On-line Viterbi Algorithm for Analysis of Long Biological Sequences

Rastislav Šrámek[1], Broňa Brejová[2], and Tomáš Vinař[2]

[1] Department of Computer Science, Comenius University,
842 48 Bratislava, Slovakia, e-mail: rasto@ksp.sk
[2] Department of Biological Statistics and Computational Biology, Cornell University,
Ithaca, NY 14853, USA, e-mail: {bb248,tv35}@cornell.edu

Abstract. Hidden Markov models (HMMs) are routinely used for analysis of long genomic sequences to identify various features such as genes, CpG islands, and conserved elements. A commonly used Viterbi algorithm requires $O(mn)$ memory to annotate a sequence of length n with an m-state HMM, which is impractical for analyzing whole chromosomes. In this paper, we introduce the on-line Viterbi algorithm for decoding HMMs in much smaller space. Our analysis shows that our algorithm has the expected maximum memory $\Theta(m \log n)$ on two-state HMMs. We also experimentally demonstrate that our algorithm significantly reduces memory of decoding a simple HMM for gene finding on both simulated and real DNA sequences, without a significant slow-down compared to the classical Viterbi algorithm.

Keywords: biological sequence analysis, hidden Markov models, on-line algorithms, Viterbi algorithm, gene finding

1 Introduction

Hidden Markov models (HMMs) are generative probabilistic models that have been successfully used for annotation of protein and DNA sequences. Their numerous applications in bioinformatics include gene finding [1], protein secondary structure prediction [2], promoter detection [3], and CpG island detection [4]. More complex phylogenetic HMMs are used to analyze multiple sequence alignments in comparative gene finding [5] and detection of conserved elements [6]. The linear-time Viterbi algorithm [7] is the most commonly used algorithm for these tasks. Unfortunately, the space required by the Viterbi algorithm grows linearly with the length of the sequence (with a high constant factor), which makes it unsuitable for analysis of very long sequences, such as whole chromosomes or whole-genome multiple alignments. In this paper, we address this problem by proposing an on-line Viterbi algorithm that on average requires much less memory and that can even annotate continuous streams of data on-line without reading the complete input sequence first.

An HMM, composed of states and transitions, is a probabilistic model that generates sequences over a given alphabet. In each step of this generative process,

the current state generates one symbol of the sequence according to the *emission probabilities* associated with that state. Then, an outgoing transition is randomly chosen according to the *transition probability table*, and this transition is followed to the new state. This process is repeated until the whole sequence is generated.

The states of the HMM represent distinct features of the observed sequences (such as protein coding and non-coding sequences in a genome), and the emission probabilities in each state represent statistical properties of these features. The HMM thus defines a joint probability $\Pr(X, S)$ over all possible sequences X and all *state paths* S through the HMM that could generate these sequences. To annotate a given sequence X, we find the state path S that maximizes this joint probability. For example, in an HMM with one state for protein-coding sequences, and one state for non-coding sequences, the most probable state path marks each symbol of sequence X as either protein coding or non-coding.

To compute the most probable state path, we use the Viterbi dynamic programming algorithm [7]. For every prefix $X_1 \dots X_i$ of sequence X and for every state j, we compute the most probable state path generating this prefix ending in state j. We store the probability of this path in table $P(i, j)$ and its second last state in table $B(i, j)$. These values can be computed from left to right, using the recurrence $P(i, j) = \max_k\{P(i-1, k) \cdot t_k(j) \cdot e_j(X_i)\}$, where $t_k(j)$ is the transition probability from state k to state j, and $e_j(X_i)$ is the emission probability of symbol X_i in state j. Back pointer $B(i, j)$ is the value of k that maximizes $P(i, j)$. After computing these values, we can recover the most probable state path $S = s_1, \dots, s_n$ by setting the last state as $s_n = \arg\max_k\{P(n, k)\}$, and then following the back pointers $B(i, j)$ from right to left (i.e., $s_i = B(i{+}1, s_{i+1})$). For an HMM with m states and a sequence X of length n, the running time of the Viterbi algorithm is $\Theta(nm^2)$, and the space is $\Theta(nm)$.

This algorithm is well suited for sequences and models of moderate size. However, to annotate all 250 million symbols of the human chromosome 1 with a gene finding HMM consisting of hundred states, we would require 25 GB of memory to store the back pointers $B(i, j)$. This is clearly impractical on most computational platforms.

