ACADEMIC PRESS

Journal of Computer and System Sciences 65 (2002) 545–569

JOURNAL OF COMPUTER AND SYSTEM SCIENCES

www.academicpress.com

The consensus string problem and the complexity of comparing hidden Markov models

Rune B. Lyngsø[a,1] and Christian N.S. Pedersen[b,*,2,3]

[a] *Department of Computer Science, University of California at Santa Cruz, Santa Cruz, CA 95064, USA*
[b] *BRICS, Department of Computer Science, University of Aarhus, 8000 Århus C, Denmark*

Received 30 April 2001; received in revised form 31 October 2001

Abstract

The basic theory of hidden Markov models was developed and applied to problems in speech recognition in the late 1960s, and has since then been applied to numerous problems, e.g. biological sequence analysis. Most applications of hidden Markov models are based on efficient algorithms for computing the probability of generating a given string, or computing the most likely path generating a given string. In this paper we consider the problem of computing the most likely string, or consensus string, generated by a given model, and its implications on the complexity of comparing hidden Markov models. We show that computing the consensus string, and approximating its probability within any constant factor, is **NP**-hard, and that the same holds for the closely related labeling problem for class hidden Markov models. Furthermore, we establish the **NP**-hardness of comparing two hidden Markov models under the L_∞- and L_1-norms. We discuss the applicability of the technique used for proving the hardness of comparing two hidden Markov models under the L_1-norm to other measures of distance between probability distributions. In particular, we show that it cannot be used for proving **NP**-hardness of determining the Kullback–Leibler distance between two hidden Markov models, or of comparing them under the L_k-norm for any fixed even integer k.

1. Introduction

A hidden Markov model is a description of a probability distribution over a set of strings. It is convenient to consider a hidden Markov model as a generative model in which a run generates a string with a certain probability. A run starts in a special start-state, and continues by following a first-order Markov chain of states, called

*Corresponding author.
E-mail addresses: rlyngsoe@cse.ucsc.edu (R.B. Lyngsø), cstorm@daimi.au.dk (C.N.S. Pedersen).
[1] Supported by grants from the Carlsberg Foundation and the Program in Mathematics and Molecular Biology.
[2] Partially supported by the Bioinformatics Research Center (BiRC) at the University of Aarhus, and the IST Programme of the EU under contract number IST-1999-14186 (ALCOM-FT).
[3] Basic Research in Computer Science, funded by the Danish National Research Foundation.

0022-0000/02/$ - see front matter
PII: S0022-0000(02)00009-0

the path, until a special end-state is reached. A symbol from a finite alphabet is emitted according to some probability distribution each time a non-silent state is entered. The theory of hidden Markov models was developed and applied to problems in speech recognition in the late 1960s and early 1970s. Rabiner [17] gives a good overview of the theory of hidden Markov models and its applications in speech recognition. Hidden Markov models are also applied in other areas than speech recognition. One prominent area is computational biology where they have found many applications, e.g. modeling of DNA sequences [4], protein secondary structure prediction [2], gene finding [13], recognition of transmembrane proteins [19], and characterization of biological sequence families [14].

Applications of hidden Markov models are often based on two fundamental questions. Given a hidden Markov model and a string, we might want to determine the probability of the string under the model, i.e. the probability that the model has generated the string. This can be used for *classification* of the string as either belonging to the family of strings represented by the model or not. Or we might want to determine the most likely path of states through the model that generates the string. This can be used for *annotating* the string with states from the model. Dynamic programming algorithms solving these problems are described in e.g. [17]. Constructing a hidden Markov model for analyzing a specific family of strings is stated as a third fundamental problem in [17]. This problem, referred to as the *loading* problem, is usually solved by finding a model adhering to some structural constraints that fit a set of known strings from the family. In general, the loading problem is hard, see e.g. [1], but methods exist for *training* a model, i.e. iteratively updating its parameters to improve the fit to the known strings.

A fourth problem which is natural to state about hidden Markov models is the complementary problem of the loading problem. Given a hidden Markov model, what is the string that best fits the model, i.e. what string is the most likely string under the model. We will refer to this problem as the *consensus string* problem. The consensus string problem seems to be of less importance than the three above-mentioned problems, at least we are not aware of any literature where the problem has been addressed previously. However, if an efficient method for determining the consensus string was available, applications could be obtaining a single string representative of a family of strings, and guiding the training of a model such that the probabilities of the strings used to train the model are close to the probability of the consensus string, or such that the consensus string has high probability (i.e. discouraging the trained model from generalizing too broadly). Furthermore, Krogh in [13] demonstrates that superior annotations of sequences can be obtained by using class hidden Markov models instead of standard hidden Markov models for gene finding. One can say that the annotation, or labeling, problem for class hidden Markov models is an instance of the consensus string problem. Unfortunately, in this paper we prove that the consensus string problem is **NP**-hard. But it turns out that the consensus string problem is a good basis for investigating some problems concerned with comparing hidden Markov models; indeed, this was our main motivation for studying the consensus string problem.

Comparing two hidden Markov models is an interesting theoretical problem with practical applications as well. For example, by comparing two profile hidden Markov models, e.g. from the Pfam protein families database [3], we compare entire sequence families instead of just individual members. In [15] we present methods for comparing hidden Markov models, and describe how to compute the Euclidean distance (the L_2-distance) between two models in polynomial time. In this paper we study the problem of comparing two hidden Markov models under the L_∞- and the L_1-norms. Using the hardness of determining the consensus string, we show that

comparing two hidden Markov models under the L_∞-norm is **NP**-hard. Furthermore, we link the consensus string probability for models constructed in the consensus string hardness proof to the L_1-norm between two models. We utilize this link to show that comparing two hidden Markov models under the L_1-norm is **NP**-hard but show that it cannot be used to establish the hardness of comparing models under the L_{2k}-norm for any $k \in \mathbf{N}$. The L_1-norm is of special interest as it equals twice the *variation distance*, i.e. the maximum numerical difference between the probability of any set of events under two distributions, see e.g. [6].

The rest of the paper is organized as follows. In Section 2 we discuss hidden Markov models in more detail. In Section 3 we show that computing the consensus string, and approximating its probability within any constant factor, is **NP**-hard. In Section 4 we show that the complexity results apply to the labeling problem for class hidden Markov models. In Section 5 we present and discuss heuristics for computing strings of high probability. In Section 6 we consider the general problem of comparing hidden Markov models, and show that comparison under the L_∞- and the L_1-norms is **NP**-hard. In Section 7 we discuss future work and open problems and summarize the status of the tractability of comparing hidden Markov models by various well-known distance measures.

2. Hidden Markov models

Let M be a hidden Markov model that generates strings over some finite alphabet Σ with probability distribution P_M, i.e. $P_M(s)$ denotes the probability of $s \in \Sigma^*$ in model M. Like a classical Markov model, a hidden Markov model consists of a set of interconnected states. We use $a^M_{q,q'}$ to denote the probability of a transition from state q to state q' in model M. These probabilities are called *state transition probabilities*. For convenience, we let the *number of transitions* of a model M denote the number of transitions with non-zero probability in M. The transition structure of a hidden Markov model is often shown as a directed graph with a node for each state, and an edge between two nodes if the corresponding state transition probability is non-zero. Models where this transition structure is acyclic except for self-loops, i.e. transitions from a state to itself, are called left–right models [11]. Fig. 1 shows a possible transition structure. Unlike a classical Markov model, a state in a hidden Markov model can emit a symbol according to a local probability distribution over all possible symbols. We use $e^M_{q,\sigma}$ to denote the probability of emitting symbol $\sigma \in \Sigma$ in state q in model M. These probabilities are called *symbol emission probabilities*. If a state does not have symbol emission probabilities we say that the state is a *silent* state.

It is convenient to consider a hidden Markov model as a generative model in which a *run* generates a string. A run of a hidden Markov model begins in a special start-state and continues from state to state according to the state transition probabilities until a special end-state is reached. Each time a non-silent state is entered, a symbol is emitted according to the symbol emission probabilities of the state. In this paper we also use the notion of a *partial run* which is a run that does not necessarily start in the special start-state or end in the special end-state. We refer to the Markovian sequence of states in a run as the *path* followed by the run. The string generated by a run is the concatenation of the symbols emitted along its path. The name "*hidden* Markov model" comes from the fact that the Markovian sequence of states followed by a run, the path, is hidden while only the emitted symbols, the generated string, are observable. In the rest of this paper we will restrict our attention to hidden Markov models that have zero probability for infinite runs, i.e. models

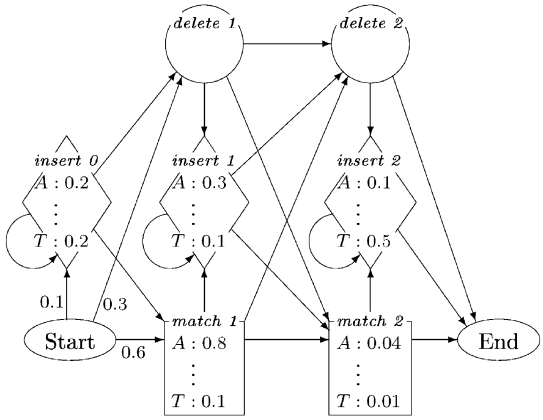

Fig. 1. The transition structure of a so-called profile hidden Markov model, a type of model used for modeling homologous biological sequences. The states are grouped into match-states (squares), insert-states (diamonds), and silent delete-states (circles).

where states to which there is a path from the start-state of non-zero probability have a path of non-zero probability to the end-state. For convenience we will assume that all states can be reached from the start-state, and hence that we can reach the end-state from all states.

