Available online at www.sciencedirect.com
ScienceDirect
Journal of Computer and System Sciences 73 (2007) 1060–1077
JOURNAL OF COMPUTER AND SYSTEM SCIENCES
www.elsevier.com/locate/jcss

# The most probable annotation problem in HMMs and its application to bioinformatics

Broňa Brejová [b], Daniel G. Brown [a], Tomáš Vinař [b,*]

[a] *David R. Cheriton School of Computer Science, University of Waterloo, 200 University Avenue West, Waterloo, ON, N2L 3G1, Canada*
[b] *Department of Biological Statistics and Computational Biology, Cornell University, Ithaca, NY 14853, USA*

Received 14 July 2006; received in revised form 25 September 2006

Available online 14 March 2007

**Abstract**

Hidden Markov models (HMMs) are often used for biological sequence annotation. Each sequence feature is represented by a collection of states with the same label. In annotating a new sequence, we seek the sequence of labels that has highest probability. Computing this most probable annotation was shown NP-hard by Lyngsø and Pedersen [R.B. Lyngsø, C.N.S. Pedersen, The consensus string problem and the complexity of comparing hidden Markov models, J. Comput. System Sci. 65 (3) (2002) 545–569]. We improve their result by showing that the problem is NP-hard for a specific HMM, and present efficient algorithms to compute the most probable annotation for a large class of HMMs, including abstractions of models previously used for transmembrane protein topology prediction and coding region detection. We also present a small experiment showing that the maximum probability annotation is more accurate than the labeling that results from simpler heuristics.


*Keywords:* Hidden Markov models; NP-hardness; Sequence annotation; Computational biology; Gene finding

## 1. Introduction

We present several contributions to improve the understanding of the most probable annotation problem in hidden Markov models (HMMs) and its impact on bioinformatics. We document the importance of this problem, show it is NP-hard for a specific HMM, and then present algorithms that solve the problem in polynomial time for several important classes of HMMs.

Hidden Markov models are often used in bioinformatics for sequence annotation tasks. In this domain, the goal is to identify various functional features in biological sequences. We represent our knowledge of the probabilistic properties of a domain in an HMM, which embodies the joint probability distribution of possible sequences and annotations. Then, for a new unannotated sequence we seek an annotation of that sequence with high conditional probability.

For many HMMs, computing the maximum probability annotation of a sequence is equivalent to finding the maximum probability state path for that sequence, easily found by the classical Viterbi algorithm [9]. However, this happy

* Corresponding author.
*E-mail addresses:* bb248@cornell.edu (B. Brejová), browndg@cs.uwaterloo.ca (D.G. Brown), tv35@cornell.edu (T. Vinař).

0022-0000/$ – see front matter 
doi:10.1016/j.jcss.2007.03.011

state of affairs is not the case for many HMMs used in bioinformatics applications, where multiple state paths correspond to the same annotation. To deal with this problem, application designers typically use a variety of heuristics to decode new sequences, which do not always yield the optimal, maximum probability annotation.

One might wish to be able to decode all HMMs optimally. Unfortunately, it is unlikely that a polynomial-time algorithm computing the most probable annotation exists: Lyngsø and Pedersen [15] showed that the problem is NP-hard (see also earlier work on a related model by Casacuberta and de la Higuera [4]). However, one possible opening does exist: the proof of Lyngsø and Pedersen [15] assumed that both the HMM and its sequence were part of the input. In our typical applications, though, the HMM is fixed and the input is just the sequence. Perhaps, for each fixed HMM, an algorithm whose runtime is polynomial-time in the length of the sequence, but exponential in the size of the HMM can be found.

We show that this is not the case, unless P = NP. We present an HMM of modest size, for which finding the most probable annotation of an input sequence is NP-hard. Still, the most probable annotation problem is not hard to solve for *all* HMMs. We present a range of algorithms with increasing running time that can compute the most probable annotation for increasingly larger classes of HMMs; the fastest algorithm in our set is the classical Viterbi algorithm. However, the problem of deciding whether a particular HMM is NP-hard to decode, or whether there exists a polynomial-time decoding algorithm for it, is still open.

## 2. Background and related work

An HMM is a generative probabilistic model composed of states and transitions [17]. We denote the set of states of a hidden Markov model as $V$. For each state $v \in V$, $e_v(x)$ is the *emission probability* of symbol $x$ in state $v$, and $a(v, w)$ is the *transition probability* from state $v$ to state $w$. The generative process, starting in a designated start state $\pi_1$, first emits a random symbol according to the emission probabilities of the current state, and then transitions to a new state determined randomly according to the transition probabilities. The process is repeated $n$ times. The probability of generating sequence $x_1 \dots x_n$ by state path $\pi_1 \dots \pi_n$ is $\Pr(x_1, \dots, x_n, \pi_1, \dots, \pi_n) = e_{\pi_1}(x_1) \cdot \sum_{i=2}^{n} a(\pi_{i-1}, \pi_i) \cdot e_{\pi_i}(x_i)$. If the sequence is clear from the context, we call this probability the *probability of state path* $\pi_1 \dots \pi_n$. (The conditional probability of a state path given a sequence, $\Pr(\pi_1, \dots, \pi_n | x_1, \dots, x_n)$, is proportional to the joint probability, so maximizing one maximizes the other.) The Viterbi algorithm [9] computes a state path of highest probability, in time linear in $n$.

For notational convenience, we also introduce *silent states*, which emit no symbols (see, e.g., [6, Section 3.4]). An HMM is well-defined only if there is no directed cycle composed of silent states.

When we use HMMs for sequence annotation, we label each state with the feature to which it corresponds. We denote the label of state $v$ by $\lambda(v)$. For a given sequence $x_1 \dots x_n$, a *sequence annotation* is a sequence of $n$ labels. Each state path $\pi_1 \dots \pi_n$ corresponds naturally to its annotation $\lambda(\pi_1) \dots \lambda(\pi_n)$. However, in some HMMs this mapping is not one-to-one; many state paths may correspond to the same annotation. Such HMMs have the *multiple path problem*. These state paths represent alternative explanations with the same semantic meaning. If we are trying to find the most probable meaning, or annotation, for the sequence, we should add all their probabilities. Formally, the *probability of a given annotation* $L = \lambda_1 \dots \lambda_n$ is the sum of the probabilities of all state paths $\pi_1 \dots \pi_n$ with annotation $L$. The *most probable annotation problem* is to find the annotation $L$ with maximum probability.

In our diagrams, we display hidden Markov models as graphs with states as vertices and non-zero probability transitions as edges. Vertex color corresponds to the label of the state. Silent states are indicated by smaller size.

Choosing labels is an important step in creating models where there are different classes of features. If we assign the same sets of labels to all classes of a feature, the most probable annotation decoding will find the most likely overall feature structure. On the other hand, we can also assign separate sets of labels to each class, in which case the most probable annotation will also distinguish between these classes.

The most widely used annotation method is the *Viterbi algorithm* [9] that finds the state path of maximum probability for the sequence, and reports the annotation of that path. For HMMs without the multiple path problem, the Viterbi annotation is the maximum probability annotation.

For HMMs with the multiple path problem, the most probable state path does not necessarily correspond to the most probable (or even a high probability) annotation. Consider, for example, the simple HMM in Fig. 1 with one state labeled $\alpha$ and two states labeled $\beta$. The most probable state path for the string $(ab)^n a$ is always $\alpha^{2n+1}$; it has probability $0.09^n \cdot 0.5$. However, the most probable annotation for the sequence $(ab)^n a$ is always $\alpha(\beta\alpha)^n$, which has

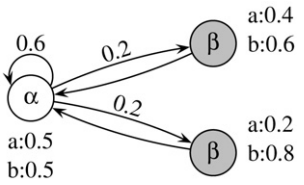



Fig. 1. The most probable path can differ from the most probable annotation.

higher probability, $0.14^n \cdot 0.5$, even though the highest probability single path with this annotation has probability exponentially smaller, just $0.08^n \cdot 0.5$.

Differences between the most probable path and the most probable annotation are a cause for concern. In our example, as $n$ grows, the number of paths forming the most probable annotation grows exponentially. The probability of each path with the most probable annotation is very low compared to the probability of the most probable path, though the most probable annotation is exponentially higher in probability than the most probable path.

Previous authors [3,13] noted the discrepancy between finding the most probable state paths, easily done in polynomial time, and finding the most probable annotations in HMMs with the multiple path problem, which is much harder. Their proposed heuristics are quite diverse.

A simple (but rarely implemented) suggestion is to compute the $k$ most probable paths in the HMM, which provide $k$ candidate annotations [6]. These can be found efficiently using an algorithm of Eppstein [8]. For each path, we identify its corresponding annotation and compute the probability of that annotation, and then choose the most probable of these $k$ annotations. This approach will fail when the probability of each path in the most probable annotation is small, as in Fig. 1.

A different heuristic, the $N$-best algorithm, was introduced by Schwartz and Chow [19] and used in the context of biological sequence analysis by Krogh [13]. It maintains a pool of several candidate annotations and guarantees only that the probability of the chosen annotation is at least the probability of the most probable state path, not that it finds the most probable annotation.

