-labelled transition to q, q ceases to be equivalent with q_1. However, with the addition of the transition, it might have become equivalent to a (unique) other state, in which case we merge them, thus maintaining the invariant.

The size of the resulting automaton and the running time to build it, are both bounded by the number of transitions in $\mathcal{C}$.

Algorithm 1: Markov automaton

Data: $\mathcal{C}$ the set of valid Markovian transitions
Result: $\mathcal{M}$ the minimal Markov automaton

1 $\mathcal{M} \leftarrow \langle Q, \Sigma, \delta, q_0, F \rangle$
2 $q \leftarrow NewState(Q)$
3 $a(q) \leftarrow \emptyset$
4 **forall the** $a \in \Sigma$ **do**
5 $\quad \delta(q_0, a) \leftarrow q$
6 $\quad Q(a) \leftarrow q$; $a(q) \leftarrow a(q) \cup \{a\}$
7 $F \leftarrow F \cup \{q_0, q\}$
8 **forall the** $a_1 a_2 \in \mathcal{C}$ **do**
9 $\quad q_1 \leftarrow Q(a_1)$
10 $\quad q \leftarrow$ `separate`(q_1, a_1)
11 $\quad q_2 \leftarrow Q(a_2)$
12 $\quad \delta(q, a_2) \leftarrow q_2$
13 $\quad$ **if** $\exists q' \in Q$ *such that* q *and* q' *are equivalent* **then**
14 $\quad\quad$ Merge q with q'
15 $\quad\quad Q(a_1) \leftarrow q'$; $a(q') \leftarrow a(q') \cup a(q)$
16 **function** `separate`(q_1, a_1)
17 $\quad q \leftarrow NewState(\mathcal{A})$
18 $\quad accept(q) \leftarrow True$
19 $\quad$ **forall the** $a \in \Sigma$ *such that* $\delta(q_1, a)$ *is def* **do**
20 $\quad\quad \delta(q, a) \leftarrow \delta(q_1, a)$
21 $\quad$ **forall the** $q' \in Q$ *such that* $\delta(q', a_1) = q_1$ **do**
22 $\quad\quad \delta(q', a_1) \leftarrow q$
23 $\quad Q(a_1) \leftarrow q$; $a(q) \leftarrow \{a_1\}$

Maximum Order Automaton

The second algorithm consists in removing from the Markov automaton any sequence that contains at least one no-good, i.e. a sequence of forbidden length appearing in the corpus. This is performed by Algorithm 2. Intuitively, a trie of the no-goods is computed, where all states but the ones corresponding to a full no-good are accepting states. This ensures that a no-good is never accepted. However, this is not sufficient. The key part of the algorithm is to add the connection between overlapping prefixes. For example, if we have two no-goods ABCD and BCEF, the prefixes ABC and BCEF