Several solutions are used in practice to overcome this problem. For example, most practical gene finding programs process only sequences of limited size. The long input sequence is split into several shorter sequences which are processed separately. Afterwards, the results are merged and conflicts are resolved heuristically. This approach leads to suboptimal solutions, especially if the genes we are looking for cross the boundaries of the split.

Grice et al. [8] proposed a checkpointing algorithm that trades running time for space. We divide the input sequence into K blocks of L symbols, and during the forward pass, we only keep the first column of each block. To obtain the most probable state path, we recompute the last block and use the back pointers to recover the last L states of the path, as well as the last state of the previous block. This information can now be used to recompute the most probable state path within the previous block in the same way, and the process is repeated for all blocks. If we set $K = L = \sqrt{n}$, this algorithm only requires $\Theta(n + \sqrt{n}m)$

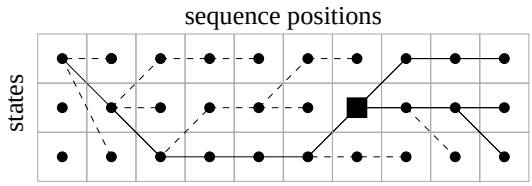

Fig. 1. Example of the back pointer tree structure. Dashed lines mark the edges that cannot be part of the most probable state path. The square marks the coalescence point of the remaining paths.

memory at the cost of two-fold slow-down compared to the Viterbi algorithm, since every value of $P(i, j)$ is computed twice. Checkpointing can be further generalized to trade L-fold slow-down for memory of $\Theta(n + \sqrt[L]{n}m)$ [9, 10].

In this paper, we propose and analyze an on-line Viterbi algorithm that does not use fixed amount of memory for a given sequence. Instead, the amount of memory varies depending on the properties of the HMM and input sequence. In the worst case, our algorithm still requires $\Theta(nm)$ memory; however, in practice the requirements are much lower. We prove, using results for random walks and theory of extreme values, that in simple cases the expected space for a sequence of length n is as low as $\Theta(m \log n)$. We also experimentally demonstrate that the memory requirements are low for more complex HMMs.

2 On-line Viterbi algorithm

In our algorithm, we represent the back pointer matrix B in the Viterbi algorithm by a tree structure (see [7]), with node (i, j) for each sequence position i and state j. Parent of node (i, j) is the node $(i-1, B(i, j))$. In this data structure, the most probable state path is the path from the leaf node (n, j) with the highest probability $P(n, j)$ to the root of the tree (see Figure 1).

This tree is built as the Viterbi algorithm progresses from left to right. After computing column i, all edges that do not lie on one of the paths ending column i can be removed; these edges will not be used in the most probable path [11]. The remaining m paths represent all possible initial segments of the most probable state path. These paths are not necessarily edge disjoint; in fact, often all the paths share the same prefix up to some node that we call a *coalescence point* (see Figure 1).

Left of the coalescence point there is only a single candidate for the initial segment of the most probable state path. Therefore we can output this segment and remove all edges and nodes of the tree up to the coalescence point. Forney [7] describes an algorithm that after processing D symbols of the input sequence checks whether a coalescence point has been reached; in such case, the initial segment of the most probable state path is outputted. If the coalescence point was not reached, one potential initial segment is chosen heuristically. Several

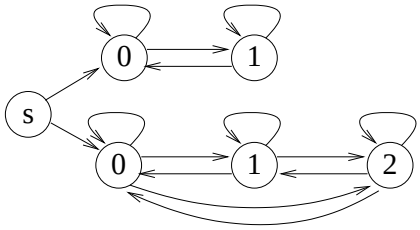

Fig. 2. An HMM requiring $\Omega(n)$ space. Every state emits only the symbol shown, with probability 1. Transition probability is evenly divided among transitions outgoing from a given state. For sequences of the form $s\{0,1\}^n\{0,2\}$, any correct decoding algorithm must retain some representation of the input before discovering whether the last symbol is 0 or 2.

studies [12, 13] suggest how to choose D to limit the expected error caused by such heuristic steps in the context of convolution codes.

Here we show how to detect the existence of a coalescence point dynamically without introducing significant overhead to the whole computation. We maintain a compressed version of the back pointer tree, where we omit all internal nodes that have less than two children. Any path consisting of such nodes will be contracted to a single edge. This compressed tree has m leaves and at most $m-1$ internal nodes. Each node stores the number of its children and a pointer to its parent node. We also keep a linked list of all the nodes of the compressed tree ordered by the sequence position. Finally, we also keep the list of pointers to all the leaves.