Or, one can apply *a posteriori* decoding and compute the most probable label for each sequence position. However, no state path may correspond to this annotation, so it may not be consistent with the biological constraints of the model. To complete the heuristic, a second step is required to modify such a labeling to obtain a plausible annotation (see, e.g., [16]).

## 3. The multiple path problem in bioinformatics applications

In this section, we discuss several scenarios from common bioinformatics applications of hidden Markov models where the multiple path problem arises. Then, in a small experiment, inspired by vertebrate gene finding, we show that computing the maximum probability annotation gives better decoding accuracy than the annotation corresponding to the maximum probability path.

### *3.1. Bioinformatics applications with the multiple path problem*

HMMs are used in the most successful software for predicting the topology of transmembrane proteins, or segmenting a protein sequence into transmembrane helices (parts traversing the membrane), cytoplasmic loops (parts inside the cell), and non-cytoplasmic loops (parts extending outside the cell). Figure 2(a) shows a simplification of an HMM for this task. The topology enforces the simple physical constraint that cytoplasmic loops must be separated from non-cytoplasmic loops by transmembrane helices. Krogh et al. [14] used a similar topology in TMHMM. They create separate modules for short and long non-cytoplasmic loops, though, since these exhibit different statistical properties (as in Fig. 2(b)). This change creates the multiple path problem, as there is an overlap in the loop lengths generated by both modules.

The second problem we examine is modeling feature length distributions in HMMs. A common method of modeling longer sequence features in HMMs is by a state with a self-loop transition as shown in Fig. 3(a). The number of consecutive symbols from the white state (the length of the white feature) will be geometrically distributed. This distribution may not necessarily match the true distribution. One approach to resolving this problem, suggested by

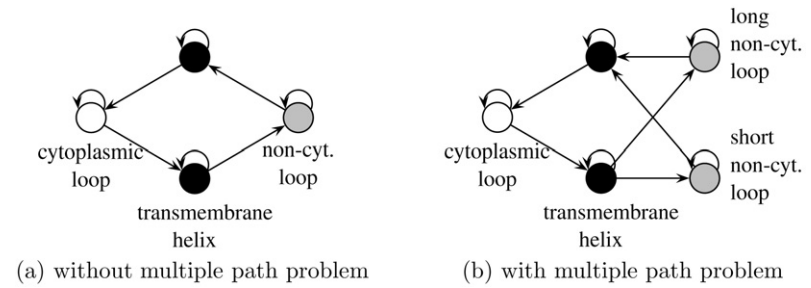



Fig. 2. Simplified topology of an HMM for transmembrane topology prediction.

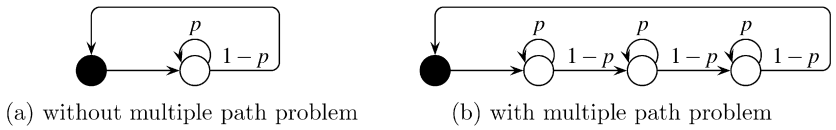



Fig. 3. One-state and three-state models of feature length distribution.

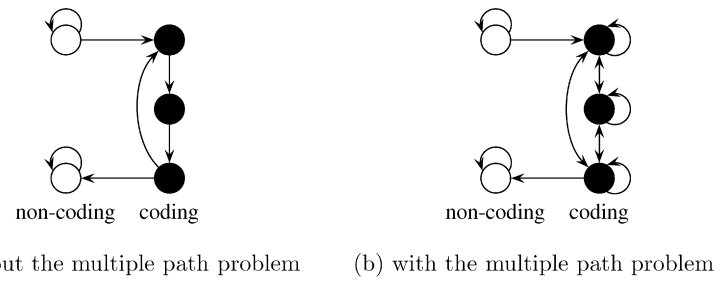



Fig. 4. Simplified model of the HMM used in ESTScan [10].

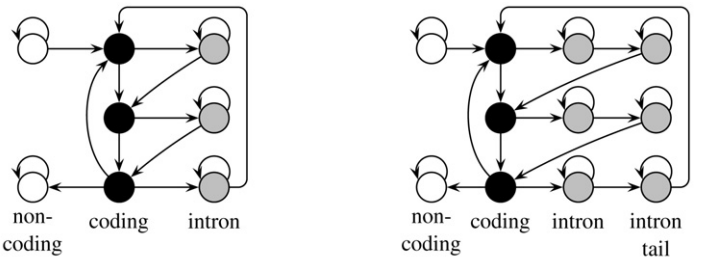
Durbin et al. [6] and recently re-examined by Johnson [11], is to replace the single state with a gadget of states, as in the example in Fig. 3(b). In such a modified HMM, the length distribution of a feature is no longer geometric, but can be increasingly complex. However, the probability of generating segments of a given length within such a gadget is the sum of the probabilities of many low-probability paths; this trick lets the model achieve the non-geometric length distribution. Such an HMM has the multiple path problem, and applying the Viterbi algorithm to decode it yields surprising and seemingly paradoxical results [21].

Another problem commonly addressed by HMMs is protein coding region prediction. In mRNA molecules (such as those used to produce expressed sequence tags, or ESTs), proteins are encoded by a sequence of triples of RNA nucleotides. To distinguish coding regions from surrounding non-coding regions, we might propose the HMM in Fig. 4(a). It has three different states labeled black (for coding), but does not have the multiple path problem since each annotation of an mRNA sequence has one path through the HMM.

The model in Fig. 4(a) will not work well for real EST sequences, which routinely include insertions and deletions in them. Iseli et al. [10], in their program ESTScan, add transitions to the simple model for insertions and deletions; as in Fig. 4(b). A path in this model may include some combination of insertions and deletions, and many paths yield the same annotation, so it has the multiple path problem.

The situation gets more complicated if we want to predict protein coding regions in genomic sequences. In such sequences, the coding part of a gene can be interrupted by several non-coding regions called introns. During transcription, introns are removed from mRNA, and the remaining coding regions are concatenated and translated to protein. A simple HMM for such genomic sequence is shown in Fig. 5(a). There are three identical intron models included to maintain the three-periodicity of the DNA code across intron boundaries.

This model represents introns by a single state, thus introducing an assumption that the composition of introns is homogeneous. However, vertebrate intronic sequences contain a variable-length tail rich in the nucleotides C and T




(a) without multiple path problem (b) with multiple path problem

Fig. 5. Simple model of exon/intron structure.

[3], with composition is very different from the rest of the intron. Its presence provides strong support that an intron boundary may be nearby. To incorporate this information into the HMM topology, we can include a second intron state representing the tail, as shown in Fig. 5(b). This creates the multiple-path problem: there are several high-probability sites for the transfer from the intron state to the tail state.

### *3.2. The multiple path problem can cause bad annotations*

All of our examples of the previous subsection developed the multiple path problem when we increased the faithfulness of the model to what is found in reality. However, if we use the Viterbi algorithm for decoding such models, we find a single most probable path through the model, not the most probable annotation. Will this result in bad annotations? Here, we show a simple synthetic example, motivated by the intron tail scenario in Fig. 5. Using Viterbi decoding can lead to decreased accuracy of predictions, relative to what would result from computing the maximum probability annotation.

The HMM $A$ in Fig. 6 emits symbols over the alphabet $\{0, 1\}$, where the numbers inside states represent the emission probability of the symbol 1. This HMM outputs alternating white regions of mean length 20 and gray regions of mean length 34. The distribution of symbols (the composition) in the white regions is constant, while in the gray regions, it changes towards their right ends. The gray regions are bounded by a two-symbol signal, always 11, on both sides. We used this HMM to generate 5000 binary sequences of mean length about 500 for various combinations of the parameters $p_1$ and $p_2$.

Then we annotated these sequences with three decoding algorithms. We identified the annotation corresponding to the maximum probability path using the standard Viterbi algorithm, we computed the maximum probability annotation using our extended Viterbi algorithm, described later in Section 5, and we identified the annotation corresponding to the maximum probability path through the simplified HMM $B$, shown in Fig. 7. In this simplified model, we replaced the two gray-labeled states, with probabilities $p_1$ and $p_2$ of emitting a 1, by a single state and set the emission and transmission parameters of that state to maximize the likelihood of it generating sequences from model $A$. The gray state of model $B$ has probability approximately 0.97 of a self-loop, and emits the symbol 1 with probability $p' = (2p_1 + p_2)/3$. The simplified HMM $B$ does not have the multiple path problem, so the Viterbi algorithm yields the most probable annotation of a sequence.

For each sequence, we compared the annotation found by our three annotation algorithms to the true annotation, the labels of the states that generated the sequences. The error rate is the percentage of mislabeled positions. Figure 8 shows our experimental results.

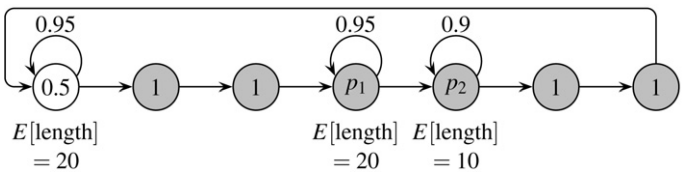



Fig. 6. HMM $A$: A simple HMM with the multiple path problem.