The probability $P_M(\pi)$ of following a path $\pi = (\pi_0, \pi_1, \ldots, \pi_k)$ in model M is given by the state transition probabilities as

$$P_M(\pi) = \prod_{i=1}^{k} a^M_{\pi_{i-1},\pi_i}. \tag{1}$$

The probability $P_M(\pi, s)$ of following a path $\pi = (\pi_0, \pi_1, \ldots, \pi_k)$ in model M and emitting string s depends on the subsequence $(\pi_{i_1}, \pi_{i_2}, \ldots, \pi_{i_l})$ of non-silent states on the path π. If the length of string $s = s_1\, s_2 \cdots s_{l'}$ is different from the number of non-silent states along path π, the probability $P_M(\pi, s)$ is zero. Otherwise, the probability of following path π and emitting string s is

$$P_M(\pi, s) = P_M(\pi) P_M(s \mid \pi) = \prod_{i=1}^{k} a^M_{\pi_{i-1},\pi_i} \prod_{j=1}^{l} e^M_{\pi_{i_j},s_j}. \tag{2}$$

Since a run r of a hidden Markov model M is identified by a path π_r through the model and an emitted string s_r, we can define the probability of a run as

$$P_M(r) = P_M(\pi_r, s_r). \tag{3}$$

Finally, the probability $P_M(s)$ of model M generating a string s is the probability of following any path and emitting string s, that is

$$P_M(s) = \sum_{\pi} P_M(\pi, s). \tag{4}$$

In a standard hidden Markov model, a run generates just a single string. A *multi-track* hidden Markov model with t tracks can for each state emit a symbol to each of the t tracks, thus generating t strings by a single run. For multi-track hidden Markov models we use $e^M_{q,u,\sigma}$ to denote the probability of emitting symbol $\sigma \in \Sigma$ to track u in state q in model M. If a state does not have symbol emission probabilities for a track u we say that the state is *silent with respect to track u*.

The probability $P_M(\pi, s^1, \ldots, s^t)$ of following a path $\pi = (\pi_1, \pi_1, \ldots, \pi_k)$ and emitting string s^u to track u for all tracks $1 \leqslant u \leqslant t$ depends on the subsequence $(\pi_1^u, \pi_2^u, \ldots, \pi_{l_u}^u)$ of states that are non-silent with respect to track u for all $1 \leqslant u \leqslant t$. If l_u is different from the length of s^u for any $1 \leqslant u \leqslant t$, the probability is zero. Otherwise, the probability is

$$P_M(\pi, s^1, \ldots, s^t) = P_M(\pi) \prod_{u=1}^{t} P_M(s^u \mid \pi) = \prod_{i=1}^{k} a^M_{\pi_{i-1},\pi_i} \prod_{u=1}^{t} \prod_{j=1}^{l_u} e^M_{\pi_j^u, u, s_j^u}. \tag{5}$$

Two track hidden Markov models where one track is viewed as generating the observed sequence and the other track is viewed as generating an annotation, or classification, of the symbols of the observed sequence are sometimes called *class* hidden Markov models, cf. [12,13]. In these a state is either silent with respect to both tracks, or non-silent with respect to both tracks. Moreover, for class hidden Markov models one will often see that each non-silent state emits a particular symbol with probability one on the annotation track. In this case the annotation track becomes a listing of the classes which the non-silent states generating the observed sequence belong to, more than an independently generated sequence.

3. Hardness of finding the consensus string

Given a hidden Markov model M it is obvious to ask for the most likely string, i.e. the consensus string, in the model. In this section we will prove that answering this question is likely to be intractable as we will show that computing the probability of the consensus string generated by a hidden Markov model is **NP**-hard. We establish the hardness of the problem by a reduction from the problem of computing the size of the maximum clique in an undirected graph. I.e. we show that we can compute the size of the maximum clique in an undirected graph within the time it takes to compute the probability of the consensus string generated by a hidden Markov model plus the polynomial time it takes to construct a specific hidden Markov model from the given undirected graph. Since computing the size of the maximum clique is **NP**-hard, see e.g. [8], this implies that computing the probability of the consensus string is also **NP**-hard. As the probability of a given string can be computed in polynomial time, cf. [17], it also implies that computing the consensus string itself, and not only its probability, is **NP**-hard.

Let $G = (V, E)$ be the graph in which we want to find the size of a maximum clique. For simplicity, we assume that $V = \{1, 2, \ldots, |V|\}$. We will describe how to construct a hidden Markov model M_G that generates strings over the alphabet $\Sigma = V$ such that the probability of the consensus string in M_G is proportional to the maximum clique size of G. The model M_G consists of a start-state, an end-state, and a *layer* for every node v in V. The layers are sub-models that are only connected to the rest of the model by a transition from the start-state and a transition to the end-state. The purpose of the layer for node u is to generate with uniform probability all strings

- that are an ordered sequence of distinct nodes of G,
- that contain u, and
- where every node in the sequence is either u itself, or is connected to u by and edge in G.

Intuitively we can use these layers to count the number of nodes in a maximum clique of G, because the string listing the nodes in a clique can be generated by the layers corresponding to the nodes in the clique, and only by those layers.

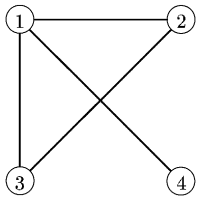

Fig. 2. A graph, $G = (\{1,2,3,4\}, \{\{1,2\},\{1,3\},\{1,4\},\{2,3\}\})$, of four nodes.

More precisely, the layer for node u consists of a single state that generates u with probability one, and a pair of states for every node v where $\{u,v\} \in E$, i.e. a pair of states for every node v connected to u by an edge. In each pair of states, one state generates v with probability one while the other state is a silent state allowing us to bypass the state generating v. The states in each layer are connected via silent states according to the ordering of V such that there is an equal probability of $1/2$ for choosing either of the two states in any pair. The transition probability from the start-state to the layer that corresponds to node u is set according to the degree of node u as $2^{\deg(u)}/\gamma$, where $\gamma = \sum_{v \in V} 2^{\deg(v)}$. Thus the probability of choosing the transition from the start-state to the layer of node u is proportional to the number of different paths through that layer. Fig. 3 shows the model corresponding to the graph shown in Fig. 2.

We observe that the probability of any run in M_G is $1/\gamma$. Now consider the probability, $P_{M_G}(s)$, of generating a string s. We observe that the probability of generating s is k/γ, where k is the number of runs that generate s, and that there is at most one run through each layer that generates s. If s is generated by a run through the layer for node u, then

- u must be in s, because the run that generates s must pass the single state in the layer for node u that generates u with probability one.
- u must have an edge to all other nodes that occur in s, because, by construction, the states in the layer for node u can only generate nodes (i.e. symbols $1, 2, \ldots, |V|$) that u has an edge to.

Hence, if a string s is generated with probability k/γ by the model M_G, it is generated by a run through the layers of k different nodes. Each of these nodes are in the string, and have an edge to all other nodes in the string, that is, they constitute a clique of size k in the graph G. On the other hand, if the graph G contains a clique of size k, the string that corresponds to the ordered list of the nodes in the clique is generated with probability k/γ by the model M_G. In conclusion this implies that we can compute the size of the maximum clique in the graph G within the time it takes to compute the probability of the consensus string generated by the hidden Markov model M_G plus the time polynomial in the size of G it takes to construct M_G, i.e. computing the probability of the consensus string is **NP**-hard.

The just established **NP**-hardness of the consensus string problem might not seem a major concern. Most, if not all, hidden Markov models for which we might want to find the consensus string are only approximate representations of the real-world phenomenon of interest. So just finding an approximate solution to the consensus string problem will usually be almost as good as solving it exactly. But the problem of finding the maximum clique size, which we have just reduced to the consensus string problem, is likely to be computationally hard even to approximate [7,10]. More precisely, for any $\varepsilon > 0$ it is hard to approximate the maximum clique size in G within $O(|V|^{1/2-\varepsilon})$ unless $\mathbf{P} = \mathbf{NP}$ and within $O(|V|^{1-\varepsilon})$

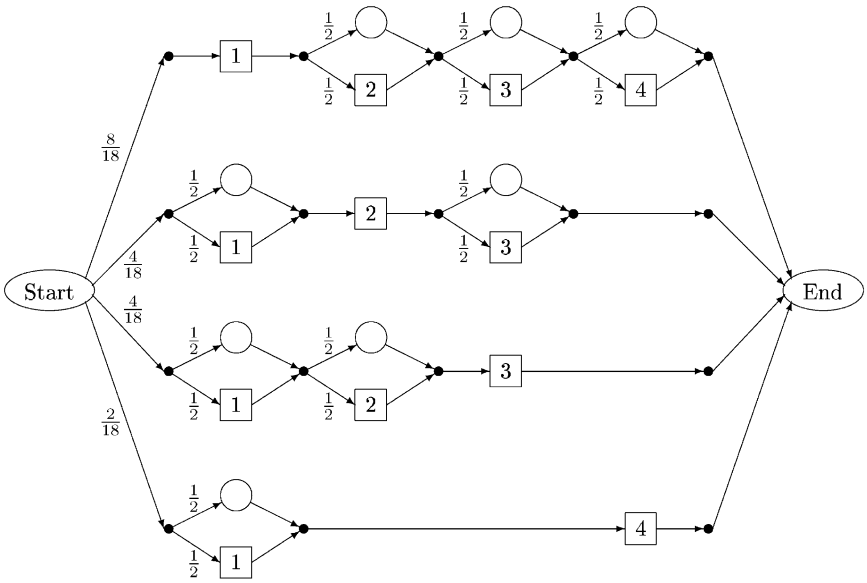

Fig. 3. The hidden Markov model M_G constructed from the graph G in Fig. 2. Silent states are drawn as circles, and non-silent states are drawn as rectangles with the symbol emitted with probability one written inside. The probability of any run in this model is $1/(\sum_{v \in V} 2^{\deg(v)})$.

unless **ZPP** = **NP**.[4] Specifically this means that it is **NP**-hard to approximate the maximum clique size for graphs within any constant.

This implies that if our reduction preserves approximability, then it is **NP**-hard even to approximate the probability of the consensus string of a hidden Markov model within any constant. By construction of M_G it follows that given an algorithm that approximates the probability of the consensus string of M_G within some factor, then we only need to multiply this probability with γ (and round up) to get an approximation of the size of the maximum clique in G to within the same factor. Hence, the existence of a polynomial time algorithm that guarantees to approximate the probability of the consensus string of a hidden Markov model within some constant factor would imply **P** = **NP**. If we translate the bounds on the approximability of maximum clique from [10] to the approximability of the probability of the consensus string of a hidden Markov model, we further obtain a bound of $O(n^{1/4-\varepsilon})$ unless **P** = **NP**, and a bound of $O(n^{1/2-\varepsilon})$ unless **ZPP** = **NP**, where n is the number of states in the hidden Markov model. This follows as the number of states in the hidden Markov model M_G is quadratic in the number of nodes of the graph G.

