

# An Efficient Self-Tuning Explicit and Adaptive HMM with Duration Algorithm

S. Winters-Hilt\* and Z. Jiang\*\*

Department of Computer Science

University of New Orleans

2000 Lakeshore Drive

New Orleans, LA 70148

\* corresponding author: SWH: email: winters@cs.uno.edu;

office: 504-280-2407; fax: 504-280-7228

\*\* ZJ is an equally contributing author

## Abstract

We describe an efficient, self-tuning, explicit and adaptive, hidden Markov model with Duration (the ESTEAHMMD algorithm). The standard hidden Markov model (HMM) constrains state occupancy durations to be geometrically distributed, while the standard hidden Markov model with duration (HMMD) addresses this limitation, but at significant computational expense. A standard HMM requires computation of order  $O(TNN)$ , where  $T$  is the period of observations and  $N$  is the number of states. An explicit-duration HMM (HMMD) requires computation of order  $O(TNN + TND^2)$ , where  $D$  is the maximum interval between state transitions, while a hidden semi-Markov HMMD requires computation of order  $O(TNN + TND)$ . The latter improvement is still fundamentally limited if  $D \gg N$  (where  $D > 500$ , typically), and imposes a maximum state interval constraint that may be too restrictive in some situations such as intron modeling in gene structure identification. The ESTEAHMMD algorithm proposed here relaxes the maximum state interval constraint and requires computation of order  $O(TNN + TND^*)$ , where  $D^*$  is the bin number in an adaptive representation of the distribution on the interval between state transitions, and is typically reducible to  $\sim 50$  for standard single-peak probability distributions. This provides a means to do forward-backward and Viterbi algorithm HMMD computations at an expense only marginally greater than the standard HMM for  $N < 50$ ; and at negligible added expense when  $N > 50$ .

## I. INTRODUCTION

In this paper we first describe an explicit hidden Markov model with Duration (HMMD) with order of computation  $O(TNN + TND)$ , where  $T$  is the period of observations,  $N$  is the number of states, and  $D$  is the maximum interval between state transitions ( $D$  is typically  $> 500$ ). We then show how adaptive self-tuning can be used to further reduce the order of computation to  $O(TNN + TND^*)$ , where  $D^*$  is typically  $\sim 50$ . The adaptive reduction in computational expense is accomplished at no appreciable loss in accuracy over the explicit (exact) HMMD, and also provides a generalization to arbitrarily large intervals of state self-transitions (where  $D_{max} \gg D$ ). This is an important result because the critically important, HMM-based, Viterbi and Baum-Welch algorithms [1], with computational expense  $O(TNN)$ , are directly enhanced in their practical usage. The Viterbi and Baum-Welch algorithms are the underlying communication, error-coding, and structure-identification algorithms used in cell-phone communications, deep-space satellite communications, voice recognition, and in gene-structure identification [1,2], with growing applications in areas such as image processing now becoming commonplace as well. The HMMD generalization is important because the standard, HMM-based, Viterbi and Baum-Welch algorithms are critically constrained in their modeling ability to distributions on state intervals that are geometric. This works fine for the special instance where the state-interval distributions are geometric, but can lead to a significant decoding failure in noisy environments when the state-interval distributions are not geometric (or approximately geometric). The HMM with duration eliminates this deficiency by also exactly modeling the interval distributions themselves. The original description of an explicit HMMD required computation of order  $O(TNN + TND^2)$  [3], which was prohibitively computationally expensive in practical, real-time, operations, and introduced a severe maximum-interval constraint on the interval-distribution model. Improvements via hidden semi-Markov models to computations of order  $O(TNN + TND)$  were described in [4,5], where the maximum-interval constraint is still employed, and comparisons of these methods were subsequently detailed in [6].

The intuition guiding the result obtained here is that the standard HMM already does the desired duration modeling when the distribution modeled is geometric, suggesting that, with sufficient effort, a self-tuning explicit HMMD might be possible to achieve this efficiency in an adaptive context. We begin the description of our method with a description of the HMM and HMMD algorithms [1] in Sections I.A and I.B, followed by a full description of the hidden semi-Markov model formulations in Section II,

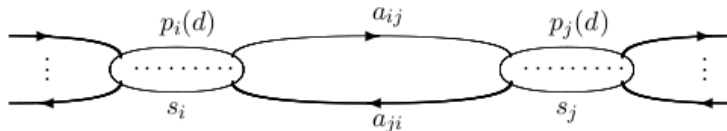
followed by the description of our adaptive solution in Section III. We present experimental results to further validate our Section III formalism in Section IV, and Conclusions are discussed in Section V.