When processing the k-th sequence position in the Viterbi algorithm, we update the compressed tree as follows. First, we create a new leaf for each node at position i, link it to its parent (one of the former leaves), and insert it into the linked list. Once these new leaves are created, we remove all the former leaves that have no children, and recursively all of their ancestors that would not have any children.

Finally, we need to compress the new tree: we examine all the nodes in the linked list in order of decreasing sequence position. If the node has zero or one child and is not a current leaf, we simply delete it. For each leaf or node that has at least two children, we follow the parent links until we find its first ancestor (if any) that has at least two children and link the current node directly to that ancestor. A node (ℓ, j) that does not have an ancestor with at least two children is the coalescence point; it will become a new root. We can output the most probable state path for all sequence positions up to ℓ, and remove all results of computation for these positions from memory.

The running time of this update is $O(m)$ per sequence position, and the representation of the compressed tree takes $O(m)$ space. Thus the asymptotic running time of the Viterbi algorithm is not increased by the maintenance of the compressed tree. Moreover, we have implemented both the standard Viterbi algorithm and our new on-line extension, and the time measurements suggest that the overhead required for the compressed tree updates is less than 5%.

The worst-case space required by this algorithm is still $O(nm)$. In fact, any algorithm that correctly finds the most probable state path for the HMM shown in Figure 2 requires at least $\Omega(n)$ space in the worst case.

However, our algorithm rarely requires linear space for realistic data; the space changes dynamically depending on the input. In the next section, we show that for simple HMMs the expected maximum space required for processing sequence of length n is $\Theta(m \log n)$. This is much better than checkpointing, which requires space of $\Theta(n + m\sqrt{n})$ with a significant increase in running time. We conjecture that this trend extends to more complex cases. We also present experimental results on a gene finding HMM and real DNA sequences showing that the on-line Viterbi algorithm leads to significant savings in memory.

Another advantage of our algorithm is that it can construct initial segments of the most probable state path before the whole input sequence is read. This feature makes it ideal for on-line processing of signal streams (such as sensor readings).

3 Memory requirements of the on-line Viterbi algorithm

In this section, we analyze space requirements of the on-line Viterbi algorithm. The space is variable throughout the execution of the algorithm, but of special interest are asymptotic bounds on the expected maximum memory while decoding a sequence of length n. We use results from random walks and extreme value theory to argue that for two-state HMMs, the expected maximum memory is $O(m \log n)$. We give tight bounds for the symmetric case. We conduct experiments on a gene finding HMM and both real and simulated DNA sequences.

Symmetric two-state HMMs. Consider a *symmetric* two-state HMM over a binary alphabet as shown in Figure 3a. For simplicity, we assume $t < 1/2$ and $e < 1/2$. The back pointers between the sequence positions i and $i + 1$ can form configurations i–iii shown in Figure 3b. Denote $p_A = \log P(i, A)$ and $p_B = \log P(i, B)$, where $P(i, j)$ are probabilities computed in Viterbi algorithm. The Viterbi algorithm recurrence implies that the configuration i occurs when $\log t - \log(1 - t) \leq p_A - p_B \leq \log(1 - t) - \log t$, configuration ii occurs when $p_A - p_B \geq \log(1 - t) - \log t$, and configuration iii occurs when $p_A - p_B \leq \log t - \log(1 - t)$. Configuration iv never occurs for $t < 1/2$.[3]

For the two-state HMM, a coalescence point occurs whenever one of the configurations ii or iii occur. Thus the space is proportional to the length of continuous sequence of configurations i, which we call a *run*. First, we analyze the length distribution of runs assuming that the input sequence is a sequence of uniform i.i.d. binary random variables. In such case, we represent the run by a symmetric random walk corresponding to a random variable $X = \frac{p_A - p_B}{\log(1-e) - \log e} - (\log t - \log(1 - t))$. The configuration i occurs whenever $X \in (0, K)$, where

[3] We can easily extend analysis to other values of e and t. If $t > 1/2$, only configurations ii, iii, and iv may occur. Similar analysis applies by considering two steps of the algorithm together. If $t = 1/2$, we only have configurations ii and iii and the memory is constant. Case $e > 1/2$ is equivalent to the case of $e < 1/2$ after relabeling the states. If $e = 1/2$, the algorithm requires linear memory because of symmetry.

