

Article

## Algorithms for Hidden Markov Models Restricted to Occurrences of Regular Expressions

Paula Tataru <sup>1</sup>, Andreas Sand <sup>1,2</sup>, Asger Hobolth <sup>1</sup>, Thomas Mailund <sup>1</sup> and Christian N. S. Pedersen <sup>1,2,\*</sup>

<sup>1</sup> Bioinformatics Research Centre, Aarhus University, C. F. Møllers Allé 8, DK-8000 Aarhus C, Denmark; E-Mails: paula@birc.au.dk (P.T.); asand@birc.au.dk (A.S.); hobolth@birc.au.dk (A.H.); mailund@birc.au.dk (T.M.)

<sup>2</sup> Department of Computer Science, Aarhus University, Aabogade 34, DK-8200 Aarhus N, Denmark

\* Author to whom correspondence should be addressed; E-Mail: cstorm@birc.au.dk; Tel.: +45-871-55559; Fax: +45-871-54102.

Received: 28 June 2013; in revised form: 8 October 2013 / Accepted: 5 November 2013 /

Published: 8 November 2013

---

**Abstract:** Hidden Markov Models (HMMs) are widely used probabilistic models, particularly for annotating sequential data with an underlying hidden structure. Patterns in the annotation are often more relevant to study than the hidden structure itself. A typical HMM analysis consists of annotating the observed data using a decoding algorithm and analyzing the annotation to study patterns of interest. For example, given an HMM modeling genes in DNA sequences, the focus is on occurrences of genes in the annotation. In this paper, we define a pattern through a regular expression and present a restriction of three classical algorithms to take the number of occurrences of the pattern in the hidden sequence into account. We present a new algorithm to compute the distribution of the number of pattern occurrences, and we extend the two most widely used existing decoding algorithms to employ information from this distribution. We show experimentally that the expectation of the distribution of the number of pattern occurrences gives a highly accurate estimate, while the typical procedure can be biased in the sense that the identified number of pattern occurrences does not correspond to the true number. We furthermore show that using this distribution in the decoding algorithms improves the predictive power of the model.

**Keywords:** Hidden Markov Model; decoding; Viterbi; forward; algorithm

---

## 1. Introduction

A Hidden Markov Model (HMM) is a probabilistic model for sequential data with an underlying hidden structure. Because of their computational and analytical tractability, they are widely used especially in speech recognition [1–3], image processing [4] and in several applications in bioinformatics; e.g., modeling of proteins [5–7], sequence alignment [8–10], phylogenetic analysis [11,12] and identification of coding regions in genomes [13,14].

Patterns in the hidden structure are, however, often more relevant to study than the full hidden structure itself. When modeling proteins, one might be interested in neighboring secondary structures that differ, while for sequence alignments, the pattern could capture specific characteristics, such as long indels. In phylogenetic analysis, changes in the tree along the sequence are most relevant, while when investigating coding regions of DNA data, patterns corresponding to genes are the main focus.

Counting the number of occurrences of such patterns can be approached (as in the methods based on [15]) by making inferences from the prediction of a decoding algorithm; e.g., the Viterbi algorithm or the posterior-Viterbi algorithm. As we show in our experiments, this can give consistently biased estimates for the number of pattern occurrences. A more realistic method is presented in [16], where the distribution of the number of pattern occurrences is computed by means of Markov chain embedding. To our knowledge, this is the only study of patterns in the hidden sequence of HMMs. The problem of pattern finding in random sequences generated by simple models, such as Markov chains, has been intensively studied using the embedding technique [17–19].

We present a fundamentally different approach to compute the distribution of the number of pattern occurrences and show how it can be used to improve the prediction of the hidden structure. We use regular expressions as patterns and employ their deterministic finite automata to keep track of occurrences. The use of automata to describe occurrences of patterns in Markov sequences has been described previously in [18,20,21]. However, analyzing pattern occurrences in the hidden structure of HMMs by means of automata has not been done before. We introduce a new version of the forward algorithm, the restricted forward algorithm, which computes the likelihood of the data under the hidden sequence containing a specific number of pattern occurrences. This algorithm can be used to compute the occurrence number distribution. We furthermore introduce new versions of the two most widely used decoding algorithms, the Viterbi algorithm and the posterior-Viterbi algorithm, where the prediction is restricted to containing a certain number of occurrences, e.g., the expectation obtained from the distribution. We have implemented and tested the new algorithms and performed experiments that show that this approach improves both the estimate of the number of pattern occurrences and the prediction of the hidden structure.

The remainder of this paper is organized as follows: we start by introducing Hidden Markov Models and automata; we continue by presenting our restricted algorithms, which we then validate experimentally.

