Generating non-plagiaristic Markov sequences with *max order* Sampling

Alexandre Papadopoulos, François Pachet and Pierre Roy

Abstract Plagiarism is usually studied from an analysis viewpoint: how to detect that a text contains copies of another one. In this chapter we study plagiarism from the generation viewpoint: how to generate a text with a guarantee of non-plagiarism. More precisely, we address the problem of Markov sequence generation with forbidden k-gram constraints. This problem is addressed in two steps. In a first step, we show that, given a Markov transition matrix and a set of k-grams, we can build efficiently an automaton that represents exactly the language of all sequences that can be generated from a Markov model, and that also do not contain any of the k-grams. The size of the automaton is bounded by the size of the forbidden k-grams, and so is the time for building it. This automaton can be used to solve the algebraic problem (i.e. considering non-zero probabilities are uniform), by a simple walk. In a second step, we show that the automaton can be extended so as to be exploited by a belief propagation scheme, in order to produce perfect sampling of all the solutions.

1 Introduction

Markov chains are a powerful, widely-used technique to analyse and generate sequences that imitate a given style [4, 13], with applications to many areas of automatic content generation such as music, text and more generally sequential data. A typical use of such models is to generate *novel* sequences that "look" like or "sound" like the original.

Alexandre Papadopoulos
UPMC Paris 6, UMR 7606, LIP6, 75005, Paris, France e-mail: alexandre.papadopoulos@lip6.fr

François Pachet
Sony CSL, 6 rue Amyot, 75005 Paris, France e-mail: pachetcsl@gmail.com

Pierre Roy
Sony CSL, 6 rue Amyot, 75005 Paris, France e-mail: roypie@gmail.com

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.:

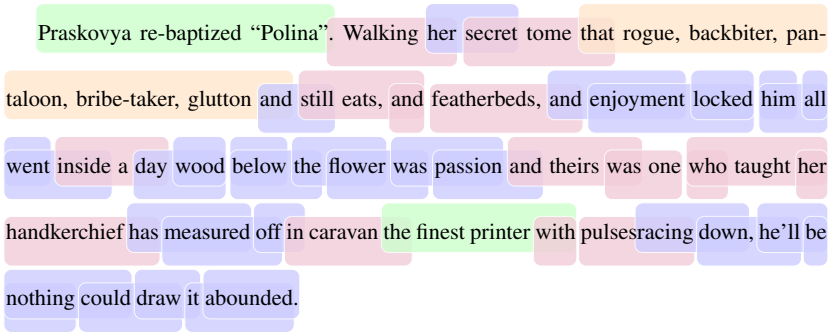
$$P(s_{i+1}|s_1,\ldots,s_i) = P(s_{i+1}|s_i).$$

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.

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

Variable order Markov models are often used to produce sequences with varying degrees of similarity with the corpus [2]: an increased Markov order produces sequences that replicate larger chunks of the original corpus, thereby improving the impression of style imitation. However, a Markov process can yield long-range effects, in spite of its local nature. The result of such effects is that undesirable substrings of a length greater than the Markov order can be generated, with a high frequency. For example, it has been long observed [4] 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. We obtained the following sequence with a Markov order of 1:



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 seven-word-long subsequence from the corpus. The maximum order of a sequence is the maximum length of its chunks (seven, in our example).

With an increased Markov order of 3, we obtained the following sequence:

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

More generally, we generated a few hundreds of sequences of size 30 based on this corpus, of varying order (from to 1 to 3): Figure 1 shows the distribution of chunk sizes observed for each Markov order. With a Markov order 2, sequences already tend to contain chunks from the corpus of length greater than 2, up to 22. 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.

To avoid this type of plagiarism, we are interested in generating Markov sequences with a guaranteed maximum order. In other words, we want to forbid all sequences of length equal to the maximum order, that appear in the corpus. But other types of undesirables sequences can occur, too. For example, anti-patterns [5] are minimal sequences (in terms of size) that do not appear in the corpus, and yet have a high probability according to the model estimated on the corpus. Although a Markov process will reproduce anti-patterns with a high probability, it can be argued that their absence in the corpus has a structural justification, and, therefore, we may want to explicitly prevent their occurrences in generated sequences. Cyclical patterns are small sequences of words repeated several times successively. Although long cyclical patterns are unlikely, short cyclical patterns have a non-trivial probability of occurring, and make no sense when generating text. If a cyclical pattern has a period longer than the Markov order, again we need another means to explicitly forbid it.