Algorithm 2: MAX ORDER AUTOMATON

Data: $\mathcal{N}$ the set of forbidden no-goods
$\mathcal{M} \leftarrow \langle Q, \Sigma, \delta, q_0, F \rangle$: a Markov automaton
Result: Any word containing at least one no-good is not recognised by $\mathcal{M}$

```
// Remove transitions that start a
   no-good
```
1 **forall the** $a_1 \dots a_L \in \mathcal{N}$ **do**
2 $\quad q \leftarrow Q(a_1)$
3 $\quad q \leftarrow \texttt{separate}(q, a_1)$
4 $\quad$ Clear outgoing transitions of q
5 $\quad w(q) \leftarrow (a_1)$

```
// Compute the trie of no-goods
```
6 $Q_{trie} \leftarrow \emptyset$
7 **forall the** $a_1 \dots a_L \in \mathcal{N}$ **do**
8 $\quad q \leftarrow q_0$
9 $\quad i \leftarrow 1$
10 $\quad$ **while** $\delta(q, a_i)$ *exists* **do**
11 $\quad\quad q \leftarrow \delta(q, a_i)$
12 $\quad\quad i \leftarrow i + 1$
13 $\quad$ **for** $a_j, i \leq j \leq L$ **do**
14 $\quad\quad q' \leftarrow NewState(Q_{trie})$
15 $\quad\quad F \leftarrow F \cup \{q'\}$
16 $\quad\quad \delta(q, a_j) \leftarrow q'$
17 $\quad\quad w(q') \leftarrow (a_1, \dots, a_j)$
18 $\quad\quad a(q') \leftarrow \{a_j\}$
19 $\quad\quad q \leftarrow q'$
20 $\quad F \leftarrow F \setminus \{q\}$

```
// Compute cross prefix transitions
```
21 **forall the** $q \in Q_{trie}$ **do**
22 $\quad S(q) \leftarrow \{q' \in Q_{trie} | w(q)$is a strict suffix of $w(q')\}$
23 **forall the** $q \in Q_{trie}$ *in order of decreasing* $|w(q)|$ **do**
24 $\quad$ **forall the** $a \in \Sigma$ *such that* $\delta(q, a)$ *exists* **do**
25 $\quad\quad$ **forall the** $q' \in S(q)$ **do**
26 $\quad\quad\quad$ **if** $\delta(q', a)$ *is undefined* **then**
27 $\quad\quad\quad\quad \delta(q', a) \leftarrow \delta(q, a)$ `// transition is Markovian`
28 $\quad\quad\quad\quad$

```
// Markovian completion
```
29 **forall the** $\forall q \in Q_{trie}$ **do**
30 $\quad \{a_1\} \leftarrow a(q)$
31 $\quad$ **forall the** $a_2 \in \Sigma$ *such that* $a_1 a_2 \in \mathcal{C}$ **do**
32 $\quad\quad$ **if** $\delta(q, a_2)$ *is undefined* **then**
33 $\quad\quad\quad \delta(q, a_2) \leftarrow Q(a_2)$
34 $Q \leftarrow Q \cup (Q_{trie} \cap F)$

overlap on BC. This means that the automaton should not accept ABCD, but it should not accept ABCEF either. This connection is made using *cross-prefix transitions*. In order to achieve this, Algorithm 2 uses an adaptation of the Aho and Corasick (1975) string-matching algorithm.

In more details, this algorithm first disconnects from the Markov automaton any transition that starts a no-good. Then, those transitions are extended to a full trie of all the no-goods. For a state q of the trie, $w(q)$ records the prefix recognised by this state, i.e. the word accepted from q_0 to q. By excluding the states that recognise a complete no-good (line 20), we guarantee that no sequence including any no-good will be recognised. However, we are not removing those states yet. We first have to deal with a key aspect of this algorithm, which concerns the overlap between no-goods. For example, with two no-goods ABCD and BCEF, supposing we started ABC, we cannot continue with D, as this would complete a no-good. However, if we continue with E, we start the no-good BCEF. Therefore, a cross-prefix transition between the state for ABC and the state for BCE must be added. Cross-prefix transitions ensure that, by avoiding a particular no-good, we will not inadvertently complete another no-good. We use an adaptation of the Aho and Corasick string-matching algorithm (Aho and Corasick 1975): when we use a transition that does not extend the current prefix, we can jump directly to the longest no-good that is a suffix of the current prefix. Finally, we add transitions in the trie for any state missing some valid Markov transitions. Those transitions either point back to the original Markov automaton, for Markov transitions that do not start any no-good, or point to the states of the first layer of the trie, for Markov transitions that start a new no-good. Since we kept the transitions to the non-accepting states that complete a no-good, we know we are not introducing any no-good. We can know finally remove those non-accepting states (line 34).

The algorithm adds exactly once each transition of the resulting automaton. Therefore, it runs in time linear in the number T of transitions of the final automaton. Let $N = L.|\mathcal{N}|$ be size of the input $\mathcal{N}$. When constructing the trie, it creates exactly N transitions. During the next phase, the added transitions are exactly those added by the Aho and Corasick algorithm. Their number is linearly bounded by N, a (non-trivial) result from Aho and Corasick (1975). Finally, the number of transitions added to each state during the completion phase is bounded by $|\Sigma|$, which is independent of $\mathcal{N}$.

Note that the general idea of this algorithm is similar to the algorithm by (Villeneuve and Desaulniers 2005), which computes shortest paths with forbidden paths. However, they operate in a very different context, and are only interested in shortest paths, whereas we are interested in all sequences.