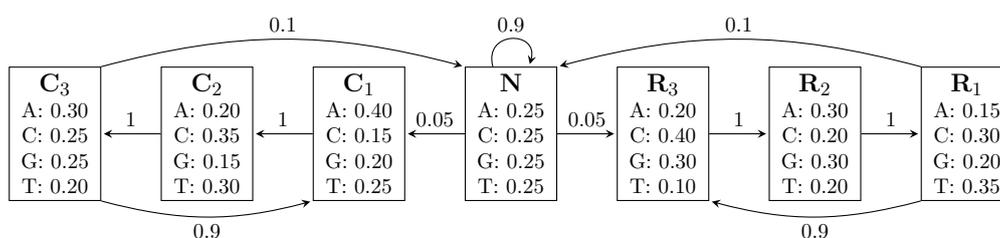
## 2. Methods

### 2.1. Hidden Markov Models

A Hidden Markov Model (HMM) [3] is a joint probability distribution over an observed sequence  $y_{1:T} = y_1y_2 \dots y_T \in \mathcal{O}^*$  and a hidden sequence  $x_{1:T} = x_1x_2 \dots x_T \in \mathcal{H}^*$ , where  $\mathcal{O}$  and  $\mathcal{H}$  are finite alphabets of observables and hidden states, respectively. The hidden sequence is a realization of a Markov process that explains the hidden properties of the observed data. We can formally define an HMM as consisting of a finite alphabet of hidden states  $\mathcal{H} = \{h_1, h_2, \dots, h_N\}$ , a finite alphabet of observables  $\mathcal{O} = \{o_1, o_2, \dots, o_M\}$ , a vector  $\Pi = (\pi_{h_i})_{1 \leq i \leq N}$ , where  $\pi_{h_i} = \mathbb{P}(X_1 = h_i)$  is the probability of the hidden sequence starting in state  $h_i$ , a matrix  $A = \{a_{h_i, h_j}\}_{1 \leq i, j \leq N}$ , where  $a_{h_i, h_j} = \mathbb{P}(X_t = h_j \mid X_{t-1} = h_i)$  is the probability of a transition from state  $h_i$  to state  $h_j$ , and a matrix  $B = \{b_{h_i, o_j}\}_{1 \leq i \leq N, 1 \leq j \leq M}$ , where  $b_{h_i, o_j} = \mathbb{P}(Y_t = o_j \mid X_t = h_i)$  is the probability of state  $h_i$  emitting  $o_j$ .

Figure 1 shows an HMM designed for gene finding. The observed sequence is a DNA sequence over the alphabet  $\mathcal{O} = \{A, C, G, T\}$ , while the hidden states encode if a position is in a non-coding area ( $N$ ) or is part of a gene on the positive strand ( $C$ ) or on the negative strand ( $R$ ). The model incorporates the fact that nucleotides come in multiples of three within genes, where each nucleotide triplet codes for an amino acid. The set of hidden states is  $\mathcal{H} = \{N\} \cup \{C_i, R_i \mid 1 \leq i \leq 3\}$ . In practice, models used for gene finding are much more complex, but this model captures the essential aspects of a gene finder.

**Figure 1.** A Hidden Markov Model (HMM) for gene prediction. Each box represents a hidden state, and the numbers inside are the emission probabilities of each nucleotide. Numbers on arcs are transition probabilities between hidden states.



HMMs can be used to generate sequences of observables, but their main application is for analyzing an observed sequence,  $y_{1:T}$ . The likelihood of a given observed sequence can be computed using the forward algorithm [3], while the Viterbi algorithm [3] and the posterior-Viterbi algorithm [22] are used for predicting a corresponding hidden sequence. All these algorithms run in  $\mathcal{O}(TN^2)$  time, using  $\mathcal{O}(TN)$  space.

#### 2.1.1. The Forward Algorithm

The forward algorithm [3] finds the probability of observing  $y_{1:T}$  by summing the joint probability of the observed and hidden sequences for all possible sequences,  $x_{1:T}$ . This is given by:

$$\mathbb{P}(y_{1:T}, x_{1:T}) = \pi_{x_1} b_{x_1, y_1} \prod_{t=2}^T a_{x_{t-1}, x_t} b_{x_t, y_t} \tag{1}$$

$$\mathbb{P}(y_{1:T}) = \sum_{x_{1:T}} \mathbb{P}(y_{1:T}, x_{1:T}) \quad (2)$$

where Equation (1) is the multiplication of the probabilities of transitions and emissions, which explain observing  $y_{1:T}$  with  $x_{1:T}$  as the hidden sequence: the HMM starts in state  $x_1$  and emits  $y_1$  from  $x_1$ , and for all  $t = 2, \dots, T$ , it makes a transition from state  $x_{t-1}$  to  $x_t$  and emits  $y_t$  from  $x_t$ .

The forward algorithm finds Equation (2) by recursively filling up a table,  $\alpha$ , with values  $\alpha_t(x_t) = \mathbb{P}(y_{1:t}, x_t) = \sum_{x_{1:t-1}} \mathbb{P}(y_{1:t}, x_{1:t})$  being the probability of observing  $y_{1:t}$  and being in state  $x_t$  at time  $t$ . The recursion is:

$$\begin{aligned} \alpha_1(h_i) &= \pi_{h_i} b_{h_i, y_1} \\ \alpha_t(h_i) &= b_{h_i, y_t} \sum_j \alpha_{t-1}(h_j) a_{h_j, h_i} \end{aligned} \quad (3)$$

and, finally,  $\mathbb{P}(y_{1:T}) = \sum_i \alpha_T(h_i)$ .

### 2.1.2. The Viterbi Algorithm

The Viterbi algorithm [3] finds the sequence of hidden states,  $x_{1:T}$ , that maximizes the joint probability of the observed and hidden sequences (1). It uses the same type of approach as the forward algorithm: a new table,  $\omega$ , is defined by  $\omega_t(x_t) = \max_{x_{1:t-1}} \{\mathbb{P}(y_{1:t}, x_{1:t})\}$ , the probability of a most likely decoding ending in  $x_t$  at time  $t$ , having observed  $y_{1:t}$ . This can be obtained as follows:

$$\begin{aligned} \omega_1(h_i) &= \pi_{h_i} b_{h_i, y_1} \\ \omega_t(h_i) &= b_{h_i, y_t} \cdot \max_j \{\omega_{t-1}(h_j) a_{h_j, h_i}\} \end{aligned} \quad (4)$$

After computing  $\omega$ , a most likely sequence of hidden states is retrieved by backtracking through the table, starting in entry  $\operatorname{argmax}_{h_i} \{\omega_T(h_i)\}$ .