### A. HMM background

In the standard HMM, when a state  $i$  is entered, that state is occupied for a period of time, via self-transitions, until transiting to another state  $j$ . If the state interval is given as  $d$ , the standard HMM description of the probability distribution on state intervals is implicitly given by:

$$p_i(d) = a_{ii}^{d-1}(1 - a_{ii}) \quad (1)$$

where  $a_{ii}$  is the self-transition probability of state  $i$ . This geometric distribution is inappropriate in many cases. An HMMD replaces Eq. (1) with a  $p_i(d)$  that models the real duration distribution of state  $i$ . In this way explicit knowledge about the duration of states is incorporated into the HMM. A general HMMD can be illustrated as:



When entered, state  $i$  will have a duration of  $d$  according to its duration density  $p_i(d)$ , it then transits to another state  $j$  according to the state transition probability  $a_{ij}$  (self-transitions,  $a_{ii}$ , are not permitted in this formalism). It is easy to see that the HMMD will turn into a HMM if  $p_i(d)$  is set to the geometric distribution shown in Eq. (1). The first HMMD formulation was studied by Ferguson [3]. A detailed HMMD description was later given by Rabiner [1] (we follow much of the Rabiner notation in what follows). There have been many efforts to improve the computational efficiency of the HMMD formulation given its fundamental utility in many endeavors in science and engineering. Notable amongst these are the variable transition HMM methods for implementing the Viterbi algorithm introduced in [4], and the hidden semi-Markov model implementations of the forward-backward algorithm [5].

### B. The standard explicit-duration HMM

In this section we recapitulate the exact HMMD formalism in the notation introduced by Rabiner [1].

Elements ( $\lambda$ ) of HMMD:  $N$ , the number of states;  $M$ , the number of distinct observations (where an observation sequence is denoted as:  $O = O_1O_2...O_T$ );  $D$ , the maximum duration length;  $a_{ij}$ , the state transition probability;  $e_i(k)$ , the emission probability: probability of observing  $k$  in state  $i$ ;  $\pi_i$ , the initial

state probability: the probability of state  $i$  given the observation sequence  $O$ ;  $p_i(d)$ , the state duration density: the probability of having exactly  $d$  consecutive state  $i$  observations after state  $i$  is entered.

The forward-backward variables:

- $f_t(i) = P(O_1 O_2 \cdots O_t, S_i \text{ ends at } t | \lambda)$
- $b_t(i) = P(O_{t+1} \cdots O_T | S_i \text{ ends at } t, \lambda)$
- $f_t^*(i) = P(O_1 O_2 \cdots O_t, S_i \text{ begins at } t+1 | \lambda)$
- $b_t^*(i) = P(O_{t+1} \cdots O_T | S_i \text{ begins at } t+1, \lambda)$

where  $f_t(i)$  can be calculated by:

$$f_t(i) = \sum_{j=1}^N \sum_{d=1}^D f_{t-d}(j) a_{ji} p_i(d) \prod_{s=t-d+1}^t e_i(O_s)$$

Others can be calculated similarly. The relationships among  $f$ ,  $f^*$ ,  $b$  and  $b^*$  are:

$$f_t^*(i) = \sum_{j=1}^N f_t(j) a_{ji}$$

$$b_t^*(i) = \sum_{d=1}^D b_{t+d}(i) p_i(d) \prod_{s=t+1}^{t+d} e_i(O_s)$$

$$f_t(i) = \sum_{d=1}^D f_{t-d}^*(i) p_i(d) \prod_{s=t-d+1}^t e_i(O_s)$$

$$b_t(i) = \sum_{j=1}^N a_{ij} b_t^*(j)$$

Based on the above definitions and equations, Rabiner provided the following maximum likelihood re-estimation formulas (the Baum-Welch algorithm) for HMMD:

$$\pi_i^{new} = \frac{\pi_i b_0^*(i)}{P(O|\lambda)} \quad (2)$$

$$a_{ij}^{new} = \frac{\sum_{t=1}^T f_t(i) a_{ij} b_t^*(j)}{N \sum_{j=1}^N \sum_{t=1}^T f_t(i) a_{ij} b_t^*(j)} \quad (3)$$

$$e_i^{new}(k) = \frac{\sum_{\substack{t=1 \\ \text{s.t. } O_t=k}}^T \left\{ \sum_{r<t} f_r^*(i) b_r^*(i) - \sum_{r<t} f_r(i) b_r(i) \right\}}{\sum_{k=1}^M \sum_{\substack{t=1 \\ \text{s.t. } O_t=k}}^T \left\{ \sum_{r<t} f_r^*(i) b_r^*(i) - \sum_{r<t} f_r(i) b_r(i) \right\}} \quad (4)$$

$$p_i^{new}(d) = \frac{\sum_{t=1}^T f_t^*(i) p_i(d) b_{t+d}(i) \prod_{s=t+1}^{t+d} e_i(O_s)}{\sum_{d=1}^D \sum_{t=1}^T f_t^*(i) p_i(d) b_{t+d}(i) \prod_{s=t+1}^{t+d} e_i(O_s)} \quad (5)$$