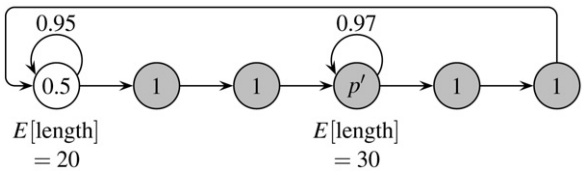



Fig. 7. HMM $B$: Simplified model of HMM $A$, with no multiple path problem, but with uniform composition throughout gray regions.

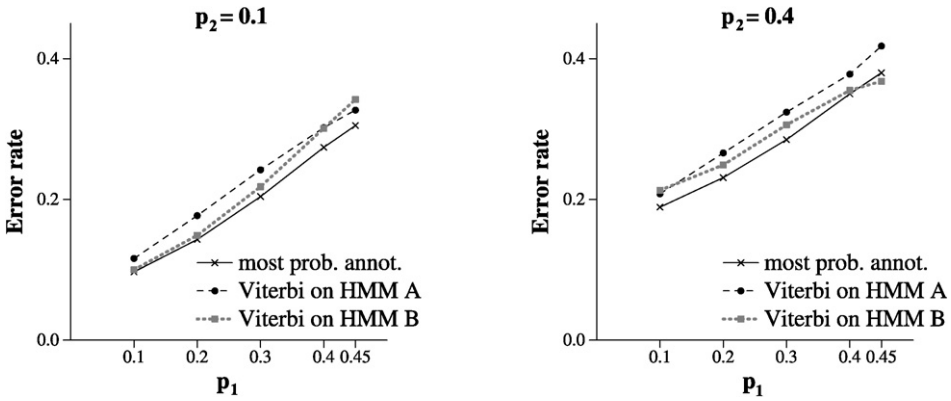



Fig. 8. Error rate of different decoding methods.

We have observed two trends in the data. First, computing the most probable annotation in model $A$ increases the accuracy, compared to applying the Viterbi algorithm to model $A$. Second, the Viterbi algorithm, applied to the simplified model $B$, often performs better than the Viterbi algorithm applied to the true model $A$. This behavior is counter-intuitive: since the data were generated by model $A$, one would expect that decoding sequences using it would give best results. However, Viterbi decoding, using a less correct model, without the multiple path problem, actually gives better results.

We verified the statistical significance of our experimental results with simple sign test. For a given pair of decoding methods, we count for how many of the testing sequences one method has lower error rate than the other. We compare these counts with the uniform distribution by a chi-squared test. The differences in prediction accuracy shown in Fig. 8 are significant at the 0.001 significance level, except for the difference between the two versions of the Viterbi algorithm for $p_2 = 0.1$ and $p_1 = 0.4$, and the difference between most probable annotation and the Viterbi algorithm on HMM $B$ for $p_2 = 0.4$ and $p_1 = 0.4$.

## 4. Finding the most probable annotation is NP-hard

Our experiment shows that most probable annotation decoding achieves better results than picking the annotation corresponding to the Viterbi path, for HMMs with the multiple path problem. In this section, we show that finding the most probable annotation is unfortunately NP-hard in general. Lyngsø and Pedersen [15] proved the NP-hardness of this problem by reduction from the maximum clique problem. From a graph with $n$ vertices, they construct an HMM with $\Theta(n^2)$ states and a sequence of length $\Theta(n)$, forming an instance of the most probable annotation problem. This does not correspond to our typical scenario of sequence annotation: we fix an HMM, and annotate sequences according to that HMM. Our new, stronger NP-hardness proof shows that finding the most probable annotation is NP-hard, even for a specific small fixed HMM.

### *4.1. Layered graphs and the* BEST-LAYER-COLORING *problem*

For a given sequence, we can represent the HMM by a layered weighted directed acyclic graph, whose vertices conjoin sequence positions with HMM states. With appropriate edge weights, paths in this graph correspond to HMM state paths. In this section, we consider an unweighted version of such graphs, which we call layered digraphs. We

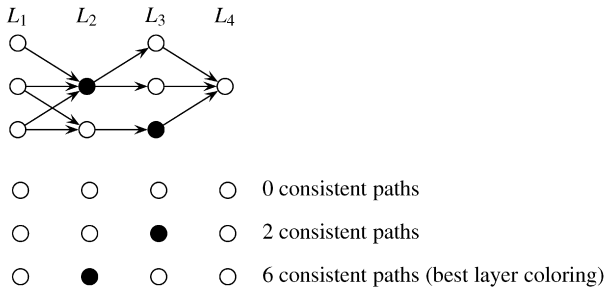



Fig. 9. A layered graph and three layer colorings.

show that a path counting problem, directly analogous to the most probable annotation problem in HMMs, is NP-complete. This proof will form the basis for the NP-hardness proof of the most probable annotation problem.

**Definition 1** *(Layered digraphs and colorings).* A *colored proper layered digraph* is a directed graph, with its vertices arranged in layers $L_1, L_2, \ldots, L_w$. Each edge connects a vertex in a layer $L_i$ to a vertex in layer $L_{i+1}$. Each vertex is colored white or black.

A *layer coloring* is an assignment of a color (white or black) to each layer of such a directed graph. A directed path from layer $L_1$ to layer $L_w$ is *consistent with a layer coloring* if the colors of the vertices on the path match the colors of the layer coloring.

Figure 9 shows an example layered digraph with four layers. No paths are consistent with layer coloring $\circ\,\circ\,\circ\,\circ$, but six paths are consistent with $\circ\,\bullet\,\circ\,\circ$. The BEST-LAYER-COLORING problem asks a natural question.

**Definition 2** *(Best-Layer-Coloring problem).* Given a colored proper layered digraph $G$ and a threshold $T$, is there a layer coloring with at least $T$ consistent paths?

**Theorem 3.** BEST-LAYER-COLORING *is NP-complete, even for graphs where each layer has at most* 30 *vertices.*

**Proof.** BEST-LAYER-COLORING is in NP; for a given layer coloring, the number of consistent paths is at most exponential in the number of layers and can be computed by simple dynamic programming.

To prove NP-hardness, we reduce SAT to BEST-LAYER-COLORING. Consider an instance of SAT: a formula in conjunctive normal form with $n$ variables $u_1, u_2, \ldots, u_n$ and $m$ clauses $c_1, c_2, \ldots, c_m$.

We give an overview of the construction in Fig. 10. The boxes in the figure represent components of the colored layered digraph for the SAT instance. Lines connecting components represent subgraphs of the final graph that propagate the same number of paths from left to right regardless of the layer coloring chosen in those layers. If $p$ paths through the layered digraph are compatible with the first $i$ layers of a layer coloring, and such a line is entered after $i$ layers, then no matter what colors are chosen in the layers corresponding to the line, $p$ paths are compatible with it at the end of the line. We use one black and one white vertex in each layer of the subgraph, and join both nodes from each layer of the subgraph to both nodes in the next layer.

The graph consists of $m+1$ blocks $0, 1, 2, \ldots, m$, each with $2n$ layers. The blocks from 1 to $m$ correspond to the $m$ clauses in the instance, while the zeroth block maintains consistency. The layer coloring of each block represents a truth assignment for the variables $u_1, \ldots, u_n$. The truth assignment of each variable is encoded by the color of two consecutive layers: $\circ\,\circ$ for "false" or $\circ\,\bullet$ for "true." Layer colorings not of this form will have no corresponding paths, so we will not need to consider them; this allows us to speak of the layer coloring and truth assignment of a block interchangeably. We will represent truth assignments by numbers in binary representation (with $u_1$ as the highest-order bit and $u_n$ as the lowest-order bit). Let $x$ be the number representing the truth assignment of block 0 and let $y_1, \ldots, y_m$ be those for blocks $1, \ldots, m$.

In a "yes" instance of SAT, all $m+1$ truth assignments $x, y_1, \ldots, y_m$ must be the *same* satisfying truth assignment of the SAT formula. If the problem is satisfiable, and we consistently use the same satisfying truth assignment in each block, our construction produces $2m(4^n - 2^{n+1} + 1) + 1$ paths consistent with the corresponding layer coloring. If

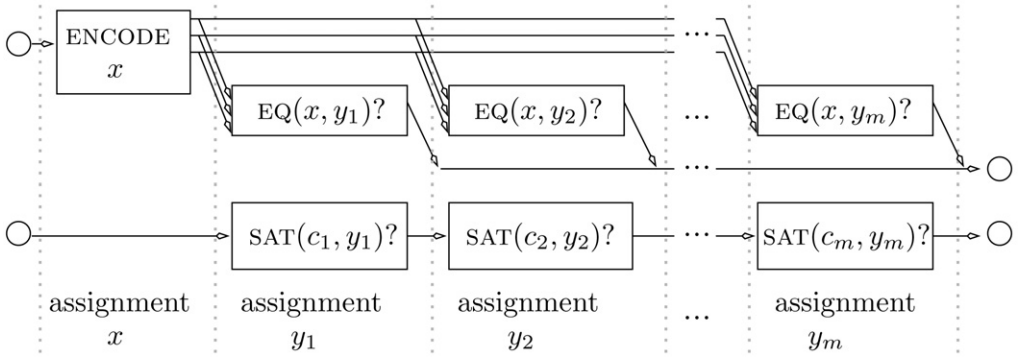



Fig. 10. Overview of NP-completeness proof of BEST-LAYER-COLORING.

the assignments chosen in each block are not all the same, or if we ever choose an assignment that does not satisfy its clause, we will not have that many paths corresponding to the layer coloring chosen. This reduces SAT to BEST-LAYER-COLORING.