### 2.1.3. The Posterior Decoding Algorithm

The posterior decoding [3] of an observed sequence is an alternative to the prediction given by the Viterbi algorithm. While the Viterbi algorithm computes the decoding with the highest joint probability, the posterior decoding computes a sequence of hidden states,  $x_{1:T}$ , such that  $x_t = \operatorname{argmax}_{h_i} \{\gamma_t(h_i)\}$  has the highest posterior probability  $\gamma_t(h_i) = \mathbb{P}(h_i | y_{1:T})$ . If we let  $\beta_t(h_i) = \mathbb{P}(y_{t+1:T} | h_i)$ , we have:

$$\begin{aligned} \gamma_t(h_i) &= \frac{\mathbb{P}(h_i, y_{1:t}) \mathbb{P}(y_{t+1:T} | h_i)}{\mathbb{P}(y_{1:T})} \\ &= \frac{\alpha_t(h_i) \beta_t(h_i)}{\sum_j \alpha_T(h_j)} \end{aligned} \quad (5)$$

The backward algorithm [3] gives  $\beta_t(h_i)$  by using a recursion similar to Equation (3). Thus, to compute the posterior decoding, we first fill out  $\alpha_t(h_i)$  and  $\beta_t(h_i)$  for all  $t$  and  $i$  and then compute the decoding by  $x_t = \operatorname{argmax}_{h_i} \{\gamma_t(h_i)\}$ .

### 2.1.4. The Posterior-Viterbi Algorithm

The posterior decoding algorithm often computes decodings that are very accurate locally, but it may return syntactically incorrect decodings; *i.e.*, decodings with transitions that have a probability of zero. The posterior-Viterbi algorithm [22] corrects for this by computing a syntactically correct decoding  $x_{1:T}^* = \operatorname{argmax}_{x_{1:T} \in A_p} \left\{ \prod_{t=1}^T \gamma_t(x_t) \right\}$  with the highest posterior probability, where  $A_p$  is the set of syntactically correct decodings. To compute this, a new table,  $\tilde{\gamma}$ , is defined by  $\tilde{\gamma}_t(x_t) = \max_{x_{1:t-1} \in A_p} \left\{ \prod_{i=1}^t \gamma_i(x_i) \right\}$ , the maximum posterior probability of a decoding from  $A_p$  ending in  $x_t$  at time  $t$ . The table is filled using the recursion:

$$\begin{aligned} \tilde{\gamma}_1(h_i) &= \gamma_1(h_i) \\ \tilde{\gamma}_t(h_i) &= \gamma_t(h_i) \cdot \max_{\{j: a_{h_j, h_i} > 0\}} \{ \tilde{\gamma}_{t-1}(h_j) \} \end{aligned} \tag{6}$$

After computing  $\tilde{\gamma}$ , a decoding from  $A_p$  with the highest posterior probability is retrieved by backtracking through  $\tilde{\gamma}$  from entry  $\operatorname{argmax}_{h_i} \{ \tilde{\gamma}_T(h_i) \}$ . We note that, provided that the posterior decoding algorithm returns a decoding from  $A_p$ , the posterior-Viterbi algorithm will return the same decoding.

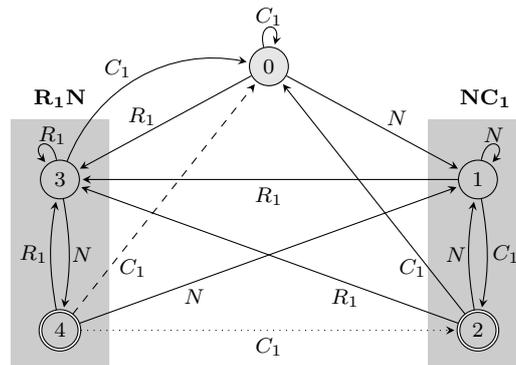
### 2.2. Automata

In this paper, we are interested in patterns over the hidden alphabet  $\mathcal{H} = \{h_1, h_2, \dots, h_N\}$  of an HMM. Let  $r$  be a regular expression over  $\mathcal{H}$ , and let  $FA_{\mathcal{H}}(r) = (Q, \mathcal{H}, q_0, A, \delta)$  [23] be the deterministic finite automaton (DFA) that recognizes the language described by  $(h_1 | h_2 | \dots | h_N)^*(r)$ , where  $Q$  is the finite set of states,  $q_0 \in Q$  is the initial state,  $A \subseteq Q$  is the set of accepting states and  $\delta : Q \times \mathcal{H} \rightarrow Q$  is the transition function.  $FA_{\mathcal{H}}(r)$  accepts any string that has  $r$  as a suffix. We construct the DFA  $EA_{\mathcal{H}}(r) = (Q, \mathcal{H}, q_0, A, \delta_E)$  as an extension of  $FA_{\mathcal{H}}(r)$ , where  $\delta_E$  is defined by:

$$\begin{aligned} \forall q \in Q \setminus A, \quad \forall h \in \mathcal{H}, \quad \delta_E(q, h) &= \delta(q, h) \\ \forall q \in A, \quad \forall h \in \mathcal{H}, \quad \delta_E(q, h) &= \delta(q_0, h) \end{aligned} \tag{7}$$

Essentially,  $EA_{\mathcal{H}}(r)$  restarts every time it reaches an accepting state. Figure 2 shows  $FA_{\mathcal{H}}(r)$  and  $EA_{\mathcal{H}}(r)$  for  $r = (NC_1) | (R_1N)$  with the hidden alphabet  $\mathcal{H} = \{N\} \cup \{C_i, R_i \mid 1 \leq i \leq 3\}$  of the HMM from Figure 2. Both automata have  $Q = \{0, 1, 2, 3, 4\}$ ,  $q_0 = 0$ , and  $A = \{2, 4\}$ . State 1 marks the beginning of  $NC_1$ , while state 3 corresponds to the beginning of  $R_1N$ . State 2 accepts  $NC_1$ , and state 4 accepts  $R_1N$ . As  $C_2, C_3, R_2$  and  $R_3$  are not part of  $r$ , using them, the automaton restarts by transitioning to state 0 from all states. We left these transitions out of the figure for clarity. The main difference between  $FA_{\mathcal{H}}(r)$  and  $EA_{\mathcal{H}}(r)$  is that they correspond to overlapping and non-overlapping occurrences, respectively. For example, for the input string,  $R_1NC_1$ ,  $FA_{\mathcal{H}}(r)$  first finds  $R_1N$  using state 4, from which it transitions to state 2 and matches  $NC_1$ . However, after  $EA_{\mathcal{H}}(r)$  recognizes  $R_1N$ , it transitions back to state 0, not matching  $NC_1$ . The algorithms we provide are independent of which of the two automata is used, and therefore, all that remains is to switch between them when needed. In our implementation, we used an automata library for Java [24] to obtain  $FA_{\mathcal{H}}(r)$ , which we then converted to  $EA_{\mathcal{H}}(r)$ .