In the above equations we can see that  $\frac{D^2}{2}$  times more computational cost is required than the standard HMM. We now introduce a more efficient implementation of the explicit duration HMM that uses hidden semi-Markov models.

## II. THE HIDDEN SEMI-MARKOV MODEL HMM

In this section we introduce our hidden semi-Markov model (HSMM) formalism and continue with the notation introduced by Rabiner [1]. An equivalent formulation of the HSMM was introduced in [4] for the Viterbi algorithm and in [5] for Baum-Welch. The formalism introduced here, however, is directly amenable to the adaptive speedup method that will be described in the section that follows, so is presented in full.

We begin in Section II.A that follows with a description of the Baum-Welch algorithm in the hidden semi-Markov model (HSMM) formalism. This is followed in Subsection II.B with a description of the Viterbi algorithm in the HSMM formalism.

A. The Baum-Welch algorithm in the explicit HMM formalism

Define:

$$f_t(i, d) = \begin{cases} e_i(O_t) \sum_{j=1, j \neq i}^N F_{t-1}(j) a_{ji} & \text{if } d = 1 \\ f_{t-1}(i, d-1) e_i(O_t) & \text{if } 2 \leq d \leq D \end{cases}$$

$$= \begin{cases} F'_{t-1}(i) e_i(O_t) & \text{if } d = 1 \\ f_{t-1}(i, d-1) e_i(O_t) & \text{if } 2 \leq d \leq D \end{cases} \quad (6)$$

where

$$F_t(i) = \sum_{d=1}^D f_t(i, d) p_i(d) \quad (7)$$

$$F'_t(i) = \sum_{j=1, j \neq i}^N F_t(j) a_{ji} \quad (8)$$

It is easy to see the following relation:

$$f_t(i, d) p_i(d) = P(O_1 O_2 \cdots O_t, S_i \text{ ends at } t \text{ with duration of } d | \lambda)$$

Similarly define:

$$b_t(i, d) = \begin{cases} e_i(O_t) \sum_{j=1, j \neq i}^N a_{ij} B_{t+1}(j) & \text{if } d = 1 \\ e_i(O_t) b_{t+1}(i, d-1) & \text{if } 2 \leq d \leq D \end{cases}$$

$$= \begin{cases} e_i(O_t) B'_{t+1}(i) & \text{if } d = 1 \\ e_i(O_t) b_{t+1}(i, d-1) & \text{if } 2 \leq d \leq D \end{cases} \quad (9)$$

where

$$B_t(i) = \sum_{d=1}^D b_t(i, d) p_i(d) \quad (10)$$

$$B'_t(i) = \sum_{j=1, j \neq i}^N a_{ij} B_t(j) \quad (11)$$

and we have the following relation:

$$b_t(i, d) p_i(d) = P(O_1 O_2 \cdots O_t | S_i \text{ has a remaining duration of } d \text{ at } t, \lambda)$$

Now,  $f, f^*, b$  and  $b^*$  in Section I can be expressed as:

$$f_t^*(i) = F_t'(i)$$

$$b_t^*(i) = B_{t+1}(i)$$

$$b_t(i) = B_{t+1}'(i)$$

$$f_t(i) = F_t(i)$$

For convenience in what follows we now define:

$$\begin{aligned} \omega(t, i, d) &= P(O_1 \dots O_T, q_{t-1} \neq S_i, q_t \dots q_{t+d-1} = S_i, q_{t+d} \neq S_i | \lambda) \\ &= F_{t-1}'(i) b_t(i, d) p(d) \end{aligned} \quad (12)$$

$$\begin{aligned} \mu_t(i, j) &= P(O_1 \dots O_T, q_t = S_i, q_{t+1} = S_j | \lambda) \\ &= F_t(i) a_{ij} B_{t+1}(j) \end{aligned} \quad (13)$$