This chapter addresses precisely the problem of sampling Markov sequences that contain no forbidden substring, or *no-good* for short. Such properties cannot be guaranteed with greedy approaches like random walk. Our contribution combines techniques from constraint satisfaction, automaton theory and statistical inference. From a satisfaction viewpoint, we consider a constraint enforcing that generated sequences contain none of the imposed no-goods. Following a common approach in constraint programming [3, 12], we represent such a constraint with an automaton

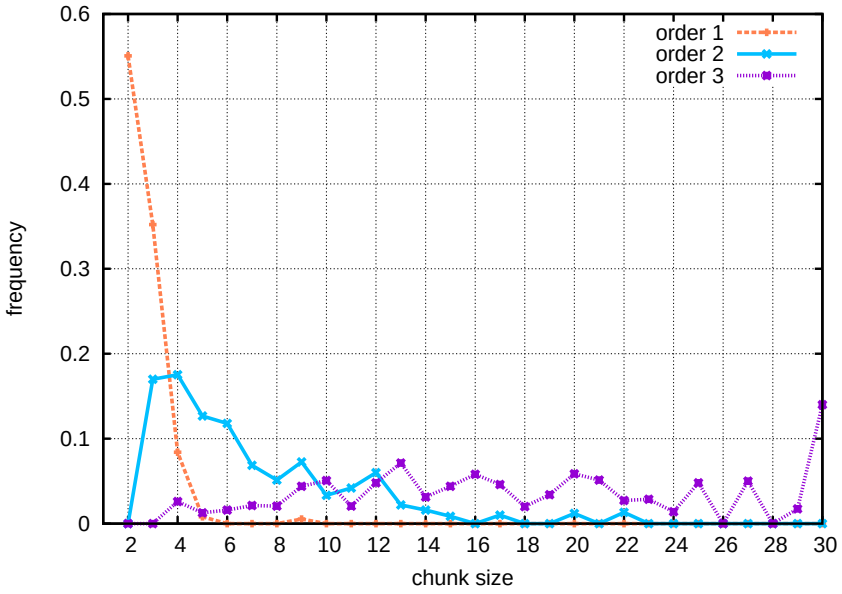


Fig. 1 Chunk size distribution for different Markov orders in a sequence of size 30

that accepts the set of such sequences. However, we show that canonical methods are not satisfactory for building this automaton. We present an algorithm that builds this automaton in linear time with respect to the set of no-goods, an aspect that has been covered in detail in previous work [9]. In a second, novel, step, this automaton is used to define a factor graph that encodes the distribution of Markov sequences containing no no-good. Belief propagation is then applied to sample such sequences in an exact manner.

2 A Simple 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.

We can encode a set of Markovian sequences, ignoring probabilities, using an *automaton*.

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, the alphabet, is a finite non-empty set of symbols, $q_0 \in Q$ is the initial state, δ is the transition function which maps a state q and a symbol a to the successor $\delta(q,a)$, and $F \subseteq Q$ is the set of final, or accepting, states.

Definition 2 (Accepted language). The word w, a string of symbols in Σ, is accepted by $\mathcal{A}$ if and only if there exists a sequences $q_0, \ldots, q_p$ of states, such that $\delta(q_{i-1}, a_i) = q_i$, for all i, $1 \leq i \leq p$, and $q_p \in F$. The set of words accepted by $\mathcal{A}$, denoted $\mathcal{L}(\mathcal{A})$, is the language accepted by $\mathcal{A}$.

In order to represent an order k Markov chain, we create an alphabet Σ where each symbol corresponds to a state of the Markov chain, a k-gram. 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 Markov chain is an automaton $\mathcal{A}$, such that in each accepted word $a_1 \ldots a_p$ in $\mathcal{L}(\mathcal{A})$, for each $i < p$, $a_i a_{i+1}$ corresponds to a non-zero probability Markov transition.

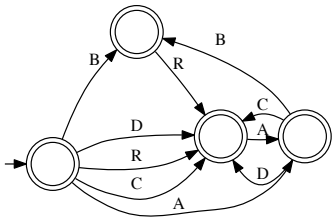


Fig. 2 A Markov automaton for the ABRACADABRA corpus, with $k = 1$

Figure 2 shows a Markov automaton for the Markov chain estimated on ABRACADABRA with $k = 1$. This automaton is essentially a syntactic rewrite of the

Markov chain, with a different semantics attached to its states and transitions. Since the automaton accepts sequences of arbitrary length, all states are accepting. 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. Furthermore, we can impose, at the expense of minimality, that all transitions pointing to a given state are labelled with the same symbol. For example, on Figure 2, all transitions labelled with R point to the same state, but transitions labelled with D and C also point to this state. In order to enforce the second invariant too, we need to make three copies of this state, for C, D and R. Satisfying those two invariants allows us to introduce the following notation.

Definition 4. We relate states and the labels of its ingoing transitions, using the following notation:

- Let a be a symbol of the alphabet, $Q(a)$ is the *unique* state q to which a-labelled transitions point.
- Let q be a state in Q, $a(q)$ is the label of the transitions pointing to q, i.e. $Q(a(q)) = q$.

Using this notation, 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. We can now represent the set of Markov sequences containing no forbidden no-good with the following automaton.

Definition 5 (No-good Automaton). Let $\mathcal{M}$ be a Markov automaton, and $\mathcal{N}$ a list of strings accepted by $\mathcal{M}$, called *no-goods*.

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 $\mathcal{L}(\mathcal{A}(a_1 \dots a_L)) = \{w \in \Sigma^* \mid a_1 \dots a_L \text{ is a substring of } w\}$, i.e. the language of words containing at least one occurrence of the no-good. An automaton $\mathcal{NG}$ is a no-good automaton for $\mathcal{M}$ and $\mathcal{N}$ if:

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

For example, in the ABRACADABRA corpus, with $k = 1$ and a max order limit of 4, we have 7 max order 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 4 property: its max order is strictly less than 4. Figure 3 shows a no-good automaton, forbidding those no-goods. Again, all states are accepting, and the labels in the states correspond to prefixes of forbidden no-goods. Note that all transitions pointing to any given state are labelled with the same symbol.