**Figure 2.** Two automata,  $FA_{\mathcal{H}}(r)$  and  $EA_{\mathcal{H}}(r)$ , for the pattern  $r = (NC_1) \mid (R_1N)$ ,  $\mathcal{H} = \{N\} \cup \{C_i, R_i \mid 1 \leq i \leq 3\}$ ,  $Q = \{0, 1, 2, 3, 4\}$ ,  $q_0 = 0$ , and  $A = \{2, 4\}$ . States 1, 2 and 3, 4 are used for matching sequences ending with  $NC_1$  and  $R_1N$ , respectively, as marked with gray boxes. The two automata differ only with respect to transitions from accepting states: the dotted transition belongs to  $FA_{\mathcal{H}}(r)$  and the dashed one to  $EA_{\mathcal{H}}(r)$ . For clarity, the figure lacks transitions going from all states to state 0 using  $C_2, C_3, R_2$  and  $R_3$ .



### 3. Results and Discussion

Consider an HMM as defined previously, and let  $r$  be a regular expression over the alphabet of hidden states,  $\mathcal{H}$ . We present a modified version of the forward algorithm to compute the distribution of the number of occurrences of  $r$ , which we then use in an adaptation of the Viterbi algorithm and the posterior-Viterbi algorithm to obtain an improved prediction.

#### 3.1. The Restricted Forward Algorithm

Let  $O_r(x_{1:T})$  be the number of matches of  $r$  in  $x_{1:T}$ . We wish to estimate  $O_r$  by using its probability distribution. We do this by running the HMM and  $FA_{\mathcal{H}}(r)$  in parallel. Let  $FA_{\mathcal{H}}(r)_t$  be the state in which the automaton is after  $t$  transitions, and define  $\hat{\alpha}_t(x_t, k, q) = \sum_{\{x_{1:t-1} : O_r(x_{1:t})=k\}} \mathbb{P}(y_{1:t}, x_{1:t}, FA_{\mathcal{H}}(r)_t = q)$  to be the entries of a new table,  $\hat{\alpha}$ , where  $k = 0, \dots, m$  and  $m \leq T$  is the maximum number of pattern occurrences in a hidden sequence of length  $T$ . The table entries are the probabilities of having observed  $y_{1:t}$ , being in hidden state  $x_t$  and automaton state  $q$  at time  $t$  and having seen  $k$  occurrences of the pattern, corresponding to having visited accepting states  $k$  times. Letting  $\delta^{-1}(q, h_i) = \{q' \mid \delta(q', h_i) = q\}$  be the automaton states from which a transition to  $q$  exists, using hidden state  $h_i$  and  $\mathbb{1}$  being the indicator function, mapping a Boolean expression to one if it is satisfied and to zero otherwise, we have that:

$$\hat{\alpha}_t(x_t, k, q) = \sum_{\substack{x_{1:t-1} : \\ O_r(x_{1:t-1})=k-\mathbb{1}(q \in A)}} \mathbb{P}(y_{1:t}, x_{1:t}, FA_{\mathcal{H}}(r)_t = q) \tag{8}$$

$$\begin{aligned} \mathbb{P}(y_{1:t}, x_{1:t}, FA_{\mathcal{H}}(r)_t = q) &= \sum_{q' \in \delta^{-1}(q, x_t)} \mathbb{P}(y_{1:t}, x_{1:t}, FA_{\mathcal{H}}(r)_{t-1} = q') \\ &= b_{x_t, y_t} a_{x_{t-1}, x_t} \sum_{q' \in \delta^{-1}(q, x_t)} \mathbb{P}(y_{1:t-1}, x_{1:t-1}, FA_{\mathcal{H}}(r)_{t-1} = q') \end{aligned} \tag{9}$$

Using Equations (8) and (9), the recursion for  $\hat{\alpha}$  becomes:

$$\hat{\alpha}_1(h_i, k, q) = \pi_{h_i} b_{h_i, y_1} \mathbb{1}(q_0 \in \delta^{-1}(q, h_i)) \cdot \begin{cases} \mathbb{1}(q \notin A) & \text{if } k = 0 \\ \mathbb{1}(q \in A) & \text{if } k = 1 \\ 0 & \text{otherwise} \end{cases} \tag{10}$$

$$\hat{\alpha}_t(h_i, k, q) = b_{h_i, y_t} \sum_j a_{h_j, h_i} \sum_{q' \in \delta^{-1}(q, h_i)} \hat{\alpha}_{t-1}(h_j, k - \mathbb{1}(q \in A), q')$$

These probabilities now allow for the evaluation of the distribution of the number of occurrences, conditioned on the observed data:

$$\begin{aligned} \mathbb{P}(k \text{ occurrences of } r \mid y_{1:T}) &= \frac{\mathbb{P}(k \text{ occurrences of } r, y_{1:T})}{\mathbb{P}(y_{1:T})} \\ &= \frac{1}{\mathbb{P}(y_{1:T})} \sum_{i,q} \hat{\alpha}_T(h_i, k, q) \end{aligned} \tag{11}$$

from which the expectation can be computed. The likelihood of the data can be obtained from either the forward or restricted forward algorithm,  $\mathbb{P}(y_{1:T}) = \sum_i \alpha(h_i) = \sum_{i,k,q} \hat{\alpha}_T(h_i, k, q)$ .