Figure 10 decomposes the structure of the graph into several components, each of them having several inputs and outputs; we will further decompose these in what follows. An input of a component is the number of consistent paths ending in a designated vertex on the left-most layer of the component. Similarly, an output of a component is the number of consistent paths ending at a designated vertex on the right-most layer of the component. Let $A \xrightarrow{x} B$ denote a component that transforms a vector of inputs $A$ to a vector of outputs $B$, when the layers of the component have coloring $x$.

The component ENCODE$(x)$ in Fig. 10 encodes the truth assignment $x$, of block 0 as a vector of three integers, $v(x)$, on its output. In blocks $i = 1, 2, \ldots, m$, we test if the truth assignment of that block, $y_i$, is the same as $x$ in the component EQ$(x, y_i)$. The input of this component is the vector $v(x)$ that comes from the ENCODE$(x)$ module in block 1. If $x$ and $y_i$ are the same truth assignment, EQ$(x, y_i)$ outputs the number $2K(n)$, where $K(n) = 4^n - 2^{n+1} + 1$; if $x \neq y_i$, then the module EQ$(x, y_i)$ outputs a number smaller than $2K(n)$. Finally, the component SAT$(c_i, y_i)$ outputs its input if the truth assignment $y_i$ satisfies clause $c_i$, or 0 otherwise. The input to SAT$(c_1, y_1)$ is a single path; because the SAT modules connect in series, there is one path that exits the SAT modules exactly when the truth assignment chosen for all blocks $i$ satisfies clause $c_i$. An additional layer before the first block and after the last block ensure the proper start and end of each consistent path.

We now describe the components. Lemma 4 shows how to create the component SAT$(c_i, y_i)$; Lemma 5 shows the construction of ENCODE$(x)$ and EQ$(x, y_i)$. The total number of vertices in each layer of this construction is at most 30, so an instance of SAT can be reduced to BEST-LAYER-COLORING with that constant number of nodes per layer. □

**Lemma 4.** *For any clause $c$, there exists a component* SAT$(c, y)$ *with a constant number of vertices in each layer that outputs its input if truth assignment $y$ satisfies clause $c$, and* 0 *otherwise.*

**Proof.** The component has two parallel lanes, one corresponding to the clause being satisfied, the other to it being unsatisfied. There is one 2-layer section for each variable of the truth assignment. The structure of the $i$th section depends on whether variable $u_i$ is present in the clause $c_i$ as the positive literal $u_i$, the negative literal $\neg u_i$ or not at all. If the variable is present and its assignment satisfies the clause, the path switches from the "unsatisfied" lane to the "satisfied" lane. The single input to the component comes to the first vertex of the "unsatisfied" lane and the single output of the component is the last vertex of the "satisfied" lane. Figure 11 shows an example of a chain of two SAT components. Gray vertices represent vertices that can be used in a path for either layer color. These vertices are used only to simplify the drawings: each can be replaced by a white and a black vertex with the same incoming and outgoing edges. □

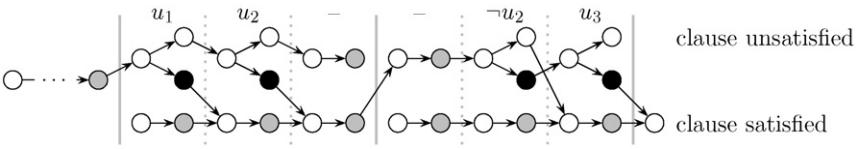



Fig. 11. Assembly of SAT components for the formula $(u_1 \vee u_2) \wedge (\neg u_2 \vee u_3)$.

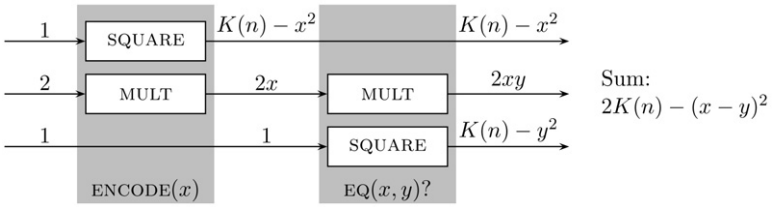



Fig. 12. Overview of ENCODE and EQ.

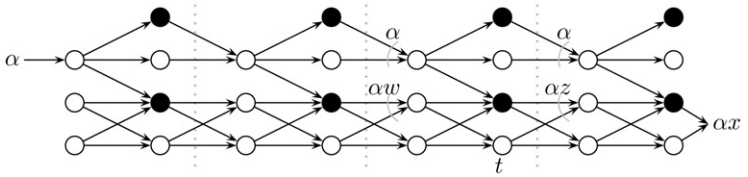



Fig. 13. Component $\text{MULT}(x) : \alpha \xrightarrow{x} \alpha x$.

**Lemma 5.** *There exist components* ENCODE$(x)$ *and* EQ$(x, y)$*, with a constant number of vertices in each layer, such that if we use the output of* ENCODE$(x)$ *as the input to* EQ$(x, y)$*, the output of* EQ$(x, y)$ *is* $2K(n)$*, if* $x = y$*, and smaller otherwise.*

**Proof.** Let $x$ be the number whose binary representation encodes the truth assignment of variables $u_1, \ldots, u_n$. We encode $x$ as a vector with the ENCODE$(x)$ component; if this encoding is used as the input of EQ$(x, y)$, the output of EQ$(x, y)$ will be $2K(n) - (x - y)^2$, which is equal to $2K(n)$ if $x = y$ and is less than $2K(n)$ otherwise. To this end, the ENCODE component computes the three element vector $\text{ENCODE}(x) : 1 \xrightarrow{x} (K(n) - x^2, 2x, 1)$ and the EQ component computes the scalar value $\text{EQ}(x, y) : (\alpha, \beta, 1) \xrightarrow{y} K(n) - y^2 + \beta \cdot y + \alpha$. An overview of the structure of these two components is shown in Fig. 12.

These two required components can be constructed as a combination of two subcomponents, $\text{MULT}(x) : \alpha \xrightarrow{x} \alpha x$ and $\text{SQUARE}(x) : 1 \xrightarrow{x} K(n) - x^2$, as shown in Fig. 12. Both MULT$(x)$ and SQUARE$(x)$ consist of identical 2-layer sections, each processing one bit of $x$.

Consider the section of component MULT$(x)$ processing the $k$th bit of the truth assignment $x$. Let $w$ be the binary representation of truth assignment of the first $k - 1$ variables, $t$ be the truth assignment of the $k$th variable, and $z = 2w + t$ be the truth assignment of the first $k$ variables.

The section has two inputs and outputs: $(\alpha, \alpha w) \xrightarrow{t} (\alpha, \alpha z)$. The value $\alpha z$ can be computed from the values $\alpha w$, $\alpha$, and $t$ by the following equation:

$$\alpha z = \begin{cases} 2\alpha w, & \text{if } t = 0, \\ 2\alpha w + \alpha, & \text{if } t = 1. \end{cases} \tag{1}$$

Figure 13 shows the component MULT$(x)$ for an assignment with four variables.

Similarly, we can design the component SQUARE$(x)$. Section $k$ of this component has four inputs and outputs $(1, B(k-1), C(w, k-1), D(w, k-1)) \xrightarrow{t} (1, B(k), C(z, k), D(z, k))$, where functions $B, C, D$ are defined as follows:

$$B(k) = 2^{k+2} - 4, \tag{2}$$

$$C(z, k) = B(k) - 4z, \tag{3}$$

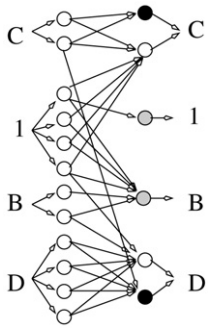



Fig. 14. One section of component SQUARE$(x): 1 \xrightarrow{x} K(n) - x^2$.

$$D(z,k) = 4^k - 2^{k+1} + 1 - z^2. \tag{4}$$

The values of $B, C, D$ can be computed bit by bit using the following recurrence relations:

$$B(k) = 2B(k-1) + 4, \tag{5}$$

$$C(z,k) = \begin{cases} 2C(w,k-1) + 4, & \text{if } t = 0, \\ 2C(w,k-1), & \text{if } t = 1, \end{cases} \tag{6}$$

$$D(z,k) = \begin{cases} 4D(w,k-1) + B(k-1) + 1, & \text{if } t = 0, \\ 4D(w,k-1) + C(w,k-1), & \text{if } t = 1. \end{cases} \tag{7}$$

The graph for one section, based on the recurrences (5)–(7), is depicted in Fig. 14. Graphs of $n$ such identical sections can be assembled into the component SQUARE$(x)$. Since $B(0)$, $C(0,0)$, and $D(0,0)$ are all zeroes, there are no paths entering these inputs in the first section. The output $D$ of the last section is the output of the whole component, and it has value $D(x,n) = K(n) - x^2$, as desired. □

Now, with all the required components, our proof is complete. We have given a layered directed graph for the SAT instance. The layer with the largest number of vertices has four lines carrying previously computed values (with two vertices in each of them), one component EQ (consisting of MULT with at most four vertices per layer, SQUARE with at most 12 vertices per layer, and a line carrying a previously computed value with two vertices), and one component SAT (with at most four vertices per layer). Therefore each layer has at most 30 vertices.