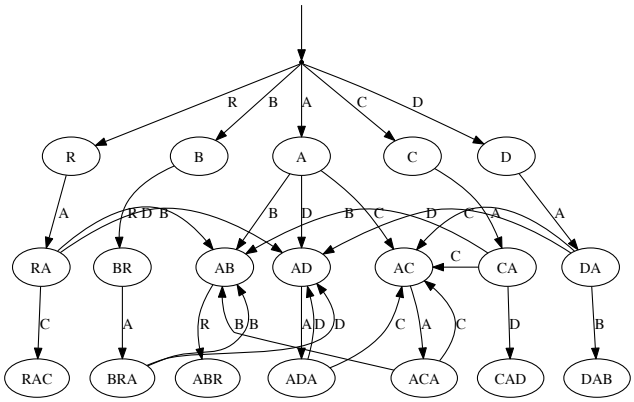


Fig. 3 A no-good automaton for the ABRACADABRA corpus, with $k = 1$ and maximum order less than 4

3 Building the No-good Automaton

The no-good automaton can be built in a generic fashion, using standard automata theory operations that implement Definition 5. Initially, we build a Markov automaton. Then, for each no-good, this automaton is intersected with the negation of the automaton recognising sequences containing the no-good. The complexity of this procedure is dominated by the complexity of intersecting a number of automata. A straightforward intersection algorithm runs in $O(t^N)$ time, with t the maximum size, in number of states, of any of the automata, and $N = |\mathcal{N}|$ the number of no-goods. And it is unlikely that an algorithm with a better complexity exists [7, 8]. Therefore, we cannot tractably compute the no-good automaton in a generic way, without exploiting its particular structure. Furthermore, this method does not give any bound on the size of the final automaton (other than $O(t^N)$).

We present, in this section, an algorithm that builds the no-good automaton in time linear in the size of the set of no-goods. As a corollary, we show that the size of this automaton is linear too. More precisely, this algorithm takes as input a Markov automaton and a list of no-goods, and operates as follows: build a trie with the no-goods, compute their overlaps, and finally use this structure to remove from the Markov automaton all sequences containing a no-good.

Algorithm 1 first computes a trie of the no-goods, where all states but the ones corresponding to a full no-good are accepting states. This ensures that a no-good is never accepted. This trie is connected to the original Markov automaton by disconnecting from the Markov automaton any transition that starts a no-good, and use those transitions to start building the trie. However, this is not sufficient. The key part of the algorithm is to add connections between overlapping no-good prefixes. For example, if we have two no-goods ABCD and BCEF, the prefixes ABC and BCEF overlap on BC. This means that the automaton not only should not accept ABCD, but it should not accept ABCEF either. This connection is made using

cross-prefix transitions: we add a cross-prefix transition, labelled with E, between the state for ABC and the state for BCE. Cross-prefix transitions ensure that, by avoiding a particular no-good, we will not inadvertently complete another no-good. In order to compute them, Algorithm 1 uses an adaptation of the Aho and Corasick [1] string-matching algorithm: a transition that does not extend the no-good of the current no-good prefix is a cross-prefix transition, and points directly to the no-good that starts with the longest 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. Now, we can finally remove those non-accepting states (line 33). It is important to observe that all the steps we described maintain the invariant that for each state of the automaton being built, the label of all its incoming transitions is always the same. This property will be exploited for sampling the sequences recognised by the automaton.

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 $N = L.|\mathcal{N}|$, where L is the size of a no-good, be the 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 [1]. Finally, the number of transitions added to each state during the completion phase is bounded by $|\Sigma|$, which is a constant independent of $\mathcal{N}$.

Note that the general idea of this algorithm is similar to the algorithm by [14], which computes shortest paths with forbidden paths. However, they operate in a very different context, and are only interested in the shortest paths of a graph, whereas we are interested in all the sequences accepted by the original Markov automaton.

4 Sampling Sequences with a Maximum Order Guarantee

We now describe how to sample Markov sequences containing no forbidden no-good, with its correct probability, using the no-good automaton computed in the previous section. As a first approximation, one can use the no-good automaton to generate new sequences, since all walks in this automaton produce valid sequences. The transition probabilities of the original Markov model are used for walking in this automaton, by choosing the successor of a state q as follows: with $a(q)$ the label of all transitions pointing to q, choose a transition labelled with b with probability $P(b|a(q))$. We show in the next section that this produces a crude approximation of the probability of sampled sequences. However, if we impose a given sequence length, we can use *belief propagation* and do perfect sampling, by exploiting a special case where belief propagation is, by principle, exact and polynomial.