Before concluding this section, we will briefly consider a possible simplification of the construction of the M_G hidden Markov model, as a simplified construction makes the hardness result stronger. The current construction is already quite simple, e.g. M_G does not contain any cycles, and almost any reasonable restriction on allowed features is already obeyed by the constructed models. However, the alphabet of M_G is identical to V, the nodes of the graph for which we want to determine the

[4] **ZPP** is the class of problems that can be solved without error in expected polynomial time. That is, there is a randomized algorithm that computes the correct answer in expected polynomial time where the expectation is over the random choices of the algorithm (but independent of the input). That **ZPP** = **NP** is generally considered almost as unlikely as **P** = **NP**.

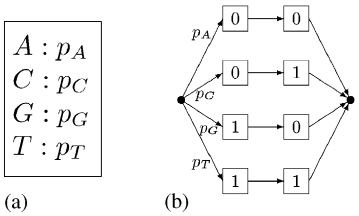

Fig. 4. An example of a sub-model that can replace a non-silent state when reducing an alphabet of arbitrary size to a binary alphabet. In the example the letters A, C, G, and T are replaced by the strings 00, 01, 10, and 11. (a) A non-silent state using the alphabets $\{A, C, G, T\}$. (b) The sub-model replacing it. All symbol emission and transition probabilities are 1 unless otherwise indicated.

maximum clique size. Thus our reduction, somewhat unrealistically, assumes an unbounded alphabet. It is however easy to extend the result to a binary alphabet by using a binary encoding of the alphabet; each symbol is replaced by a unique string over $\{0,1\}$ of length $\lceil \log |V| \rceil$. Usually this would mean replacing each non-silent state of the hidden Markov model with $|V|\lceil \log |V| \rceil$ new states—one stretch of $\lceil \log |V| \rceil$ states for each symbol, cf. Fig. 4. But in our reduction each non-silent state can emit only one possible symbol, cf. Fig. 3, so each non-silent state needs only to be replaced by $\lceil \log |V| \rceil$ new states. Thus the number of states in the modified model is $O(|V|^2 \log |V|)$. This means that the bounds for approximability stated above remain valid even for hidden Markov models with a binary alphabet as $|V|^2 \log |V| = O(|V|^{2+\varepsilon})$ for any $\varepsilon > 0$.

4. The labeling problem

A fundamental application of hidden Markov models is to annotate, or label, strings. For standard hidden Markov models, this usually consists of associating a state from the hidden Markov model to each symbol of the string. Most often this is done by using the Viterbi algorithm, cf. [17], i.e. by finding the most likely run in the hidden Markov model that generates the string and then annotating each symbol of the string with the state emitting it according to this run.

A class hidden Markov model is a two-track hidden Markov model where a run generates an annotation alongside the string. I.e. each non-silent state emits a symbol of the string on the string-track, and a labeling of that symbol on the annotation-track. Instead of being a problem of finding the most likely run, the labeling problem for class hidden Markov models becomes one of finding the labeling that is most likely when summed over all runs generating that labeling of the string, i.e. for a string s and a class hidden Markov model M we want to find

$$\arg\max_{l} \sum_{\pi} P_M(\pi, s, l). \tag{6}$$

The only difference between the labeling problem for class hidden Markov models and the consensus string problem is the added restriction of having to generate s on the string-track. This essentially just alters the problem to one of finding the most likely string, or labeling, of a specific length, namely a labeling of the same length as s. It requires only small modifications to adapt the hardness proof in Section 3 to the labeling problem for class hidden Markov models.

Let G be a graph for which we want to determine—or approximate—the maximum clique size. We construct a class hidden Markov model M'_G almost

identical to the standard hidden Markov model M_G constructed in Section 3, cf. Fig. 3. The entire structure of M'_G, i.e. the silent and non-silent states and the transition probabilities, is identical to M_G. The only change is in the behavior of the non-silent states, where we have to emit symbols on both the string- and annotation-track. On the string-track all non-silent states emit the same dummy symbol \$, and on the annotation-track the emissions mimic those of M_G. I.e. if a non-silent state in M_G emits the symbol u (i.e. emits u with probability one, as all non-silent states in M_G emit a particular symbol with probability one), then the corresponding non-silent state in M'_G emits \$ on the string-track and u on the annotation-track. It is evident that $P_{M_G}(s) = P_{M'_G}(\$^{|s|}, s)$ for all strings s. Hence, finding the most likely labeling of *any* string in M'_G immediately reveals the consensus string of M_G.

By the structure of M'_G we know that it cannot generate strings longer than $|V|$ where V is the set of nodes in G. Furthermore, we know that the only symbol emitted on the string-track of M'_G is \$. So the only strings that can be emitted on the string-track are $\$^i$ for $i \leqslant |V|$. Hence, given an algorithm $\mathscr{A}$ that determines—or just approximates—the probability of the most likely labeling of any particular string in M'_G in polynomial time, we can device a new algorithm that similarly determines, or approximates, the probability of the most likely labeling of any string in M'_G in polynomial time. This new algorithm would simply apply $\mathscr{A}$ to $\$^i$ for each $i \leqslant |V|$ in turn, and return the maximum probability found. So the hardness results established for the consensus string problem also apply to the labeling problem for class hidden Markov models.

5. Heuristics for finding the consensus string

A simple method for obtaining an estimate for the consensus string of a hidden Markov model M, i.e. a string that is likely to have a high probability in M, is to find the string generated by the most likely run of the model. This is very similar to using the Viterbi algorithm for annotating a string. Determining the most likely run is like finding the shortest path in a weighted graph, and can be solved efficiently using standard graph algorithms, cf. [5, Chapter 25].

It is however well known that a hidden Markov model possesses a lot more information about a string than just its most probable path through the model. Most often the string can be generated by a multitude of other paths that might have a probability just slightly lower than the probability of the most probable path. It seems plausible that better estimates of the consensus string might be obtained by using this information. One approach would be to construct an estimate of the consensus string in a greedy fashion. Assume that we have already decided on the prefix x, and want to determine what symbol $\sigma \in \Sigma$ to append to x. The perfect greedy choice would be the symbol σ that maximizes

$$\max_{s \in \Sigma^*} P_M(x\sigma s), \tag{7}$$

as this would guarantee that if x is a prefix of the consensus string then so is $x\sigma$. I.e. the greedy approach would yield the consensus string if we start with x being the empty string. Unfortunately, computing the maximum in Eq. (7) is essentially what we have just established the hardness of in the previous section. Hence, a greedy method based on Eq. (7) is most likely inefficient.

To circumvent this inefficiency we present four methods for deciding what symbol to append to the current estimate of the consensus string in the greedy construction. Each method has the advantage that the symbol can be computed efficiently.

However, this also implies that the constructed estimate of the consensus string is not necessarily the true consensus string. In the rest of this section we use n to denote the number of states in M, and m to denote the number of transitions in M.

5.1. Method 1

If we take all partial runs from the start-state to a given state q which generate the same string $x\sigma$, and extend them all with the same partial run from q to the end-state, then we get a number of runs all generating the same string, a string with prefix $x\sigma$. Hence, we know that

$$\max_{q \in M} \sum_{\pi_{(s \to q)}} P_M(\pi_{(s \to q)}, x\sigma) \max_{r_{(q \to e)}} P_M(r_{(q \to e)}) \leqslant \max_{s \in \Sigma^*} P_M(x\sigma s), \tag{8}$$

where $\pi_{(s \to q)}$ denotes a path form the start-state to state q in M, and $r_{(q \to e)}$ denotes a partial run from state q to the end-state in M not emitting a symbol from the initial state q, even if q is non-silent. I.e. we have lower bounded the probability of Eq. (7) by the left-hand side in Eq. (8). This implies a possible method for deciding what symbol to append to the current estimate of the consensus string: choose the symbol σ that maximizes the left-hand side probability in Eq. (8).

The advantage of replacing Eq. (7) with the left-hand side of Eq. (8) for deciding the symbol σ with which to extend x, is that the latter can be computed efficiently. The sum, $\sum_{\pi_{(s \to q)}} P_M(\pi_{(s \to q)}, x\sigma)$, is the total probability of all runs emitting $x\sigma$ and ending in state q. But this is exactly the entity, $f_q(x\sigma)$, computed for all states q by the forward algorithm when computing the probability of string $x\sigma$ in model M. The forward algorithm works in an incremental fashion, only needing $f_q(x)$ for all states q in M to compute $f_q(x\sigma)$ for all states q in M. Hence, we do not have to start from scratch when computing the $f_q(x\sigma)$ values. We can reuse the $f_q(x)$ values from the previous round of the greedy construction to compute the $f_q(x\sigma)$ values in time $O(m)$. The other term, $\max_{r_{(q \to e)}} P_M(r_{(q \to e)})$, is the probability of the most likely run from state q to the end-state which does not emit any symbol from the initial state q. As mentioned earlier, these values can be computed by standard graph algorithms, cf. [5, Chapter 25], in time $O(n \log n + m)$. Furthermore, they do not change as we extend the string but only have to be computed once. Hence, the total time requirement for constructing a string of length l by this method is $O(n \log n + ml|\Sigma|)$.

One advantage of this method is that we for each candidate string x compute $f_e(x)$, i.e. the probability of x in M. Hence, we can keep track of the probabilities of the consecutive candidate strings and store the best seen so far as we proceed to construct an ever longer string. Furthermore, as we will discuss in detail for Method 2, the maximum probability in M of any string having x as a proper prefix is upper bounded by $\sum_{q \in M,\, q \text{ non-silent}} f_q(x)$, a sum that is readily computed in time $O(n)$ when we know the values of $f_q(x)$ for all $q \in M$. Thus, we can compute this sum and when it becomes smaller than the maximum probability for a candidate string seen so far, we know that any further extensions of the candidate string will not lead to a string more likely than the best seen so far. A drawback of this method is that it might put to large an emphasis on $\max_{r_{(q \to e)}} P_M(r_{(q \to e)})$, i.e. the maximum probability partial run from state q to the end-state. Thus it might yield a result similar or identical to the result of the much simpler approach based on finding the string generated by the most likely run, i.e. the Viterbi approach, outlined in the first paragraph of this section. A method taking into account all partial runs starting in the start-state and generating $x\sigma$ might thus be more desirable.