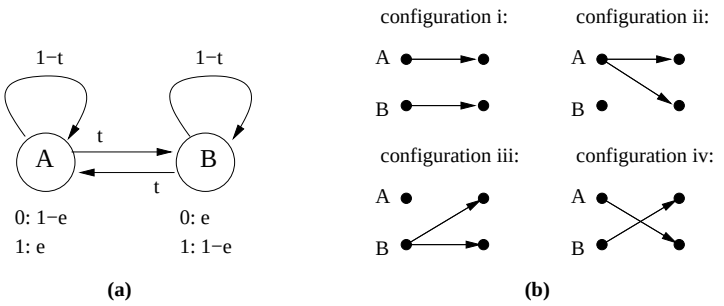

Fig. 3. (a) Symmetric two-state HMM with two parameters: e for emission probabilities and t for transitions probabilities. **(b) Possible back-pointer configurations** for a two-state HMM.

$K = \left\lceil 2\frac{\log(1-t)-\log(t)}{\log(1-e)-\log(e)} \right\rceil$. The quantity $p_A - p_B$ is updated by $\log(1-e) - \log e$, if the symbol at the corresponding sequence position is 0, or $\log e - \log(1-e)$, if this symbol is 1. This corresponds to updating X by $+1$ or -1.

When X reaches 0, we have a coalescence point in configuration iii, and the $p_A - p_B$ is initialized to $\log t - \log(1-t) \pm (\log e - \log 1 - e)$, which either means initialization of X to $+1$, or another coalescence point, depending on the symbol at the corresponding sequence position. The other case, when X reaches K and we have a coalescence point in configuration ii, is symmetric. We can now apply the classical results from the theory of random walks (see [14, ch.14.3,14.5]) to analyze the expected length of runs.

Lemma 1. *Assuming that the input sequence is uniformly i.i.d., the expected length of a run of a symmetric two-state HMM is* $K - 1$.

The larger is K, the more memory is required to decode the HMM; the worst case happens as e approaches $1/2$ and the states become indistinguishable. From the theory of random walks, we can characterize the distribution of run lengths.

Lemma 2. *Let* R_ℓ *be the event that the run length of a symmetric two-state HMM is either* $2\ell + 1$ *or* $2\ell + 2$. *Then, assuming that the input sequence is uniformly i.i.d., for some constants* $b, c > 0$*:*

$$b \cdot \cos^{2\ell} \frac{\pi}{K} \leq \Pr(R_\ell) \leq c \cdot \cos^{2\ell} \frac{\pi}{K} \tag{1}$$

Proof. For a symmetric random walk on interval $(0, K)$ with absorbing barriers and with starting point z, the probability of event $W_{z,n}$ that this random walk ends in point 0 after n steps is zero, if $n - z$ is odd, and the following quantity, if $n - z$ is even [14, ch.14.5]:

$$\Pr(W_{z,n}) = \frac{2}{K} \sum_{0<v<K/2} \cos^{n-1} \frac{\pi v}{K} \sin \frac{\pi v}{K} \sin \frac{\pi z v}{K} \tag{2}$$

Using symmetry, note that the probability of the same random walk ending after n steps at barrier K is the same as probability of $W_{K-z,n}$. Thus, if K is odd:

$$\begin{aligned}\Pr(R_\ell) &= \Pr(W_{1,2\ell+1}) + \Pr(W_{K-1,2\ell+1}) \\ &= \frac{2}{K} \sum_{0<v<K/2} \cos^{2\ell} \frac{\pi v}{K} \sin \frac{\pi v}{K} \left(\sin \frac{\pi v}{K} + (-1)^{v+1} \sin \frac{\pi v}{K} \right) \\ &= \frac{4}{K} \sum_{0<v<K/2,\ v \text{ odd}} \cos^{2\ell} \frac{\pi v}{K} \sin^2 \frac{\pi v}{K} \end{aligned} \tag{3}$$

There are at most $K/4$ terms in the sum and they can all be bounded from above by $\cos^{2\ell} \frac{\pi v}{K}$. Thus, we can give both upper and lower bounds on $\Pr(R_\ell)$ using only the first term of the sum as follows:

$$\frac{4}{K} \sin^2 \frac{\pi}{K} \cos^{2\ell} \frac{\pi}{K} \le \Pr(R_\ell) \le \cos^{2\ell} \frac{\pi}{K} \tag{4}$$