Algorithm 1: Computing the no-good 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
```
forall the $a_1 \dots a_L \in \mathcal{N}$ **do**
  $q \leftarrow Q(a_1)$
  Clear outgoing transitions of q
  $w(q) \leftarrow (a_1)$

```
// Compute the trie of no-goods
```
$Q_{trie} \leftarrow \emptyset$
forall the $a_1 \dots a_L \in \mathcal{N}$ **do**
  $q \leftarrow q_0$
  $i \leftarrow 1$
  while $\delta(q, a_i)$ *exists* **do**
    $q \leftarrow \delta(q, a_i)$
    $i \leftarrow i + 1$
  for $a_j, i \leq j \leq L$ **do**
    $q' \leftarrow NewState(Q_{trie})$
    $F \leftarrow F \cup \{q'\}$
    $\delta(q, a_j) \leftarrow q'$
    $w(q') \leftarrow (a_1, \dots, a_j)$
    $a(q') \leftarrow \{a_j\}$
    $q \leftarrow q'$
  $F \leftarrow F \setminus \{q\}$

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

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

Belief propagation [11] is an algorithm, or family of algorithms, for performing statistical inference on graphical models, i.e. graph-based representations of distributions over multiple variables. Specifically, we consider graphical models called *factor graphs*. A factor graph is a bipartite undirected graph, representing the factorisation of a probability function, where nodes represent either variables or factors, and edges connect factors to the variables to which that factor applies. The sum-product algorithm allows us to perform statistical inference, such as marginalising or sampling the probability function.

4.1 Background on Belief Propagation

Suppose we have n random variables $X_1,\ldots,X_n$, and a real-valued function $g(X_1,\ldots,X_n)$ that can be expressed as the product of m factors:

$$g(X_1,\ldots,X_n) = \prod_{j=1}^{m} f_j(S_j),$$

where the factor f_j is a function holding only on a subset $S_j \subseteq \{X_1,\ldots,X_n\}$ of the variables. The corresponding factor graph is a bipartite graph $G = (X,F,E)$, where $X = \{X_1,\ldots,X_n\}$, $F = \{f_1,\ldots,f_m\}$, and an edge (X_i,f_j) is in E iff $X_i \in S_j$.

Example 1. Consider a function holding on three variables X_1,X_2,X_3, defined as the product of four factors:

$$g(X_1,X_2,X_3) = f_1(X_1,X_2)\cdot f_2(X_2,X_3)\cdot f_3(X_1,X_3)\cdot f_4(X_3)$$

Figure 4 shows the corresponding factor graph.

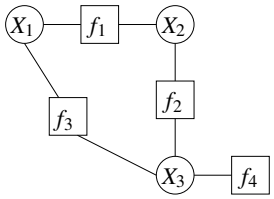


Fig. 4 The factor graph for the function $g(X_1,X_2,X_3) = f_1(X_1,X_2)\cdot f_2(X_2,X_3)\cdot f_3(X_1,X_3)\cdot f_4(X_3)$.

In general, functions that display a tree-structured factor graph have interesting properties, since several statistical properties can be computed in an exact and yet tractable way, by exploiting the independence between subsets of variables. To illustrate this on our previous example, suppose factor f_3 is suppressed, i.e. $g(X_1,X_2,X_3) = f_1(X_1,X_2)\cdot f_2(X_2,X_3)\cdot f_4(X_3)$. The resulting factor graph thus becomes a tree. Suppose we want to compute the normalisation constant Z such

that $P(X_1,X_2,X_3) = \frac{1}{Z} \cdot g(X_1,X_2,X_3)$ is a probability function. In general, $Z = \sum_{X_1} \sum_{X_2} \sum_{X_3} g(X_1,X_2,X_3)$, implying a summation over the whole cartesian product of the domains of the variables of g, which is exponential in the number of variables. However, by exploiting the tree structure of the factor graph, we can first compute $\sum_{X_3} f_2(X_2,X_3) \cdot f_4(X_3)$, which results in a function $f_{X_2}(X_2)$ on X_2 only, and then we can compute the sum $\sum_{X_1,X_2} f_1(X_1,X_2) \cdot f_{X_2}(X_2)$. In other words, we only perform local summations at a time, on the cartesian product of the variables of a given factor, and never on the whole set of variables.

The sum-product algorithm, first invented in 1982 by Pearl [10], is an algorithm that computes the marginal function of a particular node, i.e. the function $g(X_i) = \sum_{X_1,\ldots,X_{i-1},X_{i+1},\ldots,X_n} g(X_1,\ldots,X_n)$ for some X_i, using the same strategy. Likewise, if the factor graph is a tree, it gives the exact result in tractable time. This algorithm is often referred to as message passing. In turn, this can be used to sample valuations of $X_1,\ldots,X_n$ with their right probability. One needs to draw values for all X_i, from X_1 to X_n, choosing each time a value for X_i according to its marginal probability $g(X_i)$.

4.2 A factor graph model of max order sequences