5.2. Method 2

The maximum over a set of non-negative numbers is never larger than the sum of the same set of numbers. Hence,

$$\sum_{s\in\Sigma^*} P_M(x\sigma s) \geqslant \max_{s\in\Sigma^*} P_M(x\sigma s), \tag{9}$$

i.e. we have upper bounded the probability of Eq. (7) by the left-hand side in Eq. (9). Similar to Method 1, we can design a method for choosing the next symbol σ of the candidate string based on maximizing the left-hand side probability in Eq. (9).

When replacing Eq. (7) with the left-hand side of Eq. (9) for deciding the symbol σ with which to extend x, we again obtain a method that is efficiently computable. The sum, $\sum_{s\in\Sigma^*} P_M(x\sigma s)$, can be recognized as the probability of all strings having $x\sigma$ as prefix. Strings having $x\sigma$ as prefix are all generated by runs that can be separated into an initial partial run generating $x\sigma$ followed by a partial run generating any string and ending in the end-state. Hence,

$$\sum_{s\in\Sigma^*} P_M(x\sigma s) = f_e(x\sigma) + \sum_{q\in M,\, q \text{ non-silent}} f_q(x\sigma) \sum_{r_{(q\to e)}} P_M(r_{(q\to e)}), \tag{10}$$

where $r_{(q\to e)}$ again denotes a partial run from state q to the end-state in M not emitting a symbol from the initial state q, even if q is non-silent. The reason for restricting q to be a non-silent state is to avoid including the same run multiple times in the summation. We decompose a run generating a string with prefix $x\sigma$ into two partial runs where the first partial run generates $x\sigma$ and the second partial run generates the rest of the string. This decomposition can be done in several ways, if the state emitting the last symbol of $x\sigma$ is followed by a sequence of one or more silent states. Requiring the last state of the partial run generating $x\sigma$ to be non-silent ensures a unique decomposition into two partial runs.

The sum $\sum_{r_{(q\to e)}} P_M(r_{(q\to e)})$ is the probability of eventually reaching the end-state from state q. As we have assumed that the probability of an infinite run in a hidden Markov model is zero, we will eventually reach the end-state no matter what state we happen to be in M. I.e. $\sum_{r_{(q\to e)}} P_M(r_{(q\to e)}) = 1$ for all $q\in M$. Hence we can compute the desired probability by just summing the $f_q(x\sigma)$ values of the forward algorithm from all non-silent states and the end-state. In fact, one might choose only to sum the $f_q(x\sigma)$ values for only the non-silent states as this gives the total probability of all strings having $x\sigma$ as a proper prefix; $f_e(x\sigma)$, on the other hand, is the probability of the specific string $x\sigma$ in M. As discussed for Method 1, the $f_q(x\sigma)$ values can be computed in time $O(m)$ once we know the $f_q(x)$ values. Thus, the total time requirement for constructing a string of length l by this method is $O(ml|\Sigma|)$.

As for Method 1, we can keep track of the most likely candidate string constructed so far, and stop further extensions of the candidate string when the total probability of all strings having the current candidate string as a proper prefix drops below the probability of the most likely candidate string constructed so far. Moreover, this method incorporates information from all runs generating strings with the current candidate string as prefix. Unfortunately, it has its drawbacks too, as it might get "trapped" in a high probability cycle. E.g. if the major contribution to $f_e(x) + \sum_{q\in M,\, q \text{ non-silent}} f_q(x)$ is from a single state p with $a^M_{p,p} \approx 1$ (this could for example be a state emitting bases in an intron region in a gene prediction hidden Markov model), then we will choose the σ that is most likely to be emitted from p and the major contribution to $\sum_{q\in M} f_q(x\sigma)$ will again be from p. I.e. we will keep choosing the next symbol based on the symbol emission probabilities of p, which in turn makes $f_p(x\sigma)$ ever more predominant for the choice of the next symbol.

5.3. Method 3

A possible way to circumvent the above drawback is to consider the approach suggested in [13]. The problem addressed in [13] is the labeling problem for class hidden Markov models, cf. Section 4; but this is essentially the problem of finding the most likely string of a prespecified length, e.g. an algorithm for solving either the consensus string problem or the labeling problem, in general, only requires minor modifications to be used for solving the other problem. Instead of maintaining a global most likely candidate string, the approach suggested in [13] is to maintain a most likely candidate string x_q for every state q. In each round we for each state q choose a new string for that state as a string x_p from the previous round extended with a symbol σ maximizing

$$\max_{\sigma \in \Sigma, p \in M} f_q(x_p \sigma). \tag{11}$$

I.e. for each state we find the best string that can be obtained from extending any of the strings of the previous round with any of the symbols of the alphabet. This approach is quite similar to Method 1. But instead of choosing a candidate from a specific state we keep the best candidates from all states.

A drawback of this approach is that we basically have to keep track of n concurrent forward algorithm computations, one for each candidate string maintained. This increases as well the time as the space requirements with a factor of n compared to the two previous methods (when ignoring the time requirement of $O(m + n \log n)$ of the shortest paths computations of Method 1). Hence, the total time requirement for constructing a string of length l by this method is $O(nml|\Sigma|)$. Another drawback is that it, similar to Method 1, only focuses on locally good strings. A string that can be generated by numerous very diverse runs, and thus has a high overall probability, will not be discovered unless each of its prefixes is the best choice for a specific state in M.

5.4. Method 4

In [15] we show how to compute the co-emission probability $\sum_{s \in \Sigma^*} P_M(s) P_{M'}(s)$ for two left–right hidden Markov models M and M' using a method which closely mimics the forward algorithm; for each pair of states $q \in M$ and $q' \in M'$ the sum of the probabilities of all pairs of partial runs generating identical strings and ending in q and q', respectively, is computed. The time complexity of the method is $O(mm')$, where m is the number of transitions in M and m' is the number of transitions in M'. We can generalize the method to compute the co-emission probability $\sum_{s \in \Sigma^*} P_M(s) P_{M'}(s)$ for two general hidden Markov models. The generalization is based on solving a system of linear equations and has time complexity $O((nn')^3)$, where n is the number of states in M and n' is the number of states in M'. Both methods can be generalized to compute the co-emission probability $\sum_{s \in \Sigma^*} \prod_{i=1}^{k} P_{M_i}(s)$ for k hidden Markov models. This will increase time requirements to $O(\prod_{i=1}^{k} m_i)$ for left–right hidden Markov models with m_i transitions in model M_i, and to $O(\prod_{i=1}^{k} n_i^3)$ for general hidden Markov models with n_i states in model M_i. As for the forward algorithm, we can design a similar backward algorithm where for each k-tuple of states $q^{(i)} \in M_i$ we compute the sum of the probabilities of all k-tuples of partial runs starting in states $q^{(i)}$ and emitting identical strings. We will denote this "backward" probability $B(q^{(1)}, \ldots, q^{(k)})$ and for convenience assume that the partial runs summed over do not include a symbol emission at the initial $q^{(i)}$ states.

Recall that if we can determine $\max_{s\in\Sigma^*} P_M(x\sigma s)$, cf. Eq. (7), then we can make the perfect choice of what symbol σ to append to the current estimate of the consensus string. Unfortunately, as shown in Section 3, it is unlikely that there is a feasible method for computing $\max_{s\in\Sigma^*} P_M(x\sigma s)$. However, it is well known that

$$\max_{s\in\Sigma^*} P_M(x\sigma s) = \lim_{a\to\infty} \sqrt[a]{\sum_{s\in\Sigma^*} P_M(x\sigma s)^a}. \tag{12}$$

Let us consider the right-hand side of Eq. (12). For any $k\in\mathbf{N}$ we can split the sum $\sum_{s\in\Sigma^*} P_M(x\sigma s)^k$ into a sum over all k-tuples of runs emitting identical strings with prefix $x\sigma$. These can in turn be split into a k-tuple of partial runs all emitting $x\sigma$ starting in the start-state and ending in non-silent states $q^{(1)}, \ldots, q^{(k)}$, followed by a k-tuple of partial runs starting in the states $q^{(1)}, \ldots, q^{(k)}$, but not emitting any symbol from the initial states $q^{(i)}$, and ending in the end-state. Hence,

$$\sum_{s\in\Sigma^*} P_M(x\sigma s)^k = \sum_{q^{(1)},\ldots,q^{(k)}\in M} B(q^{(1)}, \ldots, q^{(k)}) \prod_{i=1}^{k} f_{q^{(i)}}(x\sigma). \tag{13}$$

I.e. we can approximate Eq. (7) by $\sqrt[k]{\sum_{s\in\Sigma^*} P_M(x\sigma s)^k}$, and use this to decide what symbol to append to the current estimate of the consensus string. The $B(q^{(1)}, \ldots, q^{(k)})$ do not change as we extend our candidate string, so we only need to compute them once. So for each extension with an extra symbol, we only need to update the forward algorithm computations for each choice of $\sigma\in\Sigma$, compute the right-hand side of Eq. (13), and remember the maximum value encountered. Hence, the total time requirement for constructing an estimate of length l by this method is $O(m^k + |\Sigma| l(n^k + m))$ for left–right models, and $O(n^{3k} + |\Sigma| l(n^k + m))$ for general models. By increasing k we should obtain better and better estimates for the consensus string. Indeed, by Eq. (12) we know that by choosing k sufficiently large we will find the consensus string of the hidden Markov model. Unfortunately, a "sufficiently large" k will result in prohibitive time and space requirements. One can observe that Method 2 is a special case of this method, obtained by choosing $k = 1$ and observing that $B(q^{(1)}) = 1$ for all $q^{(1)}\in M$.

5.5. Concluding remarks

As stated above, algorithms for finding the consensus string of a standard hidden Markov model, and algorithms for finding the most probable labeling of a sequence in a class hidden Markov model are often interchangeable. All the algorithms presented in this section can be modified to handle the labeling problem for class hidden Markov models in a straightforward manner, in fact, the algorithm of Krogh [13] was designed for the labeling problem. In the labeling problem, however, we are looking for a string of a fixed length, the length of the sequence to be labeled. For all the algorithms presented in this section, except for that of Krogh [13], a straightforward modification will result in an increase in both time and space requirements of a factor $|s|$, where s is the sequence to be labeled. This is due to the fact that the emitted labeling should be of the same length as s. Hence we need to restrict partial runs connecting specific states to the end-state to generate labelings of specific lengths. Utilizing techniques like repeated squaring [5, Chapter 26.1] and checkpoints [9] can in many cases improve the efficiency, though.