$$\varphi(i, j) = \sum_{t=1}^{T-1} \mu_t(i, j) \quad (14)$$

$$\nu_t(i) = P(O_1 \dots O_T, q_t = S_i | \lambda)$$

In what follows use is made of the following relation (as in [5]):

$$\begin{aligned} &P(O_1 \dots O_T, q_t = S_i, q_{t+1} = S_i | \lambda) \\ &= P(O_1 \dots O_T, q_t = S_i | \lambda) - P(O_1 \dots O_T, q_t = S_i, q_{t+1} \neq S_i | \lambda) \\ &= P(O_1 \dots O_T, q_{t+1} = S_i | \lambda) - P(O_1 \dots O_T, q_t \neq S_i, q_{t+1} = S_i | \lambda) \end{aligned}$$

from which we get the following recursion formula:

$$\nu_t(i) = \begin{cases} F_T(i) & \text{if } t = T \\ \nu_{t+1} + \sum_{j \neq i}^N (\mu_t(i, j) - \mu_t(j, i)) & \text{if } 1 \leq t \leq T - 1 \end{cases} \quad (15)$$

Using the above equations, Eq. (2) – (5) can be expressed as:

$$\pi_i^{new} = \frac{\pi_i B_1(i)}{P(O|\lambda)} \quad (16)$$

$$a_{ij}^{new} = \frac{\varphi(i, j)}{\sum_{j=1}^N \varphi(i, j)} \quad (17)$$

$$e_i^{new}(k) = \frac{\sum_{t=1}^T \nu_t(i)}{\sum_{t=1}^T \nu_t(i) \text{ s.t. } O_t=k} \quad (18)$$

$$p_i(d) = \frac{\sum_{t=1}^T \omega(t, i, d)}{D \sum_{d=1}^D \sum_{t=1}^T \omega(t, i, d)} \quad (19)$$

A summary of the Baum-Welch training algorithm is as follows:

- 1) initialize elements( $\lambda$ ) of HMMD.
- 2) calculate  $f_t(i, d)$  using Eq. (6) – (8).  
(save two tables:  $F_t(i)$  and  $F'_t(i)$ )
- 3) calculate  $b_t(i)$  using Eq. (9) – (11).
- 4) reestimate elements( $\lambda$ ) of HMMD using Eq. (12) – (19).
- 5) terminate if stop condition is satisfied, else goto step 2.

The memory complexity of this method is  $O(TN)$ . As shown above, the algorithm first does forward computing (step (2)), and saves two tables: one is  $F_t(i)$ , the other is  $F'_t(i)$ . Then at every time index  $t$ , the algorithm can group the computation in step (3) and (4) together. So no backward table needs to be saved. We can do a rough estimation of HMMD's computation cost by counting multiplications inside the loops of  $\sum^T \sum^N$  (which corresponds to standard HMM's computation cost.) and  $\sum^T \sum^D$  (the additional computational cost incurred by the HMMD). The computation complexity is  $O(TNN + TND)$ . In an actual implementation a scaling procedure may be needed to keep the forward-backward variables within a manageable numerical interval. One method is to rescale the emission parameter as:  $e_i(O_t) = e_i(O_t) / \sum_{i=1}^N F_{t-1}(i)$  at every time index. Another approach, used here, is dynamic scaling. First, we do a rescaling on  $e_i(k)$  using a big constant. Then at every time index we test if the numerical values become too small, if so, we use the scaled version of  $e_i(k)$  to push the numerical values up; if not, we keep using the unscaled version. In this way no additional computation complexity is introduced by

scaling.

### B. The Viterbi algorithm in the explicit HMMD formalism

Define:

$$v_t(i, d) = \begin{cases} e_i(O_t) \max_{j=1, j \neq i}^N \{V_{t-1}(j) a_{ji}\} & \text{if } d = 1 \\ v_{t-1}(i, d-1) e_i(O_t) & \text{if } 2 \leq d \leq D \end{cases}$$

where

$$V_t(i) = \max_{d=1}^D \{v_t(i, d) p_i(d)\}$$

for which we have a useful recursive definition:

$$v_t(i, d) p_i(d) = \text{the probability of the most probable path} \\ \text{that ends at state } i \text{ with duration of } d \text{ at time } t$$

The algorithm's goal is to find:

$$\operatorname{argmax}_{i, d} \{v_T(i, d) p_i(d)\}$$

Since a logarithm scaling can be applied on  $a_{ij}$ ,  $e_i(k)$  and  $p_i(d)$  in advance, the above equations become:

$$v_t(i, d) = \begin{cases} \log e_i(O_t) + \max_{j=1, j \neq i}^N \{V_{t-1}(j) + \log a_{ji}\} & \text{if } d = 1 \\ v_{t-1}(i, d-1) + \log e_i(O_t) & \text{if } 2 \leq d \leq D \end{cases} \quad (20)$$

where

$$V_t(i) = \max_{d=1}^D \{v_t(i, d) + \log p_i(d)\}$$

The goal simplifies to:

$$\operatorname{argmax}_{i, d} \{v_T(i, d) + \log p_i(d)\}$$

The complexity of HMMD's Viterbi is  $O(TNN + TND)$ . Because the logarithm scaling can be performed in advance, the Viterbi procedure consists only of additions to yield a very fast computation.