We now apply those techniques to our application, and describe message passing, by instantiating this procedure to the specific problem of sampling Markov sequences containing no forbidden no-good of size L. The problem of generating a Markov sequence of n variables $X_1,\ldots,X_n$ containing no forbidden no-good, is the problem of sampling the function g_{maxo} defined as:

$$g_{maxo}(X_1,\ldots,X_n) = \begin{cases} 0 & \text{if } X_1,\ldots,X_n \text{ contains a no-good,} \\ \frac{1}{Z} \cdot P(X_1) \cdots P(X_n|X_{n-1}) & \text{otherwise} \end{cases}$$

This function g_{maxo} can be represented in a straightforward way with a factor graph, such as the one shown on Figure 5, for the case where $k = 1$, $L = 3$ and $n = 5$. We have $f_1(X_1) = \frac{1}{Z} \cdot P(X_1)$, $f_i(X_{i-1},X_i) = P(X_i|X_{i-1})$, and $m_i(X_i,\ldots,X_{i+L-1}) = 0$, if $X_i,\ldots,X_{i+L-1}$ is a no-good of size L, and $m_i(X_i,\ldots,X_{i+L-1}) = 1$ otherwise.

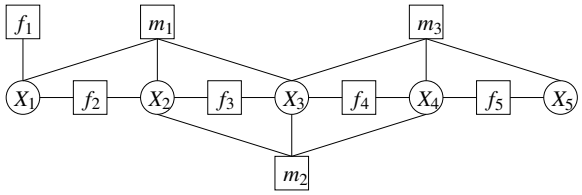


Fig. 5 The basic factor graph for max order sequences, with a Markov order $k = 1$, a maximum order $L = 3$ and a length of $n = 5$

As it stands, this factor graph is not acyclic. In order to render it acyclic, one can apply a naive, "brute force", reformulation: we merge every subsequence of $L-1$ consecutive variables into one single merged variable. As a result, we have only binary factors between every two consecutive merged variables, as shown on Figure 6. When those variables are assigned two overlapping $(L-1)$-grams that combine into an L-gram, and that this L-gram is not a no-good, the binary factor gives the Markov probability of the L-gram; otherwise it evaluates to 0. However, this would not be tractable in practice, since the alphabets of the merged variables contain all combinations of $L-1$ elements of the original alphabet, and this is exponential in the max order L.

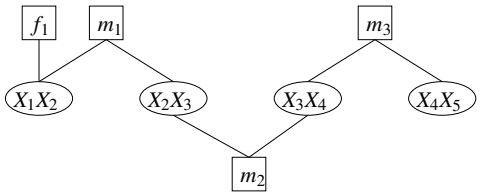


Fig. 6 The – inefficient – acyclic reformulation of the factor graph of Figure 5

We propose another reformulation, which exploits the no-good automaton. We define a function $g(X_1,\ldots,X_n)$, where the domain of each variable X_i is the set of states of the no-good automaton, and not the states of the original Markov model. Recall that the states of the original Markov chain correspond to the set Σ of labels of the no-good automaton. This function g is composed of simple binary factors, and its factor graph decomposition is shown on Figure 7.

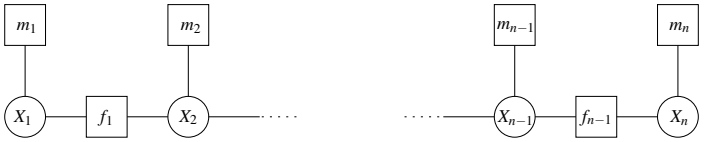


Fig. 7 The factor graph for max order using the no-good automaton

In order to define the factors involved in this factor graph, we exploit the fact that, for each state q of the automaton, the label of all incoming transitions is the same, denoted $a(q)$. As we established earlier, this is satisfied both for the Markov automaton and for the no-good automaton produced by our algorithm. The same binary factor is applied along the sequence, i.e. $\forall i \leq n, f_i = f$, with f defined as follows:

$$f(q,q') = \begin{cases} P(a(q')|a(q)), & \text{if } q' \in \delta^+(q) \text{ and } q \neq q_0, \\ 0 & \text{otherwise} \end{cases}$$

This binary factor imposes that sequences are formed by walking the automaton, with a probability given by the Markov probability. The unary factors impose that sequences are formed by traversing the automaton from the initial state to an accepting state, and can additionally impose a bias on the element at a given position of a solution. Suppose $P_i(X_i)$ is a probability distribution on Σ, biasing the probability of the Markov state occurring at position i, such that the probability of a sequence $X_1,\ldots,X_n$ is the biased Markov probability $P(X_2|X_1).\cdots.P(X_n|X_{n-1}).P_1(X_1).\cdots.P_n(X_n)$. Typically, $P_1(X_1)$ imposes the prior probability on X_1. Hard unary constraints, where some values are forbidden, and the remaining ones have a uniform probability, can also be imposed under this formalism. The unary factors of our model are thus defined as follows:

$$m_i(q) = \begin{cases} 0, & \text{if } q = q_0, \\ P_i(a(q)), & \text{otherwise} \end{cases}$$

To this general definition, we need to add special cases for $i = 1$ and $i = n$. For $i = 1$ we introduce to the equation the additional case $m_1(q) = 0$ if $q \notin \delta^+(q_0)$, stating that we can only start a sequence from a successor of the initial state of the automaton. For $i = n$, we introduce the case $m_n(q) = 0$ if $q \notin F$, stating that we can only end a sequence on an accepting state of the automaton.