If an exact solution is required for a hard optimization problem, one standard approach is to apply branch and bound technique. The branch and bound technique attempts to reduce the size of the search space by eliminating subsets that cannot

contain the optimal solution. This is done by finding a lower bound for the value of the optimal solution—here the probability of the consensus string—and try to establish upper bounds for the value of any solution from a given subset. If this upper bound is smaller than the lower bound on the value of the optimal solution, we can safely ignore solutions from that subset. Eqs. (8) and (9) provide us with a lower and an upper bound on the probability of any string having a specific prefix, respectively. Hence, we could design a branch and bound algorithm that uses Eq. (8) to find increasingly better lower bounds for the probability of the consensus string—possibly starting with the probability of the string obtained by the Viterbi approach as an initial estimate—and uses Eq. (9) to eliminate subsets consisting of all strings with a given prefix as candidates for the consensus string. As the bound obtained from Eq. (9) is the total probability of all strings with a given prefix, the bound might be far from the true maximum probability of a string with that prefix. Hence, the bound obtained from Eq. (9) may often fail in eliminating subsets not containing the consensus string. However, as

$$\sqrt[a]{\sum_{s\in\Sigma^*} P_M(xs)^a} \geqslant \max_{s\in\Sigma^*} P_M(xs) \tag{14}$$

for all $a\in\mathbf{R}_+$, we can use $\sqrt[k]{\sum_{s\in\Sigma^*} P_M(xs)^k}$ to upper bound the probability of all strings with prefix x. And as $\sqrt[k+1]{\sum_{s\in\Sigma^*} P_M(xs)^{k+1}} < \sqrt[k]{\sum_{s\in\Sigma^*} P_M(xs)^k}$ unless $\sqrt[k]{\sum_{s\in\Sigma^*} P_M(xs)^k} = \max_{s\in\Sigma^*} P_M(xs)$ we know this bound will improve for increasing k. Unfortunately, the time required to compute $\sqrt[k]{\sum_{s\in\Sigma^*} P_M(xs)^k}$ will increase with increasing k as discussed for Method 4.

A serious theoretical objection to the heuristics proposed in this section is that the produced estimate might not be a string of high probability in the model, but only a string that is likely to be a prefix of a generated string. A natural next step is thus to carry out experiments to see whether the problems identified are likely to occur for hidden Markov models representing real-world phenomena, and to examine whether the proposed heuristics are improvements compared to just using the string of the most likely run as an estimate for the consensus string. As reported in [13], this is the case for at least Method 3 when used for gene finding.

6. Comparison of hidden Markov models

One of the common tasks in computational biology is that of comparing data. Usually, hidden Markov models are considered tools for analyzing data. But we may also simultaneously view hidden Markov models as a compact representation of the set of biological sequences recognized by the model, i.e. a hidden Markov model can itself be considered as data. It is thus interesting, both from a theoretical and a practical viewpoint, to investigate how to compare two hidden Markov models, i.e. how to compare the probability distributions described by the two models. In [15] we describe how to compute the L_2-distance between two models in polynomial time. In this section we investigate the complexity of comparison of hidden Markov models by two other norms commonly encountered, the L_1-norm and the L_∞-norm.

The L_∞-norm between two probability distributions, P and Q, defined on the same set, Ω, is defined as $\|P-Q\|_\infty = \max_{s\in\Omega} |P(s)-Q(s)|$. It measures the largest difference in probabilities we can obtain for any possible single observation. A possible application of comparing hidden Markov models under the L_∞-norm is to

evaluate whether two models differ significantly in probabilities for any strings, i.e. whether it is likely that two models for closely related families are able to discriminate between strings of the two families. However, it is easy to see that we can use the L_∞-norm of a hidden Markov model M to determine the probability of the consensus string of M. We simply extend the alphabet of M with an extra symbol \$, and construct a new model M' that is identical to M except that it puts a \$ at the start of all strings, i.e. before entering the copy of M it has a state that always emits the symbol \$. We observe that M and M' cannot generate identical strings, as no state in M can emit \$, and that $P_M(s) = P_{M'}(\$s)$ for all strings s. This implies that the probabilities of the consensus strings in the two models M and M' are identical. Thus $\max_{s\in\Omega} |P_M(s) - P_{M'}(s)| = \max_{s\in\Sigma^*} P_M(s)$. Hence, computing the L_∞-norm between two hidden Markov models is as hard as computing the probability of the consensus string of a hidden Markov model.

A common way to compare two probability distributions, P and Q, defined on the same set, Ω, is to consider the *variation distance*, see e.g. [6], defined as $\|P - Q\| = \max_{\mathscr{A}\subseteq\Omega} |P(\mathscr{A}) - Q(\mathscr{A})|$. Though it resembles the L_∞-norm between probability distributions—just maximizing over subsets instead of single elements of Ω—it is actually closely related to another well-known norm. It is a standard observation that the variation distance between two probability distributions is half the L_1-distance between the two probability distributions, i.e. $\|P - Q\| = \frac{1}{2}\|P - Q\|_1$. An efficient computation of the L_1-distance is thus of interest when comparing probability distributions. In general, the L_k-norm between two hidden Markov models M and M' over the same alphabet Σ is $\|P_M - P_{M'}\|_k = \sqrt[k]{\sum_{s\in\Sigma^*} |P_M(s) - P_{M'}(s)|^k}$. Unfortunately, as we will prove in this section, computing the L_1-distance between two probability distributions given by hidden Markov models over the same alphabet is **NP**-hard. Similar to the hardness proof in Section 3, we will prove the hardness by a reduction from the problem of computing the size of a maximum clique in an undirected graph.

Let $G = (V, E)$ be the graph of interest and M_G the hidden Markov model constructed from this graph as described in Section 3. Recall that every string generated by M_G is a subsequence of $1 \cdot 2 \cdot \ldots \cdot |V|$, and has probability i/γ for some $i \in \{0, 1, \ldots, |V|\}$, where $\gamma = \sum_{v\in V} 2^{\deg(v)}$. Let a_i be the number of strings which M_G generates with probability i/γ. If we know the maximum k such that $a_k \neq 0$, we know that the probability of the consensus string in M_G is k/γ, and by the result of the previous section, that the maximum clique size of G is k.

We will show that if we can compute the L_1-distance, $\|M - M'\|_1$, between two hidden Markov models, M and M', in polynomial time, then we can also determine the maximum k, where $a_k \neq 0$, in polynomial time. Fig. 5 shows a hidden Markov model $M_{|V|}$ that generates all subsequences of $1 \cdot 2 \cdot \ldots \cdot |V|$ with equal probability of $1/2^{|V|}$. To determine a_i, for $i = 0, 1, \ldots, |V|$, we perform $|V| + 1$ comparisons under the L_1-norm between slightly modified versions of M_G and $M_{|V|}$, where the ith comparison is performed as one of the following two cases:

Case 1: If $(i2^{|V|})/\gamma \leqslant 1$, we construct a model $M^i_{|V|}$ by rescaling model $M_{|V|}$ using the rescaling trick illustrated in Fig. 6 for $x = (i2^{|V|})/\gamma$, and perform the comparison:

$$\begin{aligned}\|M_G - M^i_{|V|}\|_1 &= \sum_{s\in\Sigma^*} |P_{M_G}(s) - P_{M^i_{|V|}}(s)| \\ &= |P_{M^i_{|V|}}(\$)| + \sum_{s\in\{0,1,\ldots,|V|\}^*} |P_{M_G}(s) - P_{M^i_{|V|}}(s)|\end{aligned}$$

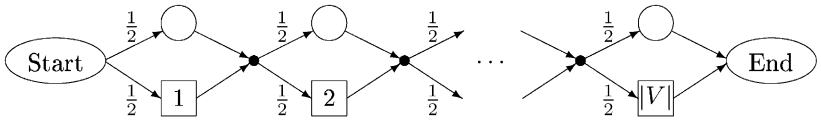

Fig. 5. A hidden Markov model M that generates any subsequence of $1 \cdot 2 \cdot \ldots \cdot |V|$ with equal probability $1/2^{|V|}$.

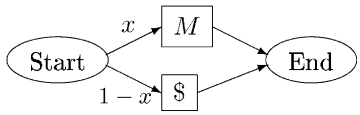

Fig. 6. Rescaling the probabilities of a hidden Markov model M by a factor of x. The new model will generate sequences of the old model with a probability that is reduced by a factor of x and a special sequence just consisting of a dummy symbol \$ with probability $1-x$.

$$= \left(1 - \frac{i2^{|V|}}{\gamma}\right) + \sum_{j=0}^{|V|} a_j \left| \frac{j}{\gamma} - \left(\frac{1}{2^{|V|}} \frac{i2^{|V|}}{\gamma}\right) \right|$$
$$= \left(1 - \frac{i2^{|V|}}{\gamma}\right) + \frac{1}{\gamma} \sum_{j=0}^{|V|} a_j |i-j|.$$

The purpose of the rescaling is to make all subsequences of $1 \cdot 2 \cdot \ldots \cdot |V|$ have probability i/γ in $M^i_{|V|}$.

Case 2: If $(i2^{|V|})/\gamma > 1$, we construct a model M^i_G by rescaling model M_G using the rescaling trick illustrated in Fig. 6 for $x = \gamma/(i2^{|V|})$, and perform the comparison:

$$\begin{aligned} \|M^i_G - M_{|V|}\|_1 &= \sum_{s \in \Sigma^*} |P_{M^i_G}(s) - P_{M_{|V|}}(s)| \\ &= |P_{M^i_G}(\$)| + \sum_{s \in \{0,1,\ldots,|V|\}^*} |P_{M^i_G}(s) - P_{M_{|V|}}(s)| \\ &= \left(1 - \frac{\gamma}{i2^{|V|}}\right) + \sum_{j=0}^{|V|} a_j \left| \left(\frac{j}{\gamma} \frac{\gamma}{i2^{|V|}}\right) - \frac{1}{2^{|V|}} \right| \\ &= \left(1 - \frac{\gamma}{i2^{|V|}}\right) + \frac{1}{i2^{|V|}} \sum_{j=0}^{|V|} a_j |i-j|. \end{aligned}$$

The purpose of the rescaling is to make all strings that have probability i/γ in M_G have probability $1/2^{|V|}$ in M^i_G, i.e. the same probability as all subsequences of $1 \cdot 2 \cdot \ldots \cdot |V|$ have in $M_{|V|}$.