Similarly, if K is even, we have $\Pr(R_\ell) = \Pr(W_{1,2\ell+1}) + \Pr(W_{K-1,2\ell+2})$, obtaining a similar bound:

$$\frac{2}{K} \sin^2 \frac{\pi}{K} \left(1 + \cos \frac{\pi}{K}\right) \cos^{2\ell} \frac{\pi}{K} \le \Pr(R_\ell) \le 2 \cos^{2\ell} \frac{\pi}{K} \tag{5}$$

□

The previous lemma characterizes the length distribution of a single run. However, to analyze memory requirements for a sequence of length n, we need to consider maximum over several runs whose total length is n. Similar problem was studied for the runs of heads in a sequence of n coin tosses [15, 16]. There the length distribution of runs is geometric. In our case the runs are only bounded by geometrically decaying functions. Still, we can prove that the expected length of the longest run grows logarithmically with the length of the sequence, as is the case for the coin tosses.

Lemma 3. *Let $X_1, X_2, \ldots$ be a sequence of i.i.d. random variables drawn from a geometrically decaying distribution over positive integers, i.e. there exist constants a, b, c, $a \in (0,1)$, $0 < b \le c$, such that for all integers $k \ge 1$, $ba^k \le \Pr(X_i > k) \le ca^k$.*

Let N_n be the largest index such that $\sum_{i=1\ldots N_n} X_i \le n$, and let Y_n be $\max\{X_1, X_2, \ldots, X_{N_n}, n - \sum_{i=1}^{N_n} X_i\}$. Then

$$E[Y_n] = \log_{1/a} n + o(\log n) \tag{6}$$

Proof. Let $Z_n = \max_{i=1\ldots n} X_n$ be the maximum of the first n runs. Clearly, $\Pr(Z_n \le k) = \Pr(X_i \le k)^n$, and therefore $(1 - ca^k)^n \le \Pr(Z_n \le k) \le (1 - ba^k)^n$ for all integers $k \ge \log_{1/a}(c)$.

Lower bound: Let $t_n = \log_{1/a} n - \sqrt{\ln n}$. If $Y_n \leq t_n$, we need at least n/t_n runs to reach the sum n, i.e. $N_n \geq n/t_n - 1$ (discounting the last incomplete run). Therefore

$$\Pr(Y_n \leq t_n) \leq \Pr(Z_{\frac{n}{t_n}-1} \leq t_n) \leq (1 - ba^{t_n})^{\frac{n}{t_n}-1} = (1 - ba^{t_n})^{a^{-t_n} a^{t_n} (\frac{n}{t_n}-1)} \quad (7)$$

Since $\lim_{n\to\infty} a^{t_n}(n/t_n - 1) = \infty$ and $\lim_{x\to 0}(1 - bx)^{1/x} = e^{-b}$, we get $\lim_{n\to\infty} \Pr(Y_n \leq t_n) = 0$. Note that $E[Y_n] \geq t_n(1 - \Pr(Y_n \leq t_n))$, and thus we get the desired bound.

Upper bound: Clearly, $Y_n \leq Z_n$ and so $E[Y_n] \leq E[Z_n]$. Let Z'_n be the maximum of n i.i.d. geometric random variables $X'_1, \ldots, X'_n$ such that $\Pr(X'_i \leq k) = 1 - a^k$.

We will compare $E[Z_n]$ to the expected value of variable Z'_n. Without loss of generality, $c \geq 1$. For any real $x \geq \log_{1/a}(c) + 1$ we have:

$$\begin{aligned}
\Pr(Z_n \leq x) &\geq (1 - ca^{\lfloor x \rfloor})^n \\
&= \left(1 - a^{\lfloor x \rfloor - \log_{1/a}(c)}\right)^n \\
&\geq \left(1 - a^{\lfloor x - \log_{1/a}(c) - 1 \rfloor}\right)^n \\
&= \Pr(Z'_n \leq x - \log_{1/a}(c) - 1) \\
&= \Pr(Z'_n + \log_{1/a}(c) + 1 \leq x)
\end{aligned}$$

This inequality holds even for $x < \log_{1/a}(c) + 1$, since the right-hand side is zero in such case. Therefore, $E[Z_n] \leq E[Z'_n + \log_{1/a}(c) + 1] = E[Z'_n] + O(1)$. Expected value of Z'_n is $\log_{1/a}(n) + o(\log n)$ [17], which proves our claim. □

Using results of Lemma 3 together with the characterization of run length distributions by Lemma 2, we can conclude that for symmetric two-state HMMs, the expected maximum memory required to process a uniform i.i.d. input sequence of length n is $(1/\ln(1/\cos(\pi/K))) \cdot \ln n + o(\log n)$.[4] Using the Taylor expansion of the constant term as K grows to infinity, $1/\ln(1/\cos(\pi/K))) = 2K^2/\pi^2 + O(1)$, we obtain that the maximum memory grows approximately as $(2K^2/\pi^2) \ln n$.