The  $\hat{\alpha}$  table has  $T \cdot N \cdot m \cdot |Q|$  entries, and the computation of each requires  $\mathcal{O}(N|Q|)$ , leading to a  $\mathcal{O}(TN^2m|Q|^2)$  running time and a space consumption of  $\mathcal{O}(TNm|Q|)$ . In practice, both time and space consumption can be reduced. The restricted forward algorithm can be run for values of  $k$  that are gradually increasing up to  $k_{max}$  for which  $\mathbb{P}(\text{at most } k_{max} \text{ occurrences of } r \mid y_{1:T})$  is greater than, e.g., 99.99%. This  $k_{max}$  is generally significantly less than  $m$ , while the expectation of the number of matches of  $r$  can be reliably calculated from this truncated distribution. The space consumption can be reduced to  $\mathcal{O}(N|Q|)$ , because the calculation at time  $t$  for a specific value,  $k$ , depends only on the results at time  $t - 1$  for  $k$  and  $k - 1$ .

### 3.2. Restricted Decoding Algorithms

The aim of the restricted decoding algorithms is to obtain a sequence of hidden states,  $x_{1:T}$ , for which  $O_r(x_{1:T}) \in [l, u]$ , where  $l$  and  $u$  are set to, for example, the expected number of occurrences, which can be calculated from the distribution. The restricted decoding algorithms are built in the same way as the restricted forward was obtained: a new table is defined, which is filled using a simple recursion. The evaluation of the table is followed by backtracking to obtain a sequence of hidden states, which contains between  $l$  and  $u$  occurrences of the pattern. The two restricted decoding algorithms use  $\mathcal{O}(TNu|Q|)$  space and  $\mathcal{O}(TN^2u|Q|^2)$  time.

The entries in the table for the restricted Viterbi algorithm contain the probability of a most likely decoding containing  $k$  pattern occurrences, ending in state  $x_t$  and automaton state  $q$  at time  $t$  and having observed  $y_{1:t}$ ,  $\hat{\omega}_t(x_t, k, q) = \max_{\{x_{1:t-1} : O_r(x_{1:t})=k\}} \{\mathbb{P}(y_{1:t}, x_{1:t}, FA_{\mathcal{H}}(r)_t = q)\}$  with  $k = 0, \dots, u$ . The corresponding recursion is:

$$\hat{\omega}_1(h_i, k, q) = \pi_{h_i} b_{h_i, y_1} \mathbb{1}(q_0 \in \delta^{-1}(q, h_i)) \cdot \begin{cases} \mathbb{1}(q \notin A) & \text{if } k = 0 \\ \mathbb{1}(q \in A) & \text{if } k = 1 \\ 0 & \text{otherwise} \end{cases} \tag{12}$$

$$\hat{\omega}_t(h_i, k, q) = b_{h_i, y_t} \cdot \max_j \left\{ a_{h_j, h_i} \cdot \max_{q' \in \delta^{-1}(q, h_i)} \{\hat{\omega}_{t-1}(h_j, k - \mathbb{1}(q \in A), q')\} \right\}$$

and the backtracking starts in entry  $\operatorname{argmax}_{i,k \in [l,u],q} \{\hat{\omega}_T(h_i, k, q)\}$ .

For the restricted posterior-Viterbi algorithm, we compute the highest posterior probability of a decoding from  $A_p$  containing  $k$  pattern occurrences, ending in state  $x_t$  and automaton state  $q$  at time  $t$  and having observed  $y_{1:t}$ ,  $\hat{\gamma}_t(x_t, k, q) = \max_{\{x_{1:t-1} : O_r(x_{1:t})=k\}} \{\mathbb{P}(x_{1:t}, FA_{\mathcal{H}}(r)_t = q \mid y_{1:T})\}$ . We have:

$$\hat{\gamma}_1(h_i, k, q) = \gamma_1(h_i) \mathbb{1}(q_0 \in \delta^{-1}(q, h_i)) \cdot \begin{cases} \mathbb{1}(q \notin A) & \text{if } k = 0 \\ \mathbb{1}(q \in A) & \text{if } k = 1 \\ 0 & \text{otherwise} \end{cases} \tag{13}$$

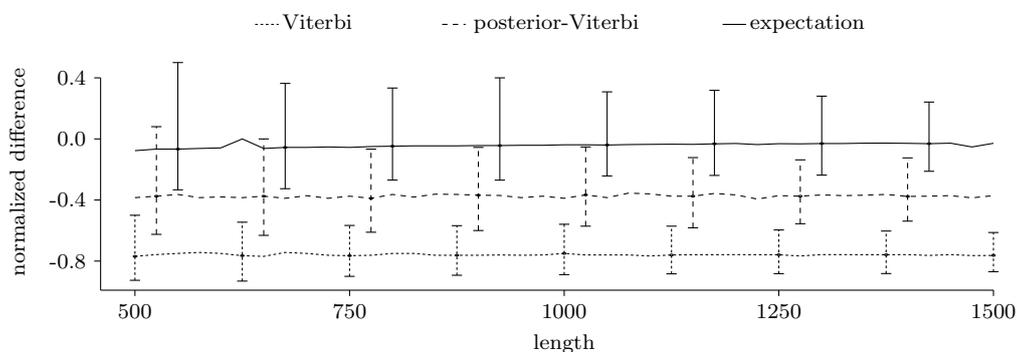
$$\hat{\gamma}_t(h_i, k, q) = \gamma_t(h_i) \cdot \max_{\{h_j : a_{h_j, h_i} > 0\}} \left\{ \max_{q' \in \delta^{-1}(q, h_i)} \{\hat{\gamma}_{t-1}(h_j, k - \mathbb{1}(q \in A), q')\} \right\}$$

The backtracking starts in entry  $\operatorname{argmax}_{i,k \in [l,u],q} \{\hat{\gamma}_T(h_i, k, q)\}$ .

### 3.3. Experimental Results on Simulated Data

We implemented the algorithms in Java, validated and evaluated their performance experimentally as follows: We first generated a test set consisting of 500 pairs of observed and hidden sequences for each length  $L = 500, 525, \dots, 1,500$  from the gene finder in Figure 2. As the HMM is used for finding genes, an obvious choice of pattern is  $r = (NC_1) \mid (R_1N)$ , corresponding to the start of a gene. For each of the sequences, we estimated the number of overlapping pattern occurrences with the expected number of pattern occurrences, computed using the restricted forward algorithm, which we then used to run the restricted decoding algorithms. We also computed the prediction given by the two unrestricted decoding algorithms for comparison.