In summary the result, $D^i(M_G, M_{|V|})$, of the ith comparison is

$$1 - \min\left\{\frac{i2^{|V|}}{\gamma}, \frac{\gamma}{i2^{|V|}}\right\} + c_i \sum_{j=0}^{|V|} a_j |i-j|, \tag{15}$$

where c_i is either $1/\gamma$ or $1/(i2^{|V|})$, cf. cases 1 and 2. Since we have $|V|+1$ variables—the a_i's—and perform $|V|+1$ comparisons, this leaves us with a linear equation

system of the form

$$\begin{bmatrix} & \vdots & \\ \cdots & |i-j| & \cdots \\ & \vdots & \end{bmatrix} \begin{bmatrix} \vdots \\ a_j \\ \vdots \end{bmatrix} = \begin{bmatrix} \vdots \\ \frac{1}{c_i}(D^i(M_G, M_{|V|}) - 1 + \min\{\frac{i2^{|V|}}{\gamma}, \frac{\gamma}{i2^{|V|}}\}) \\ \vdots \end{bmatrix}. \tag{16}$$

Since the $d \times d$ matrix given by the i,jth entry being $|i-j|$ is invertible, with inverse

$$\begin{bmatrix} \frac{2-d}{2(d-1)} & \frac{1}{2} & 0 & \cdots & 0 & \frac{1}{2(d-1)} \\ \frac{1}{2} & -1 & \frac{1}{2} & \ddots & & 0 \\ 0 & \frac{1}{2} & -1 & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ & & & -1 & \frac{1}{2} & 0 \\ 0 & & & \frac{1}{2} & -1 & \frac{1}{2} \\ \frac{1}{2(d-1)} & 0 & \cdots & 0 & \frac{1}{2} & \frac{2-d}{2(d-1)} \end{bmatrix}, \tag{17}$$

the above linear equation system has a unique solution which can be computed efficiently using standard techniques. Hence, if we in polynomial time can compute the result of the ith comparison, $D^i(M_G, M_{|V|})$, that is, if we can compare two hidden Markov models under the L_1-norm, then we can also in polynomial time compute a_i, for $i = 0, 1, \ldots, |V|$, by solving the above linear equation system. Hence, computing the L_1-distance between two hidden Markov models is as hard as computing the size of a maximum clique in an undirected graph.

In the above reduction, we do not directly relate the L_1-distance between two models to the probability of the consensus string of one (or both) of the models. Hence, the approximation hardness is not preserved by the reduction, and the hardness result obtained is only for the computation of the exact L_1-distance. In the rest of this section we examine the technique of reduction in further details.

Assume that we want to compute the distance between the probability distributions of the two models M and M' using a distance measure D, which is defined as a sum of point-wise comparisons, i.e. $D(P_M, P_{M'}) = \sum_{s \in \Sigma^*} d(P_M(s), P_{M'}(s))$. If we apply the above technique to prove the hardness of computing the D-distance between two hidden Markov models, we can set up a system of linear equations similar to Eq. (16) as the result of the ith comparison under D is

$$1 - \min\left\{\frac{i2^{|V|}}{\gamma}, \frac{\gamma}{i2^{|V|}}\right\} + \sum_{j=0}^{|V|} a_j d(c_i i, c_i j). \tag{18}$$

Whether this allows us to deduce anything about the hardness of comparing two hidden Markov models under D depends on whether the resulting linear equation system, with matrix M defined by $M_{ij} = d(c_i i, c_i j)$, can be solved with sufficient precision to determine the maximum clique size of G in time polynomial in the size of G. As an example, let us consider the L_k-norm, where k is an even integer. In general,

for the L_r-norm we have

$$(||P_{M_{|V|,i}}, P_{M_{G,i}}||_r)^r = \left(1 - \min\left\{\frac{i2^{|V|}}{\gamma}, \frac{\gamma}{i2^{|V|}}\right\}\right) + \sum_{j=0}^{|V|} a_j |c_i i - c_i j|^r$$
$$= \left(1 - \min\left\{\frac{i2^{|V|}}{\gamma}, \frac{\gamma}{i2^{|V|}}\right\}\right) + c_i^r \sum_{j=0}^{|V|} a_j |i-j|^r. \quad (19)$$

Hence, when considering the L_k-norm for an even integer k, the matrix M of the resulting system of linear equations has entries $M_{ij} = |i-j|^k = (i-j)^k$. Since any $m \times m$ matrix A defined by $A_{ij} = (x_i + y_j)^k$ is singular if $m > k+1$, cf. Appendix A, the system of linear equations defined by $M = (i-j)^k$, for an even integer k, cannot be solved for a unique solution if $|V| > k+1$, and we fail to prove the **NP**-hardness of comparing two hidden Markov models under the L_k-norm for an even integer k. This failure is no surprise because the algorithmic technique for comparing two hidden Markov models under the L_2-norm described in [15] can easily be extended to compare models under the L_k-norm, k a fixed even integer, in time $O(n^{3k})$ where n is the number of states in the two models. The algorithmic technique of Lyngsø et al. [15] cannot be applied to the computation of any other L_r-norm than those where r is an even integer, however.

Based on the **NP**-hardness of comparing two models under the L_1-norm proved above, and that the L_r-norms where r is an even integer stand out as the only ones where the absolute value operation can be ignored, we conjecture that it is **NP**-hard to compare two hidden Markov models under any L_r-norm where r is not an even integer. This claim immediately follows if we, for a fixed r, which is not an even integer, can solve a linear equation system as in Eq. (16) with an $m \times m$ matrix M with entries $M_{ij} = |i-j|^r$ in time polynomial in m.

Similar to the L_k-norm, for k an even integer, we can rule out the Kullback–Leibler divergence, or relative entropy, as a case where the technique used in the hardness proof for the L_1-norm is useful. Here the matrix M of the linear equation system can be defined by $M_{ij} = i \log \frac{i}{j} = i(\log i - \log j)$. Setting $x_i = \log i$ and $y_j = \log j$, by Proposition 1 of Appendix A the $m \times m$ matrix M' defined by $M'_{ij} = \log i - \log j$ is singular if $m > 2$. Hence, so is any matrix obtained by multiplying one or more rows of M' by scalars.

7. Discussion

In Section 3 we proved the **NP**-hardness of finding the consensus string of a hidden Markov, and the related decision problem of determining whether the probability of the consensus string is at least some target value κ. But we did not show that the decision problem is **NP**-complete, i.e. that it is in the complexity class **NP**. The standard approach to prove this would be to guess a string s and verify that $P_M(s) \geqslant \kappa$ for this guess. However, this only works if we can guarantee that we only need to make a polynomial number of guesses (each among a constant number of choices), i.e. if we can guarantee that the length of the consensus string is only polynomial in the size of the hidden Markov model. In Appendix B we investigate this problem. We show that in general the length of the consensus string can be super-polynomial in the size of the model. However, for left–right models with transition probabilities that fits into a word of size $O(\log n)$ bits, where n is the size of the model, we can show that the length of the consensus string is only polynomial in the size of the model. We further show that the model M_G, described in Section 3,

can be transformed into a model adhering to these restrictions. Hence, the decision version of the consensus string problem for left–right models with probabilities bounded to fit in $O(\log n)$ bits is **NP**-complete.

When choosing how to compare two probability distributions, one important consideration is how efficiently a given distance can be computed. In [1] a number of commonly used distance measures between probability distributions are listed: the χ^2, variation, quadratic, Hellinger and Kullback–Leibler distances. In [15] we show how to compute the quadratic distance between the probability distributions of two hidden Markov models in polynomial time. The Hellinger distance, i.e. the distance in Euclidean space between the two probability distributions after they have been normalized to 1, can also be computed in polynomial time based on the methods in [15], as we show how to compute the angle in Euclidean space between the two probability distributions when interpreted as infinite dimensional vectors. In this paper we have proved that the variation distance is **NP**-hard to compute, and furthermore that the L_∞-distance is **NP**-hard even to approximate. Furthermore, we briefly mentioned that the L_k-distance can be computed in polynomial time if k is an even integer. To our knowledge, the complexity of comparing the probability distributions of two hidden Markov models under the χ^2 distance, the Kullback–Leibler distance, and the L_k-distance for any k not 1, ∞, or an even integer, remains open problems, though a heuristic for approximating the Kullback–Leibler distance was proposed in [18].

Acknowledgments

We would like to thank Peter Bro Miltersen for bringing our attention to the problems discussed in this paper. We furthermore acknowledge the help of Ole Østerby and Kevin Karplus in preparing this manuscript. Finally, we would like to express our gratitude to an anonymous referee for bringing our attention to the labeling problem, pointing out that our hardness results could easily be extended to this problem as well, and for various other suggestions improving on this paper.

Appendix A

The following result might be a well-known result in linear algebra, but as we have failed to find any references mentioning it, we prove it here for completeness.

Proposition 1. *Let A be an $m \times m$ matrix. If there exists $k \in \mathbf{N}$, $k < m-1$ and sets $\{x_i\}_{1 \leqslant i \leqslant m}$ and $\{y_j\}_{1 \leqslant j \leqslant m}$ such that $A_{ij} = (x_i + y_j)^k$ for all $1 \leqslant i,j \leqslant m$, then A is singular.*

Proof. First we observe that if there exists $i \neq j$ such that $x_i = x_j$ then we are through, as rows i and j of A are then identical. So assume from now on that $i \neq j \Rightarrow x_i \neq x_j$. Furthermore, assume that A is invertible. We will prove that this assumption leads to a contradiction.