### *4.2. Constructing a small HMM that is NP-hard to decode*

In this section, we use ideas from the proof of Theorem 3 to reduce SAT to most probable annotation, obtaining a small HMM that is NP-hard to decode. The SAT instance will be encoded in the sequence to annotate.

If the SAT formula contains $m$ clauses with $n$ variables, then the sequence will consist of $(m+1)$ blocks of $(n+1)$ symbols, terminated by a special symbol ! as follows.

The first block is a string $0^n\#$. Each of the next $m$ blocks encodes one clause of the formula. The $i$th symbol is 1 if the clause contains the positive literal of the $i$th variable, 0 for the negative literal, and– –if the clause does not contain the $i$th variable. Each block is terminated by a special symbol \$. For example, the formula $x_1 \wedge (x_2 \vee \neg x_3)$ is encoded as 000#1––\$–10\$!.

The HMM will have two labels: white and gray. For satisfiable formulas, the most probable annotation will represent a satisfying assignment, in which the special characters are labeled white, variables with value "true" are labeled gray, and variables with value "false" are white. In the most probable annotation representing a satisfying assignment, the same pattern will repeat $m+1$ times, and in each clause at least one symbol "0" will be labeled white, or one symbol "1" will be labeled gray. Figure 15 shows an example of a satisfying assignment as an annotation.

Figure 16 shows the schema of the HMM. Each state emits only the symbols depicted inside the state with equal probability. Multiple edges join some pairs of states; the multiplicity of an edge is noted at the tail of the edge (if not explicitly stated, it is 1).

| | |
|---|---|
| Formula: | $x_1 \wedge (x_2 \vee \neg x_3)$ |
| Encoding of the formula: | 0 0 0 # 1 - - $ - 1 0 $ ! |
| Satisfying assignment: | $x_1$: true, $x_2$: true, $x_3$: false |
| The assignment as an annotation: | 0 0 0 # 1 - - $ - 1 0 $ ! |

Fig. 15. Encoding formulas and assignments for HMM solving SAT.

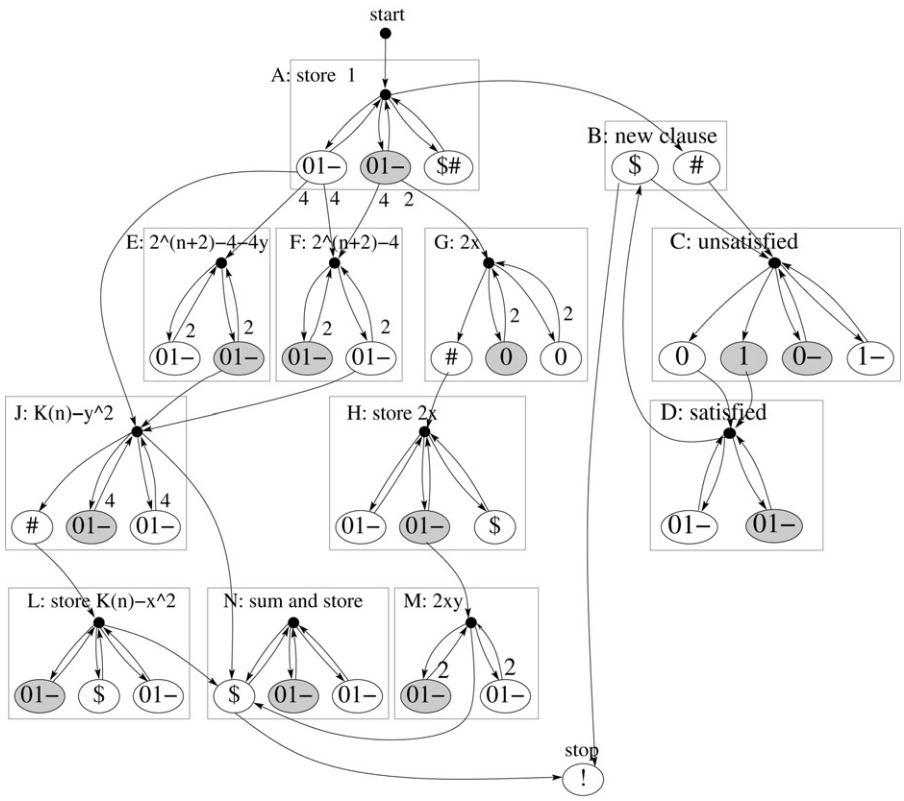



Fig. 16. HMM solving SAT.

We want all non-zero probability paths in this HMM to have the same probability, so we transform the diagram in the figure to an HMM by removing silent states and introducing an error symbol, and an error state (not shown in the figure). The error state emits only the error symbol and has only one outgoing transition, a self-loop. The error symbol can be emitted in each of the states; emissions of this symbol, and transitions to the error state, normalize the probabilities, so that every ordinary symbol emission, and every transition between two ordinary states, have the same probabilities. Once we have made this transformation, every path has the same probability, so we only need to count the number of paths produced by each particular annotation.

To understand the HMM, we relate it to the layered graph of Fig. 10. Each path in this layered graph is represented by a path in the HMM, and the coloring of the path in the layered graph will be represented by the annotation (state labels) in the HMM.

For a given annotation and given formula with $n$ variables and $m$ clauses, let $x$ be the truth assignment implied by the annotation of the first block, and $y_i$ be the truth assignment for the annotation of block $i + 1$, representing the $i$th clause.

Sub-model $A$ always carries exactly one path, corresponding to the top-most level in the layered graph. From time to time, paths split off from the path carried by sub-model $A$, and propagate to other blocks. All non-zero probability

paths must end in the stop state after $(m+1)(n+1)+1$ steps since analyzed sequences always end with !, emitted only by that state.

The sub-models $B, C, D$ of the HMM contribute one path to the final annotation if all block annotations represent satisfying assignments; thus, they embody the SAT modules of the graph.

At the beginning of each clause, if all previous clauses were satisfied, one path is in sub-model $C$. It is transferred to sub-model $D$ once a variable is found whose truth assignment, as represented in the annotation, satisfies the clause. After the clause is terminated with the symbol \$, this path is transferred back to the sub-model $B$, if there was a satisfied variable, or it is discontinued in sub-model $C$, if the clause is not satisfied. Thus, the function of sub-model $B$ is characterized by the following lemma.

**Lemma 6.** *Upon emission of the symbol \$, terminating block $i+1$ representing the $i$th clause, sub-model $B$ contains one path if the first $i$ clauses are satisfied by their assignments $y_1, \ldots, y_i$, or zero paths otherwise.*

Sub-models $E, F, G, H, J, L, M$, and $N$ enforce that all clause blocks of the input sequence are annotated with the same value assignment of the variables. The combination of these sub-models mimics the function of the components ENCODE$(x)$ and EQ$(x, y_i)$ in Fig. 10. In particular, sub-model $E$ corresponds to $C(z,k)$, sub-model $F$ corresponds to $B(k)$, and sub-model $J$ corresponds to $D(z,k)$ from recurrences (5)–(7) on page 1069. The following lemma characterizes the intermediate number of paths in sub-model $N$ and its correctness follows directly from these recurrences.

**Lemma 7.** *After emitting the symbol \$ terminating block $i+1$ of the sequence corresponding to the $i$th clause of the formula, the number of paths contained in sub-model $N$ is*

$$\sum_{j=1}^{i} 2K(n) - (x - y_i)^2. \tag{8}$$

Upon emitting the symbol ! terminating the sequence, the final "stop" state will receive all the paths accumulated in sub-model $N$. If every clause has the same assignment $y_i = x$, sub-model $N$ contributes $2mK(n)$ paths; otherwise, it contributes some smaller number. The "stop" state also receives one path from sub-model $B$ if all clauses are satisfied by their assignments.

Therefore, a satisfying annotation that is consistent over all the clauses will yield $2mK(n)+1$ paths. Any other annotation yields a smaller number of paths, so a satisfying consistent annotation is the most probable annotation, if such an annotation exists. Therefore, if we can solve the most probable annotation problem for the HMM in Fig. 16, we can use it to solve SAT in polynomial time. After removing the silent states and introducing the error state, the resulting HMM has 34 states.