Generating Sequences with a Maximum Order guarantee

We can use the maximum order automaton to generate sequences, by using any implementation of REGULAR. In practice, we use a decomposition based method (Quimper and Walsh 2006; 2008), which works better in practice with large automata, by exploiting efficient propagators for extensional (table) constraints.

In order to simulate a random walk procedure with a variable order, we chose the following variable and value ordering heuristics. We use static variable ordering heuristics, in their temporal order. For a given variable X_i, we consider up to k previous variables. We then consider all the contin-

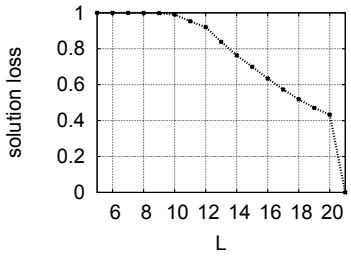


Figure 3: Solution loss from MAXORDER, on "Eugene Onegin" with $k = 3$, $n = 20$, for various values of L

uations in the corpus for the sequence $X_{i-k}, \ldots, X_{i-1}$ *that are arc-consistent* for MAXORDER. If the number of such continuations is above a certain fixed threshold, we choose one, using the transition probabilities. Otherwise, the procedure is repeated for a smaller k, until one value is found. We start with $k = L - 1$. Note that using the transition probabilities does not necessarily generate sequences with the right probability distribution (Pachet, Roy, and Barbieri 2011). This is however a reasonable approximation in the context of this paper.

A resolution on this model will be backtrack-free, and will simulate a random walk with a variable order – with the aforementioned caveat concerning transition probabilities – with the maximum order guarantee.

Evaluation

We applied our algorithm on the "Eugene Onegin" corpus (2160 sentences, 6919 unique words, 32719 words in total). An interesting question is how likely a stochastic method finds sequences satisfying the maximum order property. As we mentioned at the beginning of the paper, it has been widely observed that high order Markov models tend to replicate large chunks of the corpus. Our model enables us to make exact solution counting (as a consequence of having arc-consistency). We first count the total number S of Markovian sequences of length $n = 20$, with a Markov order 3, based on the Eugene Onegin corpus. We compare this to the number S_L of Markovian sequences with a maximum order of L, with L ranging from 5 (the minimum non-trivially infeasible value), to 21 (the minimum value for which MAXORDER is trivially satisfied). We call *solution loss* the ratio $1 - (S_L/S)$: the closer it is to 1, the more Markovian sequences are "ruled out" because they do not satisfy the maximum order property for L. We show the results on Figure 3. Naturally, the constraint is tighter for low values of L. For $L = 5$ for example, there is no solution, leading to a solution loss of 1. For bigger values, the solution loss is still close to 1, with less that 1% solutions left. To see how this translates in terms of probabilities, we generated all the solutions – 29 in total – for $L = 9$, and measured their total probability, which is 1E-22. This indicates that the possibility to generate those sequences with random walk is highly unlikely.

We show an example of a text generated with this method on our illustrating corpus "*Eugene Onegin*", with a maximum order $L = 6$, on Figure 4. By construction, chunk sizes

> Blessed is he who's left the squire no time devoted to the North. After that, you're married: no look, no word to say and then began to lift his pistol in his glance, or else. The friends in all that is the one. Ah Tanya, come to know it in the fashion and in second all in fever. We must confess that all of an earlier time. Look to the circle of our first ages from thirty down to the end. He's moved. For cousins from afar, darlings, then we'll throw at him. Never. She was still helping the poor butterfly. Happy is he apparelled. Is this the man of honour and the marriage-bed, in all the play of hope? He failed to understand and took deep in gloom and mist. I beseech, and take a swill. He arrives, the girl's attentive eyes are dreaming. But to the bereaved, as if beneath her pillow, his father died. From her husband's or the unaffected thoughts of all that is the advent of the hall.

Figure 4: An example sequence with a max order $L = 6$

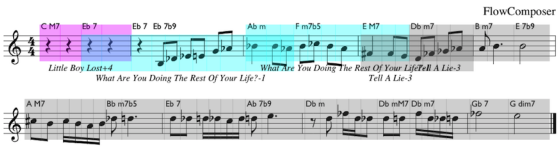


Figure 5: A lead sheet generated with a Markov order 2, in the style of Michel Legrand

are bounded by 6, and hence we do not report them on the figure. For information, 48.8% of the chunks of the sequence were of size 5, 32.5% of size 4, and 18.7% of size 3. The use of MAXORDER guarantees that no copy of size 6 or more is made from the corpus.