Let b be a vector chosen such that there is no polynomial of degree at most k passing through all the points $(x_1, b_1), (x_2, b_2), \ldots, (x_m, b_m)$. That we can find such a b follows because a polynomial of degree k is uniquely determined by its value in $k+1$ points. Hence, if we choose arbitrary values for $b_1, b_2, \ldots, b_{m-1}$, then there is a unique polynomial of degree at most k passing through the points (x_1, b_1), $(x_2, b_2), \ldots, (x_{k+1}, b_{k+1})$. So it now suffices to choose a b_m that disagrees with the

value of this polynomial in x_m. Furthermore, let $u = A^{-1}b$, i.e. $Au = b$. But

$$[Au]_i = \sum_{j=1}^{m} A_{ij}u_j = \sum_{j=1}^{m} (x_i + y_j)^k u_j = \sum_{j=1}^{m} \sum_{l=0}^{k} \binom{k}{l} x_i^l y_j^{k-l} u_j = \sum_{l=0}^{k} c_l x_i^l, \qquad (20)$$

where $c_l = \binom{k}{l} \sum_{j=1}^{m} y_j^{k-l} u_j$ is independent of i. Thus Au are the values of a polynomial of degree at most k at $x_1, x_2, \ldots, x_m$. This contradicts $Au = b$ and the choice of b as not being the values of a polynomial of degree at most k at $x_1, x_2, \ldots, x_m$. □

Appendix B

In this appendix we investigate bounds on the length of the consensus string of a model. More precisely, we will try to develop necessary and sufficient conditions for when we can guarantee that the length of the consensus string of a given hidden Markov model is bounded by a polynomial in the number of states of the model. The purpose of this is to establish a natural variant of the decision version of the consensus string problem, i.e. given a model M and a target probability κ is there a string s with $P_M(s) \geqslant \kappa$, that is **NP**-complete. The reason for this is that one might argue that it is of less interest to establish the **NP**-hardness of a problem if the complexity of the problem is not likely to be confined to **NP**.

We will study the influence of two natural restrictions one might impose on hidden Markov models. One restriction is with respect to the structure of the hidden Markov model, restricting the hidden Markov model to be a left–right model. The other restriction is concerned with the probabilities of the hidden Markov model. A standard model assumption in algorithmic and complexity theory is a word size of $O(\log n)$ bits, where n is the size of the input. For simplicity, we will use the number of states, excluding the start- and end-state, in a hidden Markov model as a measure of its size n. A word size of $O(\log n)$ bits does not in itself bound the probabilities of a hidden Markov model as we might use several words for representing a probability. But adhering to the spirit of a word size bounded by $O(\log n)$ bits, we investigate the effect of polynomially bounded probabilities, i.e. restricting probabilities to be representable by $O(\log n)$ bits; more precisely we restrict each probability to be a rational number $\frac{a}{b}$ where a and b fit in words of size $O(\log n)$, i.e. are polynomially bounded by n. This means that there exists a constant c such that the smallest difference between any two representable probabilities is $\Omega(\frac{1}{n^c})$, hence the smallest positive probability is $\Omega(\frac{1}{n^c})$.

Proposition 2. *With no restriction on the allowed transition probabilities of a hidden Markov model, we cannot bound the length of the consensus string of the model by a polynomial in n. This holds even if we restrict the model to be a left–right model.*

Proof. Consider the model shown in Fig. 7, where we have a series of n non-silent states that emit an A with probability one and then either stays in the same state with

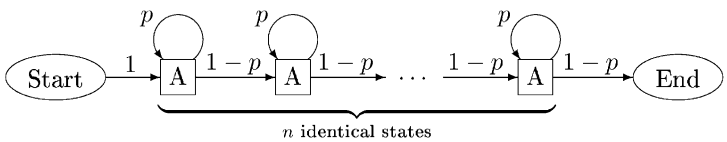

Fig. 7. A hidden Markov model with a simple left–right structure.

probability p or moves to the next state in the series with probability $1-p$. This model can only generate strings of the form A^{n+m} where $m \in \mathbf{N}_0$. The string A^{n+m} is emitted by exactly those runs that consist of $n+m$ states apart from the start-state and the end-state. All of these runs have probability $(1-p)^n p^m$, and there are $\binom{n+m-1}{m}$ different runs of this type, as we can identify each individual run by specifying the m self-loop transitions among the $n+m-1$ transitions not leading from the start-state or to the end-state. Hence

$$P_M(A^{m+n}) = \binom{n+m-1}{m}(1-p)^n p^m \tag{21}$$

and the ratio between the probability of A^{n+m+1} and the probability of A^{n+m} is

$$\frac{P_M(A^{n+m+1})}{P_M(A^{n+m})} = \frac{p(n+m)}{m+1}. \tag{22}$$

For $n > 1$ this ratio is strictly decreasing with increasing m and equals one for $m = \frac{pn-1}{1-p}$. Thus, the length of the consensus string is $n + \lceil \frac{pn-1}{1-p} \rceil = \lceil \frac{n-1}{1-p} \rceil$. We immediately observe that if p is not polynomially bounded then neither is $\lceil \frac{n-1}{1-p} \rceil$; if $1-p = O(1/n^{\omega(1)})$, then $\lceil \frac{n-1}{1-p} \rceil = \Omega(n^{\omega(1)})$. E.g. choosing $1-p = \frac{1}{2^n}$, i.e. $p = \frac{2^n-1}{2^n}$, the consensus string of M will be $A^{(n-1)2^n}$. □

As a final note, we observe that the probabilities $\frac{1}{2^n}$ and $\frac{2^n-1}{2^n}$ can be represented with $2n$ bits, wherefore the model in Fig. 7 can be represented in a number of bits that is polynomial in n, even if $p = \frac{2^n-1}{2^n}$. Hence, the above example shows that the consensus string of a hidden Markov model cannot be bounded by a polynomial in the size of a reasonable representation of the model.

Corollary 1. *With no restrictions on the allowed transition structure of a hidden Markov model, we cannot bound the length of the consensus string of the model by a polynomial in n. This holds even if we restrict probabilities to the set* $\{0, \frac{1}{2}, 1\}$.

Proof. As shown in Fig. 8 we can emulate the behavior of the model in Fig. 7 with $p = \frac{2^m-1}{2^m}$ by replacing each non-silent state with a submodel consisting of a non-silent state followed by a series of m silent states. From each of the silent states there is a probability of $\frac{1}{2}$ of proceeding to the next state and a probability of $\frac{1}{2}$ of returning to the non-silent state. The total probability of proceeding from one non-silent state to the next non-silent state without returning back to the original non-silent state is $(\frac{1}{2})^m$. Hence, the submodel behaves as a non-silent state with a probability of $\frac{1}{2^m}$ of moving to the next state and a probability of $\frac{2^m-1}{2^m}$ of staying in the same state. □

It should be noted that the modification of Fig. 8 introduces cycles to the model of Fig. 7, so after this modification the model is no longer a left–right model.

Proposition 3. *Let M be a left–right hidden Markov model with polynomially bounded probabilities. The length of the consensus string in M is polynomially bounded.*

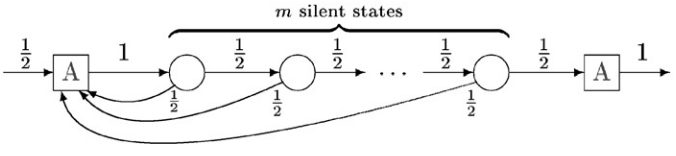

Fig. 8. Simulating the transition structure between the states of the model in Fig. 7 when $p = \frac{2^m-1}{2^m}$.

Proof. As M is a left–right model we can order the states of M such that there is no way to get from state i to state j if $i > j$, i.e. if state j is prior to state i in the ordering. So for any run with non-zero probability in M, the path will consist of an increasing sequence of states. Some states may occur several times in a row in this sequence due to self-loop transitions, but once we have left a particular state we will never revisit it.

We want to limit the total probability that any specific string of length m is generated for some suitable, polynomially bounded choice of m. Thus, we need to consider the number of different runs that can generate the same length m string. Unfortunately, this number might be infinite due to self-loop transitions of silent states. To circumvent this problem we will say that two paths are *equivalent* if they only differ in the number of times one or more silent states occur on the path. I.e. the paths of an equivalence class are all identical if we ignore silent states and just consider the sequence of non-silent states on the paths. Two runs are equivalent if they have equivalent paths and generate the same string. We can now identify each equivalence class of runs generating a string of length m with the number of occurrences of each non-silent state on the path of the run. As there is a total of m non-silent states on the path of a run generating a string of length m and at most n non-silent states in M, the number of equivalence classes is thus $\binom{n+m-1}{m}$.

On a path with m non-silent states in a left–right model with at most n non-silent states, at least $m-n$ of the transitions followed must be self-loop transitions from a non-silent state to itself. All the runs in an equivalence class share the same path when we ignore silent states. So for the runs in an equivalence class generating a string of length m, we can identify $m-n$ steps, consisting of a self-loop transition from a non-silent state to itself followed by the emission of a symbol from that state, that are part of all runs in the equivalence class. Removing these $m-n$ steps we obtain another equivalence class of runs. As the runs of this other equivalence class generate some string, the total probability of these runs cannot be larger than the probability p of the consensus string in M. Returning to the original equivalence class generating a string of length m, we can thus divide the runs into $m-n$ shared steps consisting of a self-loop transition followed by a symbol emission, and the remaining parts that summed over all the runs have a probability of at most p. By the polynomial bound on the probabilities and the requirement that the probability of any infinite run is zero, the probabilities of each of the self-loop transitions of the shared steps are at most $\frac{n^c-1}{n^c}$ for some constant c. So the total probability of all runs in the equivalence class is at most $p(\frac{n^c-1}{n^c})^{m-n}$. Combining this upper bound on the total probability of each equivalence class with the number of equivalence classes of runs generating a string of length m, we obtain that

$$P_M(s) \leqslant p\binom{n+m-1}{m}\left(\frac{n^c-1}{n^c}\right)^{m-n} \tag{23}$$

for any string s of length m. Thus, if we can prove that $\binom{n+m-1}{m}(\frac{n^c-1}{n^c})^{m-n} < 1$ for all m larger than some suitable bound polynomial in n, the consensus string in M cannot be longer than this bound.