The asymptotic bound $\Theta(\log n)$ can be easily extended to the sequences that are generated by the symmetric HMM, instead of uniform i.i.d. The underlying process can be described as a random walk with approximately $2K$ states on two $(0, K)$ lines, each line corresponding to sequence symbols generated by one of the two states. The distribution of run lengths still decays geometrically as required by Lemma 3; the base of the exponent is the largest eigenvalue of the transition matrix with absorbing states omitted (see e.g. [18, Claim 2]).

[4] We omitted the first run, which has a different starting point and thus does not follow the distribution outlined in Lemma 2. However, the expected length of this run does not depend on n and thus contributes only a lower-order term. We also omitted the runs of length one that start outside the interval $(0, K)$; these runs again contribute only to lower order terms of the lower bound.

Theorem 1. *The expected maximum memory used by the on-line Viterbi algorithm on a symmetric two-state HMM with* $t, e < 1/2$ *is* $\Theta(m \log n)$*, assuming sequences generated by either uniform i.i.d. or the HMM itself. Moreover, for the uniform i.i.d. sequences, the constant factor is approximately* $2K^2/\pi^2$*, where* $K = \left\lceil 2\frac{\log(1-t)-\log(t)}{\log(1-e)-\log(e)} \right\rceil$.

General two-state HMMs. We can extend the result of Theorem 1 to general two-state HMMs with states A and B, assuming i.i.d. generated sequence (not necessarily binary or uniform). If $t_B(A)t_A(B) < t_A(A)t_B(B)$, only configurations i, ii and iii may occur and we can again analyze the length of a single run as random walk of the variable $\log P(i, A) - \log P(i, B)$. Here, the random walk proceeds in steps that are arbitrary real numbers, different in each direction.

The step size for symbol x is $|\log(t_A(A)e_A(x)) - \log(t_B(B)e_B(x))|$, and we assume the existence of a symbol x with non-zero step size.[5] Sufficiently large number h of consecutive occurrences of x will always take the random walk to one of the absorbing barriers (configurations ii or iii), regardless of the starting point. Thus the length of a single run is bounded from above by the length of the sequence before h consecutive occurrences of x, length distribution of which clearly decays geometrically as required by Lemma 3. Thus we obtain an $O(m \log n)$ upper bound on the expected maximum memory for i.i.d. sequences.

As long as the emission probability of x is non-zero in all states of the HMM, we can straightforwardly extend this analysis to the sequences generated by the HMM. This condition ensures that appearance of h consecutive symbols of x is possible regardless of the starting state. As before, we can also extend the analysis to the case when $t_B(A)t_A(B) > t_A(A)t_B(B)$ by considering two steps at a time. Consequently the $O(m \log n)$ upper bound applies to all two-state HMMs, excluding some boundary symmetric cases and cases with zero emission or transition probabilities.

Multi-state HMMs. Our analysis technique cannot be easily extended to HMMs with many states. In two-state HMMs, each new coalescence event clears the memory, and thus the execution of the algorithm can be divided into independent runs. A coalescence event in a multi-state HMM may leave a tree of substantial depth in the memory. Thus, the sizes of consecutive runs are no longer independent (see Figure 4a).

To evaluate the memory requirements of our algorithm for multi-state HMMs, we have implemented the algorithm and performed several experiments on both simulated and biological sequences. First, we generalized the two-states symmetric HMMs to multiple states. The symmetric HMM with m states emits symbols over m-letter alphabet, where each state emits one symbol with higher probability than the other symbols. The transition probabilities are equiprobable, except

[5] Note that in the boundary case where all symbols have zero step size, it is again impossible to distinguish between individual states, and in the absence of tie breakers, the algorithm will require linear memory.