Throughout this paper, we used text generation as an example application, since it provides an easy to grasp illustration of our approach. However, Markov chain are often used for automatic music generation (Nierhaus 2009). We also applied our approach to the generation of lead sheets with a max order control. A lead sheet is composed of a simplified melody (one note at a time), and a sequence of chord labels.

Figure 5 shows a lead sheet generated with a Markov order 2, in the style of French composer Michel Legrand. We highlight chunks in different colours, each chunk being annotated with the name of song of the corpus where it comes from. More than half of the lead sheet is a replication of an existing sequence by the composer.

n On Figure 6, we additionally impose a max order of 6. As a result, there is no replication lasting two bars or more in the generated lead sheet.

Conclusion

We have introduced the problem of generating Markov sequences satisfying a maximum order, an important issue with Markov models that has never, to our knowledge, been addressed previously. We formulate the problem in the framework of Markov constraints. We propose an arc-consistency algorithm based on an efficient automaton construction algorithm. The approach can be used to generate

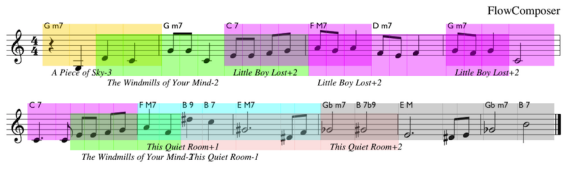


Figure 6: A lead sheet generated with an imposed max order of 6

Markov sequences with imposed maximum order on real-world corpora.

Acknowledgements

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.

References

Aho, A. V., and Corasick, M. J. 1975. Efficient string matching: An aid to bibliographic search. *Commun. ACM* 18(6):333–340.

Begleiter, R.; El-Yaniv, R.; and Yona, G. 2004. On prediction using variable order markov models. *J. Artif. Intell. Res. (JAIR)* 22:385–421.

Beldiceanu, N.; Carlsson, M.; and Petit, T. 2004. Deriving filtering algorithms from constraint checkers. In Wallace (2004), 107–122.

Brooks, F. P.; Hopkins, A.; Neumann, P. G.; and Wright, W. 1957. An experiment in musical composition. *Electronic Computers, IRE Transactions on* (3):175–182.

Karakostas, G.; Lipton, R. J.; and Viglas, A. 2000. On the complexity of intersecting finite state automata. In *IEEE Conference on Computational Complexity*, 229–234. IEEE Computer Society.

Karakostas, G.; Lipton, R. J.; and Viglas, A. 2003. On the complexity of intersecting finite state automata and n l versus n p. *Theor. Comput. Sci.* 302(1-3):257–274.

Nierhaus, G. 2009. *Algorithmic composition: paradigms of automated music generation*. Springer.

Pachet, F., and Roy, P. 2011. Markov constraints: steerable generation of markov sequences. *Constraints* 16(2):148–172.

Pachet, F.; Roy, P.; and Barbieri, G. 2011. Finite-length markov processes with constraints. In Walsh, T., ed., *IJCAI*, 635–642. IJCAI/AAAI.

Pesant, G. 2004. A Regular Language Membership Constraint for Finite Sequences of Variables. In Wallace (2004), 482–495.

Pinkerton, R. C. 1956. Information theory and melody. *Scientific American*.

Quimper, C.-G., and Walsh, T. 2006. Global grammar constraints. In Benhamou, F., ed., *CP*, volume 4204 of *Lecture Notes in Computer Science*, 751–755. Springer.

Quimper, C.-G., and Walsh, T. 2008. Decompositions of grammar constraints. In Fox, D., and Gomes, C. P., eds., *AAAI*, 1567–1570. AAAI Press.

Villeneuve, D., and Desaulniers, G. 2005. The shortest path problem with forbidden paths. *European Journal of Operational Research* 165(1):97–107.

Wallace, M., ed. 2004. *Principles and Practice of Constraint Programming - CP 2004, 10th International Conference, CP 2004, Toronto, Canada, September 27 - October 1, 2004, Proceedings*, volume 3258 of *Lecture Notes in Computer Science*. Springer.