It is easy to verity that $g(X_1,\ldots,X_n) = g_{maxo}(a(X_1),\ldots,a(X_n))$, if we do not take into account the unary factors $m_i, i > 1$. Additionally, since each sequence of Markov states correspond to a unique sequence of states in the automaton from the initial state to an accepting state, each sequence of Markov states corresponds to a unique sequence of states $X_1,\ldots,X_n$ with a non-zero g probability. As a result, sampling g_{maxo} is equivalent to sampling g. Note that if we incorporate the additional unary factors $m_i, i > 1$ into the definition of g_{maxo} too, the exact same reasoning holds.

4.3 Sampling Max Order Sequences

Algorithm 2 is a description of the sum-product algorithm applied specifically to our factor graph, and specialised for sampling sequences. The sum-product algorithm is used to compute marginalisations, which in turn is used to sample solutions with their correct probability, in the following way. Let $g_{X_1}(X_1)$ be the marginalisation of g on X_1, equal to $\sum_{X_i,i>1} g(X_1,\ldots,X_n)$. Intuitively, for a given value q_1 of X_1, $g_{X_1}(q_1)$ gives us the probability that $X_1 = q_1$ in a sequence of values of $X_1,\ldots,X_n$, drawn randomly according to this sequence's probability. Suppose we have drawn $X_1 = q_1$ with this probability, $g(q_1,X_2,\ldots,X_n)$ is now a function of only $X_2,\ldots,X_n$. If we marginalise this function on X_2, we get a function $g_{X_2}(X_2) = \sum_{X_i,i>2} g(q_1,X_2,\ldots,X_n)$, giving the probability $g_{X_2}(q_2)$ that $X_2 = q_2$ in a sequence of values of $X_1,\ldots,X_n$ starting with $X_1 = q_1$, drawn randomly according to its g probability. After repeating this process until all X_i have been assigned, we end up with a sequence of values $q_1,\ldots,q_n$, drawn with the probability that

$X_1 = q_1$ and $X_2 = q_2$ and ... and $X_n = q_n$, which is exactly the probability of the valuation $q_1, \ldots, q_n$ in the distribution g.

On the specific case where the factor graph is a tree, as in the model of Figure 7, the sum-product algorithm performs in two phases only. A first phase from the leaf to the root (for any arbitrarily chosen root) of the tree, and a second from the root to the leaf. In our case, it is convenient to traverse the graph from X_n down to X_1 (backward phase), and from X_1 back to X_n (forward phase).

During the *backward phase*, at the i-th iteration, the algorithm considers only the part of the factor graph from X_i to X_n, i.e. the function $g_{i\leftarrow}(X_i, \ldots, X_n) = m_i(X_i).f_i(X_i, X_{i+1}).\cdots.f_{n-1}(X_{n-1}, X_n).m_n(X_n)$. It computes a *message* on X_i, which is simply a probability distribution $g_{i\leftarrow}(X_i)$ on X_i, equal to the marginalisation on X_i of $g_{i\leftarrow}(X_i, \ldots, X_n)$. At the end of the backward phase, the message on X_1 is the normalisation of the full function g on X_1, i.e. $g_{1\leftarrow}(X_1) = g_{X_1}(X_1)$.

At the beginning of the *forward phase*, the algorithm draws a value for X_1 according to $g_{X_1}(X_1)$. Then, at the i-th iteration of the forward phase, the algorithm has already instantiated $X_1, \ldots, X_{i-1}$ to $q_1, \ldots, q_{i-1}$. It considers the part of the factor graph from X_1 to X_i, i.e. the function $g_{i\rightarrow}(X_1, \ldots, X_i) = m_1(X_1).f_1(X_1, X_2).\cdots.f_{i-1}(X_{i-1}, X_i)$. The message $g_{i\rightarrow}(X_i)$ on X_i is essentially the marginalisation of $g_{i\rightarrow}(q_1, \ldots, q_{i-1}, X_i)$. Since the product $g_{i\rightarrow}(X_1, \ldots, X_i).g_{i\leftarrow}(X_i, \ldots, X_n)$ is equal to $g(X_1, \ldots, X_n)$, for a value q_i of X_i, the product $g_{i\rightarrow}(q_i).g_{i\leftarrow}(q_i)$ is precisely the probability $g_{X_i}(q_i)$, i.e. the probability that $X_i = q_i$ in a sequence on $X_1, \ldots, X_n$ starting with $q_1, \ldots, q_{i-1}$, drawn randomly with probability given by g. Therefore, the algorithm draws q_i with probability $g_{i\rightarrow}(q_i).g_{i\leftarrow}(q_i)$, and continues up to iteration n, when all variables are instantiated. Note that when sampling several sequences, the backward phase needs to be performed only once, and the forward phase will sample a new sequence every time with its correct probability.

Algorithm 2: Sum-product algorithm for max order sampling

Data: Function $g(X_1, \ldots, X_n)$ and its factor graph
Result: A sequence $q_1, \ldots, q_n$, with probability $g(q_1, \ldots, q_n)$