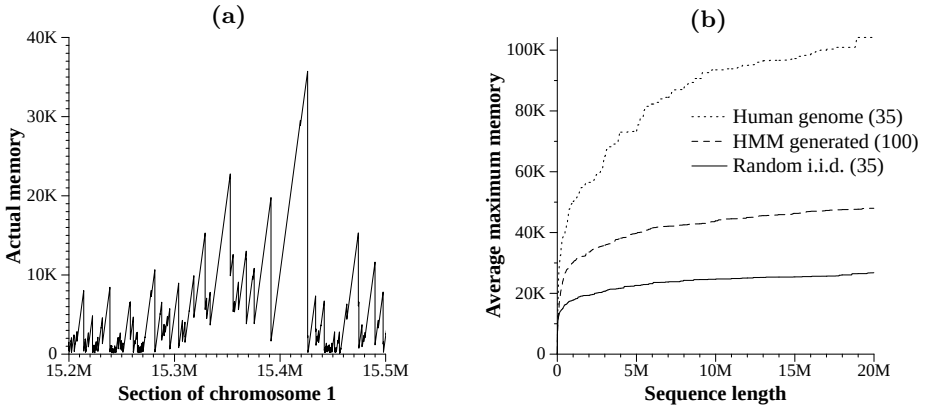

Fig. 4. Memory requirements of a gene finding HMM. a) Actual length of table used on a segment of human chromosome 1. b) Average maximum table length needed for prefixes of 20 MB sequences.

for self-transitions. We have tested the algorithm for $m \leq 6$ and sequences generated both by a uniform i.i.d. process, and by the HMM itself. Observed data are consistent with the logarithmic growth of average maximum memory needed to decode a sequence of length n (data not shown).

We have also evaluated the algorithm using a simplified HMM for gene finding with 265 states. The emission probabilities of the states are defined using at most 4-th order Markov chains, and the structure of the HMM reflects known properties of genes (similar to the structure shown in [19]). The HMM was trained on RefSeq annotations of human chromosomes 1 and 22.

In gene finding, we segment the input DNA sequence into exons (protein-coding sequence intervals), introns (non-coding sequence separating exons within a gene), and intergenic regions (sequence separating genes). Common measure of accuracy is exon sensitivity (how many of real exons we have successfully and exactly predicted). The implementation used here has exon sensitivity 37% on the ENCODE testing set [20]. A realistic gene finder, such as ExonHunter [21] achieves sensitivity of 53%. This difference is due to additional features that are not implemented in our test, namely GC content levels, non-geometric length distributions, and sophisticated signal models.

We have tested the algorithm on 20 MB long sequences: regions from the human genome, simulated sequences generated by the HMM, and i.i.d. sequences (see Figure 4b). Regions of the human genome were chosen from hg18 assembly so that they do not contain sequencing gaps. The distribution for the i.i.d. sequences mirrors the distribution of bases in the human chromosome 1. The average maximum length of the table over several samples appears to grow faster than logarithmically with the length of the sequence, though it seems to be bounded by $c \log^2 n$. It is not clear whether the faster growth is an artifact that would disappear with longer sequences or higher number of samples.

The HMM for gene finding has a special structure, with three copies of the state for introns that have the same emission probabilities and the same self-transition probability. In two-state symmetric HMMs, similar emission probabilities of the two states lead to increase in the length of individual runs. Intron states of a gene finder are an extreme example of this phenomenon.

Nonetheless, on average a table of length roughly 100,000 is sufficient to process sequences of length 20 MB, which is a 200-fold improvement compared to the trivial Viterbi algorithm. In addition, the length of the table did not exceed 222,000 on any of the 20MB human segments. As we can see in Figure 4a, most of the time the program keeps only relatively short table; the average length on the human segments is 11,000. The low average length can be of a significant advantage if multiple processes share the same memory.

4 Conclusion

In this paper, we introduced the on-line Viterbi algorithm. Our algorithm is based on efficient detection of coalescence points in trees representing the state-paths under consideration of the dynamic programming algorithm. The algorithm requires variable space that depends on the HMM and on the local properties of the analyzed sequence. Memory savings enable analysis of long biological sequences in gene finding, promoter detection, and detection of conserved elements. Previous approaches often artificially split the sequence into smaller pieces which can negatively influence the prediction accuracy.

For two-state symmetric HMMs, we have shown that the expected maximum memory needed for sequence of length n is approximately only $(2K^2/\pi^2)\ln n$. Our experiments on both simulated and real data suggest that the asymptotic bound $\Theta(m \log n)$ also extend to multi-state HMMs, and in fact, for most of the time throughout the execution the algorithm uses much less memory.

Our algorithm can also be used for on-line processing of streamed sequences; all previous algorithms that are guaranteed to produce the optimal state path require the whole sequence to be read before the output can be started.