We will see in the following section that this HMM is hard to decode because states with different labels are distributed all over the HMM, with many edges between states of different labels. For HMMs with only a few edges leading between states of different labels, we will introduce polynomial-time algorithms.

## 5. Computing the most probable annotation

In the previous section, we showed that it is NP-hard to compute the most probable annotation for some HMMs. However, for special classes of HMMs, the most probable annotation can be computed efficiently. For example, HMMs without the multiple path problem have one state path per annotation. The Viterbi algorithm, finds the maximum probability path in $O(nm^2)$ time, where $n$ is the length of the sequence, and $m$ is the number of states; this path's labeling is the most probable.

In this section, we give a collection of algorithms with increasing running time that can compute the maximum probability annotation for increasingly wider classes of HMMs; the Viterbi algorithm is the first and fastest such algorithm. For each algorithm, we give a sufficient condition for membership in the class of HMM topologies that can be decoded properly using the algorithm. While our algorithms may not always find the most probable annotation, they will return an annotation with probability at least as high as the probability of the most probable state path, even if the input HMM does not belong to the class of HMMs guaranteed to be decoded correctly by the algorithm.


Fig. 17. An HMM with two critical edges, $A \to B$ and $B \to A$.

### 5.1. Most probable extended annotation

So far, we have been considering the partition of all state paths into equivalence classes corresponding to paths with a common labeling, and our goal has been to search for the equivalence class (annotation) of highest probability. Now, we consider a possibly finer partition of the set of all paths and call the equivalence classes in this new partition *extended annotations*. The additional division is based on which edges in the path join states with different labels.

In the following definitions we assume for ease of presentation that the HMM has a designated start state $s$ and a designated final state $f$. Let $\mathrm{in}(u)$ for any state $u$ be the set of states with a transition into state $u$.

**Definition 8** *(Extended annotation).* A *critical edge* is a transition between two states of different label. The *extended annotation* of a state path $\pi_1\pi_2\ldots\pi_n$ is the pair $(L, C)$, where $L = \lambda_1, \lambda_2, \ldots, \lambda_n$ is the sequence of labels of each state in the path and $C = c_1, c_2, \ldots, c_k$ is the sequence of all critical edges in the path.

For example, the HMM in Fig. 17 has three states, two different labels, and two critical edges: $A \to B$ and $B \to A$. The state paths $ABCB$ and $ABBB$ both have the same extended annotation, (□ ■ ■ ■, $A \to B$).

The following extended Viterbi algorithm (EVA) computes the most probable extended annotation.

**Theorem 9** *(Extended Viterbi algorithm). For a given sequence $S = x_1 \ldots x_n$ and an HMM with $m$ states, it is possible to compute the most probable extended annotation of $S$ in time $O(n^2m^3)$.*

**Proof.** Our algorithm is an extension to the classical Viterbi algorithm, which computes the most probable state path in the HMM using dynamic programming. In the Viterbi algorithm, we compute values $V[u, i]$, the maximum probability of a state path for the sequence $x_1 \ldots x_i$ over all paths ending in state $u$, or $\max \Pr(x_1 \ldots x_i, \pi_1 = s, \pi_2, \ldots, \pi_{n-1}, \pi_i = u)$.

To compute a particular value of $V[u, i]$, the dynamic programming algorithm uses the following recurrence, examining all possible options for the second-to-last state:

$$V[u, i] = \max_{v \in \mathrm{in}(u)} V[v, i-1] \cdot a(v, u) \cdot e_u(x_i). \tag{9}$$

In the extended Viterbi algorithm (EVA), we instead compute maximum probability extended annotations for the first $i$ characters, ending in a particular state $u$; let $L[u, i]$ be defined as $L[u, i] = \max \Pr(x_1 \ldots x_i, (L, C), \pi_i = u)$, where the maximum is taken over all extended annotations $(L, C)$ of sequence $x_1 \ldots x_i$ where the generating process starts in state $s$ and ends in state $u$.

As in the Viterbi algorithm for HMMs with explicit state duration [17], we examine all possible durations of the last segment with the same label. Instead of choosing the single most probable path in that segment, we compute the sum of all possible state paths in this segment. If the segment starts at position $j \leqslant i$ of the sequence, let $P[v, u, j, i]$ be this sum; it is the probability of generating the sequence $x_j \ldots x_i$, starting in state $v$, and ending in state $u$, using only states with label $\lambda(u)$. This gives the following recurrence:

$$L[u, i] = \max_{j \leqslant i} \max_{v:\ \lambda(v) = \lambda(u)} \max_{w \in \mathrm{in}(v):\ \lambda(w) \neq \lambda(v)} \big(L[w, j-1] \cdot a(w, v) \cdot P[v, u, j, i]\big). \tag{10}$$

We compute values of $L$ in order of increasing $i$. For each $i$, we compute all relevant values of $P(v, u, j, i)$ in order of decreasing $j$, using the following recurrence:

$$P[v, u, j, i] = \sum_{w:\ v \in \mathrm{in}(w),\ \lambda(v) = \lambda(w)} e_v(x_j) \cdot a(v, w) \cdot P[w, u, j+1, i]. \tag{11}$$

When the computation of $L$ is finished, we can reconstruct the most probable extended annotation via backtracking, as for the Viterbi algorithm. □

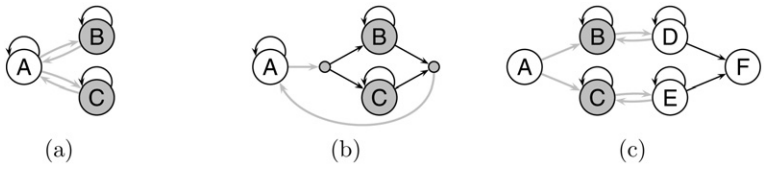



Fig. 18. Usefulness of silent states.

The probability of the extended annotation returned by the algorithm is always at least as high as the probability of the most probable state path $\Pi$ found by the Viterbi algorithm, as the probability of the extended annotation corresponding to $\Pi$ must be at least as high as the probability of $\Pi$ itself.

### 5.2. The critical edge condition

In this section, we study a class of HMMs for which we can prove that the algorithm defined above computes the most probable annotation. This class contains many HMMs with the multiple path problem that cannot be decoded simply by using the Viterbi algorithm. Here is a sufficient condition for membership in this class.

**Definition 10.** An HMM *satisfies the critical edge condition* for an input sequence $s$ if any two paths for $s$ with the same annotation have the same sequence of critical edges. An HMM *satisfies the critical edge condition* in general if for all input sequences $s$, the critical edge condition is satisfied.

The significance of the critical edge condition is shown by the following claim.

**Corollary 11.** *If an HMM satisfies the critical edge condition for a sequence $s$, then the EVA computes the most probable annotation of sequence $s$.*

The hidden Markov model in Fig. 17 clearly satisfies the critical edge condition, since for each transfer between the two labels there is only one critical edge which can be used. Several of the HMMs from Section 3 with the multiple path problem satisfy the critical edge condition, specifically those in Figs. 3(b), 4(b), 5(b), and 6.

On the other hand, the HMM in Fig. 18(a) does not satisfy the critical edge condition: the annotation □ ■ □ has two possible extended annotations: (□ ■ □, $A \to B$, $B \to A$) and (□ ■ □, $A \to C$, $C \to A$). However, we can add silent states to construct the equivalent HMM, in Fig. 18(b) that satisfies the critical edge condition, as we leave each label class via a common exit. Similarly, the HMM in Fig. 2(b) can be modified by addition of two silent states to satisfy the critical edge condition. Thus, in our case, the silent states are a crucial modeling tool. However, the silent state technique is not universal: the HMM in Fig. 18(c) cannot be transformed to comply with the condition.

We can test algorithmically whether a given HMM topology (where we only consider the existence or absence of transitions with non-zero probability) will satisfy the critical edge condition for every input sequence. We first use depth-first search to build a set $S_s$ of all pairs of states that are reachable from the start state by the same annotation. We start from the pair $(s, s) \in S_s$, and in each step we add a new pair $(u, v)$ if $\lambda(u) = \lambda(v)$, and there exists $(u', v') \in S_s$ such that $u' \in \text{in}(u)$ and $v' \in \text{in}(v)$. Similarly, we also build a set $S_f$ of all pairs of states from which the final state can be reached by the same annotation. For the critical condition to be violated, there must exist a pair $(u, v) \in S_s$ and $(u', v') \in S_f$ such that $\lambda(u) \neq \lambda(u')$, and $(u, u')$ and $(v, v')$ are two different transitions. The algorithm takes $O(m^4)$ time.

It is possible to modify this verification algorithm to verify the critical edge condition in $O(m^4|\Sigma|^2)$ time, if the emission probabilities are given. Note that this test may yield a different result, since some states may not produce some alphabet symbols, so two different paths with the same extended annotation cannot generate the same string; hence, this extension identifies even more HMMs that satisfy the condition.