```
// Backward phase
```
$\mathbf{g_{n\leftarrow}}(X_n) \leftarrow m_n(X_n)$
for $i \leftarrow n-1$ *to 1* **do**
  foreach $q \in Q$ **do**
    $\mathbf{g_{i\leftarrow}}(q) \leftarrow \sum_{q' \in Q} m_i(q).f_i(q, q').\mathbf{g_{i+1\leftarrow}}(q')$
  Renormalise $\mathbf{g_{i\leftarrow}}$
```
// Forward phase
```
$\mathbf{q_1} \leftarrow draw(\mathbf{g_{1\leftarrow}}(X_1))$
for $i \leftarrow 2$ *to* n **do**
  foreach $q \in Q$ **do**
    $\mathbf{g_{i\rightarrow}}(q) \leftarrow f_{i-1}(\mathbf{q_{i-1}}, q)$
  Renormalise $\mathbf{g_{i\rightarrow}}$
  $\mathbf{q_i} \leftarrow draw(\mathbf{g_{i\rightarrow}}(X_i).\mathbf{g_{i\leftarrow}}(X_i))$
return $(\mathbf{q_1}, \ldots, \mathbf{q_n})$

5 Evaluation

Following our opening example, we applied our algorithm on the "Eugene Onegin" corpus (2160 sentences, 6919 unique words, 32719 words in total). On this corpus, depending on the Markov order (ranging from 1 to 3) and the maximum order parameters (ranging from 3 to 20), which both affect the number and the size of no-goods, computing the no-good automaton takes consistently around 200ms and never more than 300ms.

5.1 Solution loss

An interesting question is how likely a stochastic method finds sequences satisfying the maximum order property. In order to assess this, we try to estimate the probability that a Markov sequence contain no no-good. We perform this estimation by making simple solution counting: we count the number of sequences of a given length with a non-zero probability, with and without a max order constraint, and consider the ratio of the two values. Under the assumption that valid max order sequences follow a distribution similar to that of all Markov sequences, this ratio should be a good estimation of the probability of satisfying the imposed max order. 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. This is obtained by raising to the power of 20 the transition matrix where non-zero entries are replaced with 1. 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 20 (the maximum value for which max order is not trivially satisfied). This is obtained by raising to the power of 20 the adjacency matrix of the no-good automaton (i.e. the matrix where 1-entries correspond to pairs of linked states). 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 8. 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.

Note that a formal combinatorics approach for characterising this ratio has been given by Guibas and Odlyzko [6], where they define a generating function for the number of sequences of a given length not containing any forbidden pattern from a list of forbidden patterns. This generating function has a closed form, which can be used, in principle, to estimate the number of such sequences using partial fraction decompositions. However, it is much more practical, and in principle equivalent, to estimate this number based on the no-good automaton, as we did here.

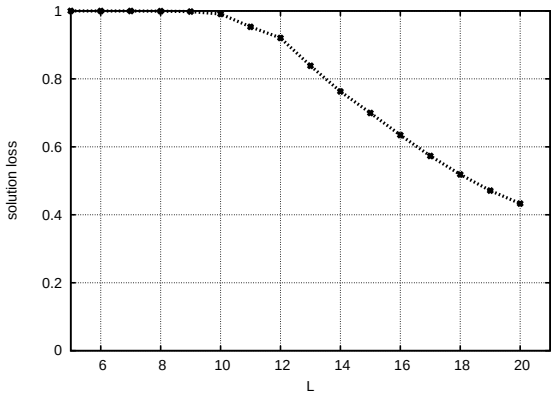


Fig. 8 Solution loss from MAXORDER, on "Eugene Onegin" with $k = 3$, $n = 20$, for various values of L

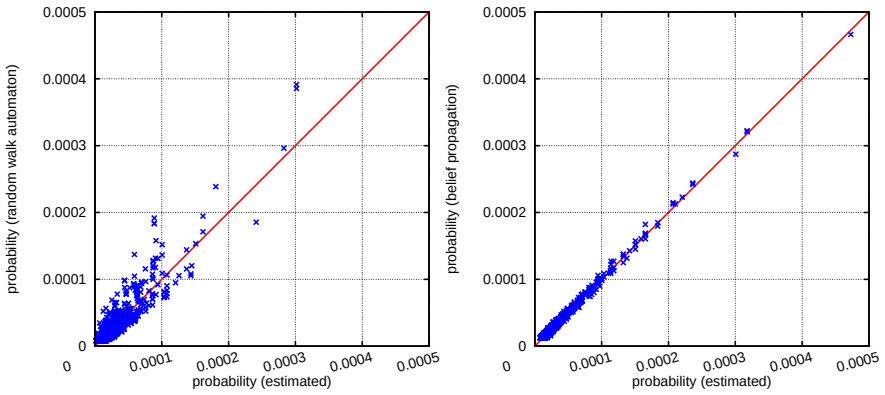
5.2 Sampling

We compare the probability of generating a sequence by the two sampling methods mentioned in Section 4: a random walk in the no-good automaton, and our fixed-length belief propagation model. The purpose of this experiment is twofold: show that a random walk in the automaton does not sample sequences correctly, and confirm empirically that our belief propagation-based model is indeed correct.

We applied each method to generate sequences of length 8, of Markov order 1 and with an imposed max order of 4. For the random walk method, we imposed the length simply by rejecting shorter sequences. In total, we sampled over 20 million sequences. Of those, 5 million were unique sequences. Concerning running times, the baseline random walk-based procedure generated an average of 5500 sequences per second (counting only non-rejected sequences), while the exact belief propagation-based method generated an average of 3500 sequences per second. To measure empirical probabilities more accurately, we filtered only those that were generated over 50 times, of which there were about 47000 with random walk, about 35000 with belief propagation (because of the slight time penalty for belief propagation). We used a simple method to estimate the correct probability of a max order sequence, by computing Z, equal to the sum of the probability of all unique sequences found by either method, and use $1/Z$ as the normalising constant, i.e. the probability of a max order sequence is equal to its Markov probability divided by Z. With the high number of sequences that we generated, this gave a reasonably accurate estimation.