There are still many open problems. We have only been able to analyze the algorithm for two-state HMMs, though trends predicted by our analysis seem to generalize even to more complex cases. Can our analysis be extended to multi-state HMMs? Apparently, design of the HMM affects the memory needed for the decoding algorithm; for example, presence of states with similar emission probabilities tends to increase memory requirements. Is it possible to characterize HMMs that require large amounts of memory to decode? Can we characterize the states that are likely to serve as coalescence points?

Acknowledgments: We thank Richard Durrett for useful discussions. TV is supported by NSF grant DBI-0644111, and NSF/NIGMS grant DMS-0201037. BB is supported by NIH/NCI (subcontract 22XS013A). Recently, we discovered parallel work on this problem by another research group [22]. Focus of their work is on implementation of a similar algorithm in their gene finder, and possible applications to parallelization; we focus on the expected space analysis.

References

1. Burge, C., Karlin, S.: Prediction of complete gene structures in human genomic DNA. Journal of Molecular Biology **268**(1) (1997) 78–94
2. Krogh, A., Larsson, B., von Heijne, G., Sonnhammer, E.L.: Predicting transmembrane protein topology with a hidden Markov model: application to complete genomes. Journal of Molecular Biology **305**(3) (2001) 567–570
3. Ohler, U., Niemann, H., Rubin, G.M.: Joint modeling of DNA sequence and physical properties to improve eukaryotic promoter recognition. Bioinformatics **17**(S1) (2001) S199–206
4. Durbin, R., Eddy, S., Krogh, A., Mitchison, G.: Biological sequence analysis: Probabilistic models of proteins and nucleic acids. Cambridge University Press (1998)
5. Pedersen, J.S., Hein, J.: Gene finding with a hidden Markov model of genome structure and evolution. Bioinformatics **19**(2) (2003) 219–227
6. Siepel, A., et al.: Evolutionarily conserved elements in vertebrate, insect, worm, and yeast genomes. Genome Research **15**(8) (2005) 1034–1040
7. Forney Jr., G.D.: The Viterbi algorithm. Proceedings of the IEEE **61**(3) (1973) 268–278
8. Grice, J.A., Hughey, R., Speck, D.: Reduced space sequence alignment. Computer Applications in the Biosciences **13**(1) (1997) 45–53
9. Tarnas, C., Hughey, R.: Reduced space hidden Markov model training. Bioinformatics **14**(5) (1998) 401–406
10. Wheeler, R., Hughey, R.: Optimizing reduced-space sequence analysis. Bioinformatics **16**(12) (2000) 1082–1090
11. Henderson, J., Salzberg, S., Fasman, K.H.: Finding genes in DNA with a hidden Markov model. Journal of Computational Biology **4**(2) (1997) 127–131
12. Hemmati, F., Costello, D., J.: Truncation error probability in Viterbi decoding. IEEE Transactions on Communications **25**(5) (1977) 530–532
13. Onyszchuk, I.: Truncation length for Viterbi decoding. IEEE Transactions on Communications **39**(7) (1991) 1023–1026
14. Feller, W.: An Introduction to Probability Theory and Its Applications, Third Edition, Volume 1. Wiley (1968)
15. Guibas, L.J., Odlyzko, A.M.: Long repetitive patterns in random sequences. Probability Theory and Related Fields **53** (1980) 241–262
16. Gordon, L., Schilling, M.F., Waterman, M.S.: An extreme value theory for long head runs. Probability Theory and Related Fields **72** (1986) 279–287
17. Schuster, E.F.: On overwhelming numerical evidence in the settling of Kinney's waiting-time conjecture. SIAM Journal on Scientific and Statistical Computing **6**(4) (1985) 977–982
18. Buhler, J., Keich, U., Sun, Y.: Designing seeds for similarity search in genomic DNA. Journal of Computer and System Sciences **70**(3) (2005) 342–363
19. Brejova, B., Brown, D.G., Vinar, T.: Advances in hidden Markov models for sequence annotation. In Mandoiu, I., Zelikovski, A., eds.: Bioinformatics Algorithms: Techniques and Applications. Wiley (2007) To appear.
20. Guigo, R., et al.: EGASP: the human ENCODE Genome Annotation Assessment Project. Genome Biology **7**(S1) (2006) 1–31
21. Brejova, B., Brown, D.G., Li, M., Vinar, T.: ExonHunter: a comprehensive approach to gene finding. Bioinformatics **21**(S1) (2005) i57–65
22. Keibler, E., Brent, M.: Personal communication (2006)