And finally, we can also verify the condition for a given HMM and input string in $O(nm^4)$ time. In that case, we build a set of state pairs that can be reached by the same annotation for each position in the sequence.

A simplified version of these algorithms can also test if an HMM has the multiple path problem. We build the sets $S_s$ and $S_f$ as before and then test if there is a pair $(u, v)$ in the intersection of $S_s$ and $S_f$ such that $u \neq v$. Finding

such a pair means that for the same annotation there exist two state paths: one going through state $u$ and one through state $v$.

### 5.3. Generalizing the EVA and the critical edge condition

We have seen two classes of HMMs for which the most probable annotation can be found in polynomial time. If there is a one-to-one mapping between annotations and state paths, then the Viterbi algorithm, whose runtime is linear in the sequence length $n$, finds the most probable annotation. Or, if the HMM comes from the larger class of HMMs that satisfy the critical edge condition, then the extended Viterbi algorithm (EVA), whose runtime is quadratic in the sequence length, finds the most probable annotation.

In this section we extend the critical edge condition and the EVA to growing classes of HMMs. As the configuration of critical edges in HMM gets more complex, the running time of the decoding algorithms also increase.

**Definition 12** *(Generalized extended annotation).* A *generalized extended annotation* is a pair $(L, C)$, where $L = \lambda_1, \lambda_2, \ldots, \lambda_n$ is a sequence of labels, and $C = c_1, c_2, \ldots, c_k$ contains for every transition between different labels in $L$ either a critical edge that can be used for that transition or the symbol $*$ (a *masked critical edge*).

A generalized extended annotation *matches the state path* $\pi_1\pi_2\ldots\pi_n$ if the labeling $L$ corresponds to the labels of the states, and if every critical edge in $C$ matches the one used on the path (masked critical edges can be arbitrary).

A generalized extended annotation is *of order* $d$ if there are no blocks of $d$ consecutive masked critical edges in it.

One path can be matched by several generalized extended annotations, since different critical edges can be masked.

Our extended Viterbi algorithm (EVA) can be extended to the domain of generalized extended annotations as well. The EVA of Theorem 9 is a special case of the following algorithm for $d = 1$.

**Theorem 13** *(Generalized EVA). For a given sequence $S = x_1\ldots x_n$ and an HMM with $m$ states, it is possible to compute the most probable generalized extended annotation of order $d$ in time $O(n^{d+1}m^{d+2})$. We call the corresponding algorithm the* generalized EVA of order $d$.

**Proof.** In Theorem 9, we used dynamic programming to compute the values of $L[u, i]$, which is the maximum probability of an extended annotation of the first $i$ symbols of the sequence, ending at state $u$. Our recurrence decomposed the problem of computing $L[u, i]$ into subproblems depending on the last critical edge used.

We can further modify the algorithm to account for short blocks of masked critical edges in generalized extended annotations. The dynamic programming algorithm will compute values of $L_d[u, i]$: the probability of the most probable *generalized* extended annotation of *order at most* $d$ of the first $i$ symbols ending in state $u$. We decompose the problem into subproblems depending on the position $j$ of the last unmasked critical edge. We also consider all possible annotations of the region of the sequence $x_j\ldots x_i$ with at most $d-1$ transitions between different labels. In particular:

$$L_d[u,i] = \max_{j\leqslant i,\, v} \max_{w\in \mathrm{in}(v):\ \lambda(v)\neq\lambda(w)} L_d[w, j-1]\cdot a(w,v)\cdot \max_{\lambda_j,\ldots,\lambda_i\in\mathcal{L}_d(j-i+1)} P_d(v, j, u, i, \lambda_j\ldots\lambda_i), \tag{12}$$

where $\mathcal{L}_d(k)$ is the set of labellings of length $k$ with at most $d-1$ transitions between different labels, and $P_d(v, j, u, i, \lambda_j\ldots\lambda_i)$ is the probability of labeling the symbols $x_j\ldots x_i$ with labels $\lambda_j\ldots\lambda_i$, if the model starts in state $v$ for position $j$ and finishes in state $u$ with position $i$. For a given labeling, this probability can be computed in $O(nm^2)$ with the backward algorithm.

For each tuple $(v, j, u, i)$ there are $O(n^{d-1}m^{d-2})$ possible labellings $\lambda_j\ldots\lambda_i$, and for each pair $u, i$ there are $O(nm^2)$ values of $v, w$ and $j$. Therefore, to compute the $O(nm)$ values of $L_d[u, i]$, we need a running time of $O(n^{d+2}m^{d+3})$.

This running time can be reduced by careful organization of how the values of $P_d(v, j, u, i, \lambda_j\ldots\lambda_i)$ are computed. The labellings that need to be explored for a given pair $(u, i)$ can be organized in a tree. The root is a labeling with only the single label $\lambda(u)$. The children of each node either extend this labeling backward with the same label as their parent, or change the label to a different label; such extensions are done until either the beginning of the sequence is reached, or the threshold $d-1$ of allowed label changes is reached. If the computation of the values of $P_d$ is organized

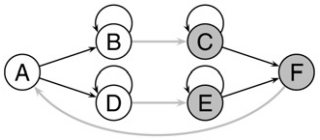



Fig. 19. An HMM that satisfies the generalized critical edge condition of degree 2.

using this tree, only $O(m^2)$ time is needed to compute the desired probabilities for each labeling from the probabilities of its parent labeling. Thus, for a given $u$ and $i$, we can compute all relevant values of $P_d$ in $O(n^d m^{d+1})$ time and the overall running time is $O(n^{d+1}m^{d+2})$. □

We can now generalize the critical edge condition, which gave a sufficient condition for membership in the class of models for which the EVA gave maximum probability annotations, giving a sufficient condition for the class of HMM topologies that can be decoded by the generalized EVA.

**Definition 14** *(Consensus generalized extended annotation).* For a set of $\ell$ extended annotations $\{(L, C_1), (L, C_2), \ldots, (L, C_\ell)\}$, all sharing the same labeling $L$, but differing only in their critical edges, their *consensus generalized extended annotation* is the generalized extended annotation $(L, (c_1, c_2, \ldots, c_k))$ where $c_i = c$, if the $i$th critical edge of all of $C_1, C_2, \ldots, C_\ell$ is the same edge $c$, and $c_i$ is masked to $*$ otherwise.

**Definition 15** *(Generalized critical edge condition).* An HMM *satisfies the generalized critical edge condition of order $d$*, if for any sequence $s$, and any non-zero probability annotation $L$ of $s$, the consensus generalized extended annotation of all non-zero probability extended annotations $(L, C)$ of $s$ is of order at most $d$.

The condition holds if for every non-zero probability annotation $L$ of sequence $s$, there can never be a block of $d$ transitions between labels where none of these transitions is restricted to being only a single edge. Note that the critical edge condition is a special case of the generalized critical edge condition where $d = 1$.

**Corollary 16.** *If an HMM satisfies the generalized critical edge condition of order $d$, then the generalized EVA of order $d$ finds the generalized extended annotation corresponding to the most probable annotation.*

The HMM in Fig. 19 does not satisfy the critical edge condition: there are two critical edges leading from states with the white label to states with the gray label, $B \to C$ and $D \to E$. It does satisfy the generalized critical edge condition for $d = 2$. For example, the annotation ■ □□ ■ ■ ■ ■ □□□ has only two possible sequences of critical edges in an extended annotation: $(F \to A, B \to C, F \to A)$ and $(F \to A, D \to E, F \to A)$. This yields the consensus annotation $(F \to A, *, F \to A)$, consistent with the generalized critical edge condition of order $d = 2$. Corollary 16 shows that the most probable labeling for this HMM can be found by the generalized EVA in running time $O(n^3m^4)$.

### 5.4. Beyond the generalized critical edge condition

Figure 20 shows two small HMMs which do not satisfy the generalized critical condition for any constant $d$, and therefore cannot be exactly decoded by the generalized EVA algorithm. The HMM in Fig. 20(a) can be decoded in polynomial time by a completely different strategy.

HMMs like the one in Fig. 20(a) are characterized by being divided into $k$ components, where each component contains exactly one state with each label (though that state may have no incoming transitions), and there are no transitions between different components. We pick which component generates a sequence using a single silent start

(a) can be decoded in polynomial time (b) no known polynomial-time decoding

Fig. 20. Two small HMMs that cannot be decoded by the generalized EVA.

state connected to at least one state in each component. For example, the HMM in Fig. 20(a) has two components with two states each.