We plot our results on Figure 9. Each point on either graph corresponds to a sequence. Its value on the x-axis is its actual probability, estimated as described previously, while the values on the y-axis is the empirical probability, i.e. the frequency at which the specific sequence has been sampled compared to the total number of sequences. Figure 9(a) shows that the baseline sampling approach performs rather

poorly: many sequences, even of similar probability, are over or under-represented. On the other hand, Figure 9(b) provides a striking empirical confirmation of the correctness of the belief propagation model.



(a) Random walk in the automaton (b) Belief propagation

Fig. 9 Sampling with random walk in the automaton compared to belief propagation. A point corresponds to a sequence, having a certain probability (x-axis), sampled with a certain frequency (y-axis).

5.3 Example

We conclude this section by showing an example of a text generated with this method, with a Markov order 2 and a maximum order 6:

> 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 .

By construction, chunk sizes are bounded by 6. For information, about 49% of the chunks of the sequence were of size 5, 32% of size 4, and 19% of size 3. The max order guarantee implies that no copy of size 6 or more is made from the corpus.

6 Conclusion

We have introduced the problem of generating Markov sequences satisfying a maximum order, or more generally, containing no forbidden sequence (no-good), an important issue with Markov models that has not, to our knowledge, been addressed previously. In a first step, we formulate the problem in the framework of automata theory, and exhibit an automaton that solves the algebraic problem of generating Markov sequences with no no-goods. In a second step, we extend the automaton and associate it with a linear factor graph to achieve perfect sampling of these sequences, thereby closing the issue. The set of no-goods can be arbitrary (finite) so as to encode any unwanted set of subsequences (not only coming from the corpus). Interestingly, the inverse problem (*plagiaristic generation*) consisting in guaranteeing that at least one no-good is present in each generated sequence is not of the same nature. This is work in progress.

References

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

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

[3] Beldiceanu, N., Carlsson, M., Petit, T.: Deriving filtering algorithms from constraint checkers. In: [15], pp. 107–122

[4] Brooks, F.P., Hopkins, A., Neumann, P.G., Wright, W.: An experiment in musical composition. Electronic Computers, IRE Transactions on **6**(3), 175–182 (1957)

[5] Conklin, D., Weisser, S.: Antipattern discovery in ethiopian bagana songs. In: Dzeroski, S., Panov, P., Kocev, D., Todorovski, L. (eds.) Discovery Science - 17th International Conference, DS 2014, Bled, Slovenia, October 8-10, 2014. Proceedings, *Lecture Notes in Computer Science*, vol. 8777, pp. 62–72. Springer (2014)

[6] Guibas, L.J., Odlyzko, A.M.: String overlaps, pattern matching, and nontransitive games. J. Comb. Theory, Ser. A **30**(2), 183–208 (1981). DOI 10.1016/0097-3165(81)90005-4

[7] Karakostas, G., Lipton, R.J., Viglas, A.: On the complexity of intersecting finite state automata. In: IEEE Conference on Computational Complexity, pp. 229–234. IEEE Computer Society (2000)

[8] Karakostas, G., Lipton, R.J., Viglas, A.: On the complexity of intersecting finite state automata and NL versus NP. Theor. Comput. Sci. **302**(1-3), 257–274 (2003)
[9] Papadopoulos, A., Roy, P., Pachet, F.: Avoiding Plagiarism in Markov Sequence Generation. In: Brodley, C.E., Stone, P. (eds.) AAAI. AAAI Press (2014)
[10] Pearl, J.: Reverend bayes on inference engines: A distributed hierarchical approach. In: Waltz, D.L. (ed.) Proceedings of the National Conference on Artificial Intelligence. Pittsburgh, PA, August 18-20, 1982., pp. 133–136. AAAI Press (1982)
[11] Pearl, J.: Probabilistic reasoning in intelligent systems - networks of plausible inference. Morgan Kaufmann series in representation and reasoning. Morgan Kaufmann (1989)
[12] Pesant, G.: A Regular Language Membership Constraint for Finite Sequences of Variables. In: [15], pp. 482–495
[13] Pinkerton, R.C.: Information theory and melody. Scientific American (1956)
[14] Villeneuve, D., Desaulniers, G.: The shortest path problem with forbidden paths. European Journal of Operational Research **165**(1), 97–107 (2005)
[15] Wallace, M. (ed.): Principles and Practice of Constraint Programming - CP 2004, 10th International Conference, CP 2004, Toronto, Canada, September 27 - October 1, 2004, Proceedings, *Lecture Notes in Computer Science*, vol. 3258. Springer (2004)