**Figure 3.** Normalized difference,  $\frac{\text{estimate}}{\text{true value}} - 1$ , between the true number of pattern occurrences, the number given by the two unrestricted decoding algorithms and the expected number of pattern occurrences computed using the restricted forward algorithm. For each sequence length, we show the median of the normalized differences in the 500 experiments, together with the 0.025 and 0.975 quantiles, given as error bars.



#### 3.3.1. Counting Pattern Occurrences

Figure 3 shows the power of the restricted forward algorithm and the two unrestricted decoding algorithms to recover the true number of pattern occurrences. For each given length, we computed the normalized difference,  $\frac{\text{estimate}}{\text{true value}} - 1$ , and plotted the median over the 500 sequences together with

the 0.025 and 0.975 quantiles, given as error bars. As Figure 3 shows, the expectation gives a very good estimate. The decoding algorithms' performances are significantly lower, and they always give underestimates. This may be partly due to the model structure, where transitions to and from coding regions have low probability, leading to few, but long, genes.

### 3.3.2. Quality of Predictions

We compared the predictive power of the two decoding algorithms in the original and restricted versions, using the expectation for the number of pattern occurrences. For each length in the test set, we measured the quality of each method, both at the nucleotide and gene level, following the analysis in [25]. Because the measures we use require binary data, we first converted the true hidden sequence and decoding to binary data containing non-coding areas and genes, by first considering the genes on the reverse strand as non-coding and calculating the measures for the genes on the direct strand, and *vice versa*. The final plotted measures are the averages obtained from the two conversions.

**Nucleotide level.** To investigate the quality at the nucleotide level, we compared the decoding and the true hidden state position by position. Each position can be classified as a true positive (predicted as part of a gene when it was part of a gene), true negative (predicted as non-coding when it was non-coding), false positive (predicted as part of a gene when it was non-coding) and false negative (predicted as non-coding when it was part of a gene). Using the total number of true positives ( $tp$ ), true negatives ( $tn$ ), false positives ( $fp$ ) and false negatives ( $fn$ ), we calculated the sensitivity, specificity and Matthew's correlation coefficient (MCC):

$$sens = \frac{tp}{tp + fn} \quad (14)$$

$$spec = \frac{tn}{tn + fp} \quad (15)$$

$$mcc = \frac{tp \cdot tn - fp \cdot fn}{\sqrt{(tp + fp) \cdot (tp + fn) \cdot (tn + fp) \cdot (tn + fn)}} \quad (16)$$

Sensitivity and specificity are always between zero and one and relate to how well the algorithms are able to find genes (true positives) and non-coding regions (true negatives), respectively. MCC reflects the overall correctness and lies between  $-1$  and  $1$ , where  $1$  represents perfect prediction.

**Gene level.** When looking at the decoding position by position, genes that are predicted correctly do not contribute to the measures in an equal manner, but rather, the longer the gene, the more contribution it brings. However, it is interesting how well the genes are recovered, independent of how long they are. To measure this, we consider a predicted gene as one true positive if it overlaps by at least 50% of a true gene and as one false positive if there is no true gene with which it overlaps by at least 50%. Each true gene for which there is no predicted gene that overlaps by at least 50% counts as one false negative; see Figure 4. In this context, true negatives, *i.e.*, the areas where there was no gene and no gene was predicted, are not considered, as they are not informative. The final measures are the recall, precision and F-score:

$$rec = \frac{tp}{tp + fn} \quad (17)$$

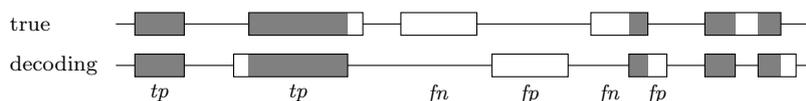
$$prec = \frac{tp}{tp + fp} \quad (18)$$

$$f\text{-score} = 2 \cdot \frac{rec \cdot prec}{rec + prec} \quad (19)$$

$$= \frac{2 \cdot tp}{2 \cdot tp + fn + fp} \quad (20)$$

They are all between zero and one and reflect how well the true genes have been recovered. The recall gives the fraction of true genes that have been found, while the precision gives the fraction of the predicted genes that are true genes. The F-score is the harmonic mean of the two.

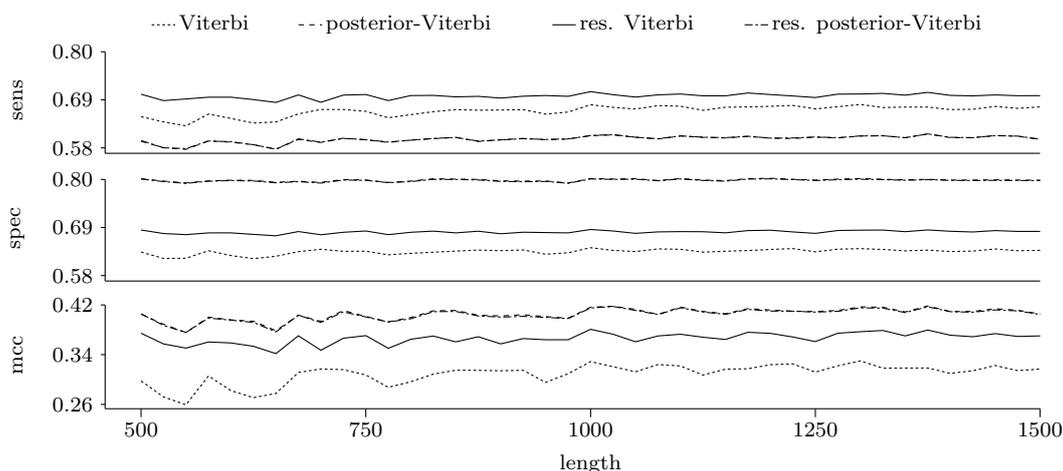
**Figure 4.** Error types at the gene level. A predicted gene is considered one true positive if it overlaps with at least 50% of a true gene and one false positive if there is no true gene with which it overlaps by at least 50%. Each true gene for which there is no predicted gene that overlaps by at least 50% counts as one false negative. True genes that are covered more than 50% by predicted genes, but for which there is no single predicted gene that covers a minimum of 50% are disregarded.