The first element of this proof is to make the observation that for n sufficiently large, $(\frac{n^c-1}{n^c})^{n^c} \leqslant \frac{1}{2}$ as $\lim_{x\to\infty}(\frac{x-1}{x})^x = \frac{1}{e}$, cf. e.g. [16, Chapter X.2, Example 3]. For n sufficiently large, we thus obtain

$$\begin{aligned}\binom{n+m-1}{m}\left(\frac{n^c-1}{n^c}\right)^{m-n} &\leqslant \binom{n+m-1}{m}\left(\frac{1}{2}\right)^{(m-n)/n^c} \\ &\leqslant (n+m-1)^{n-1}2^{n^{1-c}-m/n^c}.\end{aligned} \tag{24}$$

Taking the logarithm and setting $m = n^{c+2}$ (actually $m = 2cn^{c+1}\log n$ suffices with a slightly more careful analysis) the last line of Eq. (24) becomes

$$(n-1)\log(n^{c+2}+n-1) - (n^2 - n^{1-c})\log 2. \tag{25}$$

The first term of this subtraction is $\Theta(n\log n)$ while the second term is $\Theta(n^2)$. For sufficiently large n this expression thus becomes negative, hence the expression of which we took the logarithm must be less than 1. Furthermore, differentiating the last line of Eq. (24) with respect to m we find

$$\begin{aligned}&\frac{\partial}{\partial m}(n+m-1)^{n-1}2^{n^{1-c}-m/n^c}\\ &\quad = (n+m-1)^{n-2}2^{n^{1-c}-m/n^c}\left((n-1) - \frac{(n+m-1)\ln 2}{n^c}\right).\end{aligned} \tag{26}$$

For positive n and m the first two factors are always greater than zero. Thus the derivative is zero if and only if the expression in the parentheses is zero, i.e. if

$$m = (n-1)\left(\frac{n^c}{\ln 2} - 1\right). \tag{27}$$

It can be checked that for values of m larger than this, the derivative is negative. Furthermore, for n sufficiently large, $n^{c+2} \geqslant (n-1)(\frac{n^c}{\ln 2} - 1)$. Thus $P_M(s) < p$ for all strings s with $|s| \geqslant n^{c+2}$ when n is sufficiently large which establishes the polynomial bound for the length of the consensus string. □

As a concluding remark it should be mentioned that by setting $p = \frac{n^c-1}{n^c}$ in the model of Fig. 7 the consensus string will have length $\Theta(n^{c+1})$. Hence, the bounds developed in this section cannot be improved significantly.

Proposition 4. *Let M be a left–right hidden Markov model with polynomially bounded probabilities, and let κ be a target probability. The problem of determining whether the probability of the consensus string of M is at least κ is* **NP***-complete.*

Proof. By Proposition 3 the problem is clearly in **NP**. We simply non-deterministically construct a string s that is of length bounded by some polynomial in the number of states of M and check whether $P_M(s) \geqslant \kappa$. As for the hardness we can almost, but not quite, just refer to Section 3. The models constructed in the hardness proof of this section are indeed left–right models, and most of the probabilities are either zero, one half, or one. But the probabilities of the transitions from the start-state to the first state of the layer for node u are $2^{\deg(u)} / \sum_{v \in G} 2^{\deg(v)}$ which might not be representable by a logarithmic number of bits.

The problem is that we want all runs to have identical probability, so we need to set the probability of choosing the transition to the layer of node u according to the number of different paths through that layer. But if there were the same number of paths through all layers, we could set the probabilities of the transitions from the start-state to the first state of each layer uniformly to $\frac{1}{|V|}$. A way to ensure the same number of paths through all layers is in the layer for node u to have a pair of a silent and a non-silent state for all nodes $v \neq u$, independent of whether there is an edge between u and v. But if there is an edge between u and v, the non-silent state of the pair will emit the symbol v with probability one; otherwise it will emit a dummy symbol $\$_u^v$ with probability one, a symbol that can only be emitted by this one particular state. This way there will be $2^{|V|-1}$ paths through each layer. Hence, all runs will have probability $1/|V|2^{|V|-1}$. The modified construction is illustrated in Fig. 9.

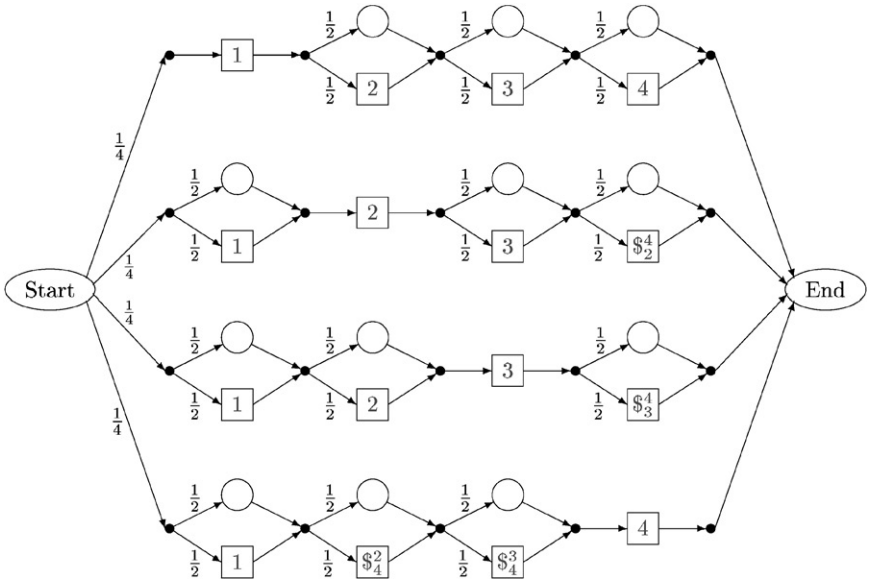

Fig. 9. The modified construction of a hidden Markov model from the graph in Fig. 2. Compare with the hidden Markov model in Fig. 3.

One can observe that any string containing one or more dummy symbols can be generated by at most one run. A specific dummy symbol can only be emitted by a state in a specific layer. And as no two non-silent states in the same layer can emit the same symbol, two different runs through the same layer cannot generate identical strings. The runs generating strings not containing dummy symbols are obtained by always choosing to bypass the non-silent states emitting dummy symbols. With these choices fixed, the remaining choices are identical to those in the original model of M_G constructed in Section 3—we choose the layer for a node u and for each v connected to u with an edge we choose to either enter the non-silent state emitting the symbol v, or to bypass it. So if there are k runs generating a string s in the original model M_G, there will also be k runs generating s in this modified model. Hence, the consensus string in the modified model will have probability $k/|V|2^{|V|-1}$ if and only if the maximal clique of G is of size k. Thus, it is **NP**-hard to determine whether the probability of the consensus string of this model is at least $k/|V|2^{|V|-1}$. As the model is a left–right model with polynomially bounded probabilities, the proposition follows. □

References

[1] N. Abe, M.K. Warmuth, On the computational complexity of approximating distributions by probabilistic automata, Mach. Learning 9 (1992) 205–260.
[2] K. Asai, S. Hayamizu, K. Handa, Prediction of protein secondary structure by the hidden Markov model, Comput. Appl. Biosci. (CABIOS) 9 (2) (1993) 141–146.
[3] A. Bateman, E. Birney, R. Durbin, S.R. Eddy, K.L. Howe, E.L.L. Sonnhammer, The Pfam protein families database, Nucleic Acid Res. 28 (1) (2000) 263–266.
[4] G.A. Churchill, Stochastic models for heterogeneous DNA sequences, Bull. Math. Biol. 51 (1989) 79–94.
[5] T.H. Cormen, C.E. Leiserson, R.L. Rivest, Introduction to Algorithms, MIT Press, Cambridge, MA, 1990.

[6] T.M. Cover, J.A. Thomas, Elements of Information Theory, Wiley, New York, 1991.
[7] L. Engebretsen, J. Holmerin, Clique is hard to approximate within $n^{(1-o(1))}$, in: Proceedings of the 27th International Colloquium on Automata, Languages and Programming (ICALP), 2000, pp. 2–12.
[8] M.R. Garey, D.S. Johnson, Computers and Intractability—A Guide to the Theory of NP-completeness, W.H. Freeman and Company, San Francisco, CA, 1979.
[9] J.A. Grice, R. Hughley, D. Speck, Reduced space alignment, Comput. Appl. Biosci. (CABIOS) 13 (1) (1997) 45–53.
[10] J. Håstad, Clique is hard to approximate within $n^{1-\varepsilon}$, Acta Math. 182 (1999) 105–142.
[11] F. Jelinek, Continuous speech recognition by statistical methods, Proc. IEEE 64 (4) (1976) 532–536.
[12] A. Krogh, Hidden Markov models for labeled sequences, in: Proceedings of the 12th International Conference on Pattern Recognition, 1994, pp. 140–144.
[13] A. Krogh, Two methods for improving performance of an HMM and their application for gene finding, in: Proceedings of the Fifth International Conference on Intelligent Systems for Molecular Biology (ISMB), 1997, pp. 179–186.
[14] A. Krogh, M. Brown, I.S. Mian, K. Sjölander, D. Haussler, Hidden Markov models in computational biology: applications to protein modeling, J. Mol. Biol. 235 (1994) 1501–1531.
[15] R.B. Lyngsø, C.N.S. Pedersen, H. Nielsen, Metrics and similarity measures for hidden Markov models, in: Proceedings of the Seventh International Conference on Intelligent Systems for Molecular Biology (ISMB), 1999, pp. 178–186.
[16] T.G. Madsen, Matematisk Analyse, Department of Mathematical Sciences, University of Aarhus, 1991.
[17] L.R. Rabiner, A tutorial on hidden Markov models and selected applications in speech recognition, Proc. IEEE 77 (2) (1989) 257–286.
[18] Y. Singer, M.K. Warmuth, Training algorithms for hidden Markov models using entropy based distance functions, in: Advances in Neural Information Processing Systems 9 (NIPS), 1996, pp. 641–647.
[19] E.L.L. Sonnhammer, G. von Heijne, A. Krogh, A hidden Markov model for predicting transmembrane helices in protein sequences, in: Proceedings of the Sixth International Conference on Intelligent Systems for Molecular Biology (ISMB), 1998, pp. 175–182.