Let $\ell$ be the number of labels and $\sigma$ the size of the alphabet. Let $v_{j,\lambda}$ be the state of label $\lambda$ in component $j$. Any annotation in such an HMM has at most one corresponding path within each component. The probability of such an annotation is

$$\Pr(x_1 \dots x_n, \lambda_1 \dots \lambda_n) = \sum_{j=1}^{k} \left( a(s, v_{j,\lambda_1}) \prod_{\lambda,x} e_{v_{j,\lambda}}(x)^{f(\lambda,x)} \prod_{\lambda,\lambda'} a(v_{j,\lambda}, v_{j,\lambda'})^{f(\lambda,\lambda')} \right), \tag{13}$$

where $f(\lambda, x)$ is the number of positions where symbol $x$ is labeled by label $\lambda$, that is, where $\lambda_i = \lambda$ and $x_i = x$, and $f(\lambda, \lambda')$ is the number of positions where label $\lambda$ is followed by label $\lambda'$. The probability of a given annotation is determined solely by $\lambda_1$ and the configuration of the frequencies $f(\lambda, x)$ and $f(\lambda, \lambda')$. Each of these counts is between 0 and $n$. Therefore, even though there are $\ell^n$ annotations, they can have at most $\ell \cdot (n+1)^{\ell\sigma+\ell^2}$ different probabilities.

For a given string, we can compute the set of all possible configurations of these frequencies by a simple search through all prefixes of the string. For each of these configurations, we can evaluate the probability, and choose the best configuration, and consequently compute the most probable annotation. If $\sigma$ and $\ell$ are constants, this algorithm works in polynomial time.

Unfortunately, this strategy does not work for the HMM shown in Fig. 20(b), since in this HMM, the probability of an annotation is a sum of exponentially many path probabilities. We do not know a polynomial-time algorithm to decode this HMM.

## 6. Conclusion

We have investigated the most probable annotation problem in HMMs. We showed that the problem is NP-hard even for a specific small HMM, in contrast to the previous NP-hardness proof by Lyngsø and Pedersen [15], where the constructed HMM depended on the input SAT instance.

Even though the problem is NP-hard in general, we can compute the most probable annotation for many HMMs in polynomial time. If there is only a single feasible state path for every possible annotation, the problem can be solved by the Viterbi algorithm. Otherwise, the HMMs have what we have termed the multiple path problem, and we need to use different methods to decode them.

We have presented several biological applications where the multiple path problem arises: predicting the topology of transmembrane proteins, predicting protein coding regions in mRNA sequences, and finding genes in DNA sequences. In all of these applications, the multiple path problem can be solved by applying our $O(n^2m^3)$ extended Viterbi algorithm, either directly or using silent states. This is an acceptable runtime for protein and mRNA sequences.

In contrast, gene finding in DNA sequences is typically performed on longer sequences, potentially on entire chromosomes. A runtime quadratic in the lengths of these sequences is not feasible, since for example the smallest human chromosome is approximately fifty million symbols long. To capture the C/T-rich tail in the intron model (as seen in Fig. 5) and still keep the running time linear in the length of the sequence, HMM-based gene finders typically model this intron tail by a state generating a fixed number of nucleotides at the end of the intron [2,3,20]. Such a model does not have the multiple path problem, and therefore can be decoded by the linear-time Viterbi algorithm. We demonstrated by a simple experiment that when computing the most probable annotation is infeasible, simplifications that exclude the multiple path problem may be preferable to using the Viterbi decoding algorithm on the more complex model.

We can further generalize our extended Viterbi algorithm and the critical edge condition and use increasingly slower decoding algorithms to decode increasingly wider classes of HMMs. Finally, Fig. 20(a) shows an example of an HMM that cannot be decoded by any of these algorithms, though it can be decoded in polynomial time by a completely different approach.

The multiple path problem in HMMs is closely related to structural ambiguity in stochastic context-free grammars (SCFG). Such grammars are commonly used in bioinformatics for RNA secondary structure prediction [7,12]. Dowell and Eddy [5] demonstrated that ambiguity can cause significant deterioration of prediction accuracy in this application.

In general, the problem of ambiguity is harder in SCFGs than in HMMs. Determining whether a given SCFG is structurally ambiguous is undecidable [18]; the best positive result uses techniques from compiler construction to

determine that some commonly used SCFGs are unambiguous [1,18]. On the other hand, we can test whether an HMM has the multiple path problem in $O(m^4)$ time (see Section 5.2). Because HMMs are a special case of SCFGs, our hardness results also show that at least for some structurally ambiguous SCFGs, determining the most probable structure is NP-hard.

Several problems remain open. First, we do not know at present any polynomial-time algorithm for finding the most probable annotation for the model shown in Fig. 20(b). This is one of the smallest such examples, since all HMMs with at most three non-silent states can be decoded by the techniques we have shown. Is decoding this HMM NP-hard, or can we find yet another class of polynomial-time decoding algorithms? Second, are there HMM topologies (other than ones without the multiple path problem) that can be decoded in sub-quadratic time? Such models may be useful in applications where the input sequence is long. Finally, the HMM we have constructed in Section 4 can be very well approximated: any annotation that satisfies each clause by a potentially different truth assignment differs from the optimal annotation by at most one path. In contrast, Lyngsø and Pedersen [15] show that the most probable annotation is NP-hard to approximate within any constant when the HMM is part of the input. The logical question is whether there is a fixed HMM that is hard to approximate.

## Acknowledgments

This research was supported by grants from the National Science and Engineering Research Council of Canada. Most of the work was done while B.B. and T.V. were at the University of Waterloo. The authors thank Ming Li and Prabhakar Ragde for fruitful discussions, and the anonymous referees for their comments that helped to improve this article. The early version of this paper appeared in Workshop on Algorithms in Bioinformatics 2004.

## References

[1] C. Brabrand, R. Giegerich, A. Møller, Analyzing ambiguity of context-free grammars, Tech. Rep. RS-06-09, BRICS, May 2006.
[2] B. Brejová, D.G. Brown, M. Li, T. Vinař, ExonHunter: A comprehensive approach to gene finding, in: Intelligent Systems for Molecular Biology, ISMB 2005, Bioinformatics 21 (Suppl 1) (2005) i57–i65.
[3] C.B. Burge, Identification of genes in human genomic DNA, PhD thesis, Department of Mathematics, Stanford University, 1997.
[4] F. Casacuberta, C. de la Higuera, Computational complexity of problems on probabilistic grammars and transducers, in: ICGI 2000: Grammatical Inference: Algorithms and Applications, in: Lecture Notes in Comput. Sci., vol. 1891, Springer, 2000, pp. 15–24.
[5] R.D. Dowell, S.R. Eddy, Evaluation of several lightweight stochastic context-free grammars for RNA secondary structure prediction, BMC Bioinformatics 5 (2004) 71.
[6] R. Durbin, S. Eddy, A. Krogh, G. Mitchison, Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids, Cambridge University Press, 1998.
[7] S.R. Eddy, R. Durbin, RNA sequence analysis using covariance models, Nucleic Acids Res. 22 (11) (1994) 2079–2088.
[8] D. Eppstein, Finding the $k$ shortest paths, SIAM J. Comput. 28 (2) (1998) 652–673.
[9] G.D. Forney, The Viterbi algorithm, Proc. IEEE 61 (1973) 268–278.
[10] C. Iseli, C.V. Jongeneel, P. Bucher, ESTScan: A program for detecting, evaluating, and reconstructing potential coding regions in EST sequences, in: ISMB 1999: Seventh International Conference on Intelligent Systems for Molecular Biology, 1999, pp. 138–148.
[11] M.T. Johnson, Capacity and complexity of HMM duration modeling techniques, IEEE Signal Process. Lett. 12 (5) (2005) 407–410.
[12] B. Knudsen, J. Hein, Pfold: RNA secondary structure prediction using stochastic context-free grammars, Nucleic Acids Res. 31 (13) (2003) 3423–3428.
[13] A. Krogh, Two methods for improving performance of an HMM and their application for gene finding, in: ISMB 1997: Proceedings of the 5th International Conference on Intelligent Systems for Molecular Biology, 1997, pp. 179–186.
[14] A. Krogh, B. Larsson, G. von Heijne, E.L. Sonnhammer, Predicting transmembrane protein topology with a hidden Markov model: Application to complete genomes, J. Mol. Biol. 305 (3) (2001) 567–570.
[15] R.B. Lyngsø, C.N.S. Pedersen, The consensus string problem and the complexity of comparing hidden Markov models, J. Comput. System Sci. 65 (3) (2002) 545–569.
[16] P.L. Martelli, P. Fariselli, A. Krogh, R. Casadio, A sequence-profile-based HMM for predicting and discriminating beta barrel membrane proteins, Bioinformatics 18 (S1) (2002) S46–S53.
[17] L.R. Rabiner, A tutorial on hidden Markov models and selected applications in speech recognition, Proc. IEEE 77 (2) (1989) 257–285.
[18] J. Reeder, P. Steffen, R. Giegerich, Effective ambiguity checking in biosequence analysis, BMC Bioinformatics 6 (2005) 153.
[19] R. Schwartz, Y.-L. Chow, The $N$-best algorithms: An efficient and exact procedure for finding the $N$ most likely sentence hypotheses, in: ICASSP: Acoustics, Speech, and Signal Processing., vol. 1, 1990, pp. 81–84.
[20] M. Stanke, S. Waack, Gene prediction with a hidden Markov model and a new intron submodel, Bioinformatics 19 (S2) (2003) II215–II225.
[21] T. Vinař, Enhancements to hidden Markov models for gene finding and other biological applications, PhD thesis, University of Waterloo, 2005.