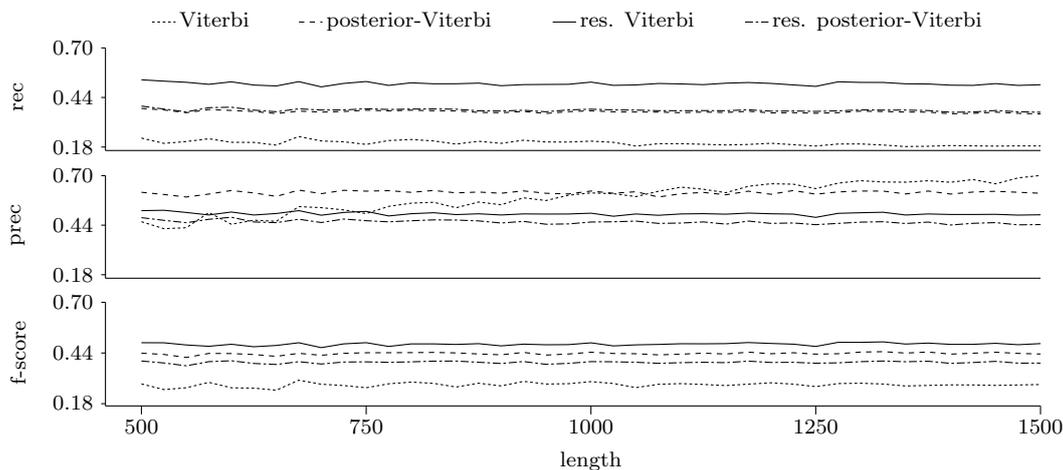
Figures 5 and 6 show the quality of predictions at the nucleotide and gene level, respectively. When comparing the Viterbi algorithm with the restricted Viterbi algorithm, it is clear that the restriction brings a great improvement to the prediction, as the restricted version has an increased power in all measures considered, with the exception of precision, where the Viterbi algorithm shows a tendency of increased precision with sequence length. However, when comparing the posterior-Viterbi algorithm with its restricted version, it is apparent that the restricted version does as good at the nucleotide level, but it performs worse at the gene level. By inspecting the predictions of the two methods, it was clear that the restricted posterior-Viterbi obtained an increased number of genes dictated by the expectation by just fractioning the genes predicted by the posterior-Viterbi. We believe this happens because the posterior-Viterbi algorithm finds the best local decoding, and therefore, adding global information, such as the total number of genes in the prediction, does not aid in the decoding. On the other hand, as the Viterbi algorithm finds the best global decoding, using the extra information results in a significant improvement of the prediction. Overall, at the nucleotide level, the posterior-Viterbi shows the best performance, while at the gene level, the restricted Viterbi has the highest quality. One might expect this, given the nature of the algorithms.

Apart from these experiments, we also ran the algorithms on the annotated *E. coli* genome (GenBank accession number U00096). In this set of experiments, we split the genome into sequences of a length of approximately 10,000, for which we ran the algorithms as previously described. We found the same trends as in the performance for simulated data (results not shown). When using the *E. coli* data, we also recorded the running time of the algorithms, and we found that the restricted algorithms are about 45 times slower than the standard algorithms, which is faster than the worst case scenario, which would lead to a  $k \cdot |Q|^2 = 7 \cdot 5^2 = 175$  slowdown, as the average expectation of the number of patterns per sequence was  $k = 7$ .

**Figure 5.** Prediction quality at the nucleotide level given by average sensitivity, specificity and Matthew’s correlation coefficient (MCC) for the decoding algorithms. We ran the restricted decoding algorithms using the expectation calculated from the distribution returned by the restricted forward algorithm. The plots show a zoom-in of the three measures. As both sensitivity and specificity are between zero and one, the Y-axes in these two plots have the same scale.



**Figure 6.** Prediction quality at the gene level given by average recall, precision and F-score for the decoding algorithms. We ran the restricted decoding algorithms using the expectation calculated from the distribution returned by the restricted forward algorithm. The plots show a zoom-in of the three measures with the same scale on the Y axes.



#### 4. Conclusions

We have introduced three novel algorithms that efficiently combine the theory of Hidden Markov Models with automata and pattern matching to recover pattern occurrences in the hidden sequence. First, we computed the distribution of the number of pattern occurrences by using an algorithm similar to the forward algorithm. This problem has been treated in [16] by means of Markov chain embedding, using simple finite sets of strings as patterns. Our method is, however, more general, as it allows the use of regular expressions.

From the occurrence number distribution, we calculated the expected number of pattern matches, which estimated the true number of occurrences with high precision. We then used the distribution to alter the prediction given by the two most widely used decoding algorithms: the Viterbi algorithm and the posterior-Viterbi algorithm. We have shown that in the case of the Viterbi algorithm, which finds the best global prediction, using the expected number of pattern occurrences greatly improves the prediction, both at the nucleotide and gene level. However, in the case of the posterior-Viterbi algorithm, which finds the best local prediction, such an addition only fragments the predicted genes, leading to a poorer prediction. Overall, deciding which algorithm is best depends on the final measure used, but as our focus was on finding genes, we conclude that the restricted Viterbi algorithm showed the best result.

As the distribution obtained from the restricted forward algorithm facilitates the calculation of the distribution of the waiting time until the occurrence of the  $k^{\text{th}}$  pattern match, the restricted Viterbi algorithm could potentially be further extended to incorporate this distribution while calculating the joint probability of observed and hidden sequences. Weighted transducers [26] are sequence modeling tools similar to HMMs, and analyzing patterns in the hidden sequence can potentially also be done by composition of the transducers, which describe the HMM and the automaton.

Our method can presumably be used with already existing HMMs to improve their prediction, by using patterns that reflect the problems studied using the HMMs. For example, in [13], an HMM is used for finding frameshifts in coding regions. In this situation, the pattern would capture the sequence of hidden states that corresponds to a frameshift. In HMMs used for phylogenetic analysis [11,12], the hidden states represent trees, and an event of interest is a shift in the tree. A pattern capturing a change in the hidden state could thus aid in improving the prediction. In HMMs built for probabilistic alignments [10], the pattern could capture the start of an indel, and our method could potentially aid in finding a more accurate indel rate estimate. There is therefore a substantial potential in the presented method to successfully improve the power of HMMs.

## Acknowledgments

We are grateful to Jens Ledet Jensen for useful discussions in the initial phase of this study.

## Conflicts of Interest

The authors declare no conflict of interest.

## References

1. Chong, J.; Yi, Y.; Faria, A.; Satish, N.; Keutzer, K. Data-parallel Large Vocabulary Continuous Speech Recognition on Graphics Processors. In Proceedings of the 1st Annual Workshop on Emerging Applications and Many Core Architecture, Beijing, China, June 2008; pp. 23–35.
2. Gales, M.; Young, S. The application of hidden Markov models in speech recognition. *Found. Trends Signal Process.* **2007**, *1*, 195–304.
3. Rabiner, L. A tutorial on hidden Markov models and selected applications in speech recognition. *Proc. IEEE* **1989**, *77*, 257–286.

4. Li, J.; Gray, R. *Image Segmentation and Compression Using Hidden Markov Models*; Springer: Berlin/Heidelberg, Germany, 2000; Volume 571.
5. Karplus, K.; Barrett, C.; Cline, M.; Diekhans, M.; Grate, L.; Hughey, R. Predicting protein structure using only sequence information. *Proteins Struct. Funct. Bioinformatics* **1999**, *37*, 121–125.
6. Krogh, A.; Brown, M.; Mian, I.; Sjolander, K.; Haussler, D. Hidden Markov models in computational biology: Applications to protein modeling. *J. Mol. Biol.* **1994**, *235*, 1501–1531.
7. Krogh, A.; Larsson, B.; von Heijne, G.; Sonnhammer, E. Predicting transmembrane protein topology with a hidden Markov model: Application to complete genomes. *J. Mol. Biol.* **2001**, *305*, 567–580.
8. Eddy, S. Multiple Alignment Using Hidden Markov Models. In Proceedings of the Third International Conference on Intelligent Systems for Molecular Biology, Cambridge, UK, 16–19 July 1995; Volume 3, pp. 114–120.
9. Eddy, S. Profile hidden Markov models. *Bioinformatics* **1998**, *14*, 755–763.
10. Lunter, G. Probabilistic whole-genome alignments reveal high indel rates in the human and mouse genomes. *Bioinformatics* **2007**, *23*, i289–i296.
11. Mailund, T.; Dutheil, J.Y.; Hobolth, A.; Lunter, G.; Schierup, M.H. Estimating divergence time and ancestral effective population size of bornean and sumatran orangutan subspecies using a coalescent hidden Markov model. *PLoS Genet.* **2011**, *7*, e1001319.
12. Siepel, A.; Haussler, D. Phylogenetic Hidden Markov Models. In *Statistical Methods in Molecular Evolution*; Nielsen, R., Ed.; Springer: New York, NY, USA, 2005; pp. 325–351.
13. Antonov, I.; Borodovsky, M. GeneTack: Frameshift identification in protein-coding sequences by the Viterbi algorithm. *J. Bioinforma. Comput. Biol.* **2010**, *8*, 535–551.
14. Lukashin, A.; Borodovsky, M. GeneMark.hmm: New solutions for gene finding. *Nucleic Acids Res.* **1998**, *26*, 1107–1115.
15. Krogh, A.; Mian, I.S.; Haussler, D. A hidden Markov model that finds genes in *E.coli* DNA. *Nucleic Acids Res.* **1994**, *22*, 4768–4778.
16. Aston, J.A.D.; Martin, D.E.K. Distributions associated with general runs and patterns in hidden Markov models. *Ann. Appl. Stat.* **2007**, *1*, 585–611.
17. Fu, J.; Koutras, M. Distribution theory of runs: A Markov chain approach. *J. Am. Stat. Appl.* **1994**, *89*, 1050–1058.
18. Nuel, G. Pattern Markov chains: Optimal Markov chain embedding through deterministic finite automata. *J. Appl. Probab.* **2008**, *45*, 226–243.
19. Wu, T.L. On finite Markov chain imbedding and its applications. *Methodol. Comput. Appl. Probab.* **2011**, *15*, 453–465.
20. Lladser, M.; Betterton, M.; Knight, R. Multiple pattern matching: A Markov chain approach. *J. Math. Biol.* **2008**, *56*, 51–92.
21. Nicodeme, P.; Salvy, B.; Flajolet, P. Motif statistics. *Theor. Comput. Sci.* **2002**, *287*, 593–617.
22. Fariselli, P.; Martelli, P.L.; Casadio, R. A new decoding algorithm for hidden Markov models improves the prediction of the topology of all-beta membrane proteins. *BMC Bioinformatics* **2005**, *6*, doi:10.1186/1471-2105-6-S4-S12.

23. Thompson, K. Programming techniques: Regular expression search algorithm. *Commun. ACM* **1968**, *11*, 419–422.
24. Møller, A. dk.brics.automaton—Finite-State Automata and Regular Expressions for Java, 2010. Available online: <http://www.brics.dk/automaton/> (accessed on 16 January 2012).
25. Burset, M.; Guigo, R. Evaluation of gene structure prediction programs. *Genomics* **1996**, *34*, 353–367.
26. Mohri, M. Weighted Automata Algorithms. In *Handbook of Weighted Automata*; Springer: Berlin/Heidelberg, Germany, 2009; pp. 213–254.

© 2013 by the authors; licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution license (<http://creativecommons.org/licenses/by/3.0/>).