### III. THE ADAPTIVE HIDDEN SEMI-MARKOV MODEL HMM

The duration distribution of state  $i$  consists of rapidly changing probability regions (with small change in duration) and slowly changing probability regions. In the last section all regions share an equal computation resource (represented as  $D$  substates of a given state) — this can be very inefficient in practice. In this section, we describe a way to recover computational resources, during the training process, from the slowly changing probability regions. As a result, the computation complexity can be reduced to  $O(TNN + TND^*)$ , where  $D^*$  is the number of “bins” used to represent the final, coarse-grained, probability distribution. A “bin” of a state is a group of substates with consecutive duration. For example,  $f(i, d), f(i, d + 1), \dots, f(i, d + \Delta d)$  can be grouped into one bin. The bin size is a measure of the granularity of the evolving length distribution approximation. A fine-granularity is retained in the active regions, perhaps with only one length state per bin, while a coarse-granularity is adopted in weakly changing regions, with possibly hundreds of length states per bin. For the latter case, an immediate generalization to the exact HMMD introduced in Sect. III is suggested for handling long duration state intervals — a “tail bin”. Such a bin is strongly indicated for good modeling on certain important distributions, such as the long-tailed distributions often found in nature, such as the exon and intron interval distributions found in gene-structure modeling and prediction. In practice, the idea is to run the exact HMMD on a small portion,  $\Delta T$ , of the training data, at  $O(\Delta TNN + \Delta TND)$  cost, to get an initial estimate of the state interval distributions. Some preliminary course-graining is then performed, where strongly indicated, and the number of bins representing the length distribution is reduced from  $D$  to  $D'$ . The exact HMMD is then performed on the  $D'$  substate model for another small portion of the training data, at computational expense  $O(\Delta TNN + \Delta TND')$ . This is repeated until the number of bin states,  $D^*$ , reduces no further, and the bulk of the training then commences with the  $D^*$  bin-states length distribution model at expense  $O(TNN + TND^*)$ . The key to this process is the retention of training information during the freezing out of length distribution states, and such that the  $D^*$  bin state training process can be done at expense  $O(TNN + TND^*)$ . This process is what is shown next. Note: the “tail-bin” mentioned above can be used via the method described here to increase the maximum state interval to fantastically large values, such as 100,000 rather than the 500 cutoff typically employed (where the bin may start at interval length 500, but includes state lengths up to 100,000 — a prospect useful in intron modeling).

Starting from the above binning idea, for substates in the same bin, a reasonable approximation is

applied:

$$\sum_{d=d'}^{d'+\Delta d} f_t(i, d) p_i(d) = p_i(\bar{d}) \sum_{d=d'}^{d'+\Delta d} f_t(i, d)$$

where  $\bar{d}$  is the duration representative for all substates in that bin. For different bins, the values of  $\bar{d}$  or  $\Delta d$  may be different. So below whenever they are mentioned, we mean those values associated with its bins.

We begin in Section III.A that follows with a description of the Baum-Welch algorithm in the adaptive hidden semi-Markov model (HSMM) formalism. This is followed in Section III.B with a description of the Viterbi algorithm in the adaptive HSMM formalism.

#### A. The Baum-Welch algorithm in the adaptive HMM formalism

First define a variable associated with each bin (which can be calculated recursively):

$$E_t(i, n) = \prod_{t-\Delta d}^t e_i(O_t)$$

In an actual implementation the scaling procedure is applied on  $e_t(O_t)$ , so the underflow problem can be avoided. Based on the above approximation, the equations in formulas (6), (7) and (8) (used by the forward algorithm) can be replaced by:

$$fbin_t(i, n) = \begin{cases} \{fbin_{t-1}(i, n) - \rho_t(i, n) + F'_{t-1}(i)\} e_i(O_t) & \text{if } n = 1 \\ \{fbin_{t-1}(i, n) - \rho_t(i, n) + \rho_t(i, n-1)\} e_i(O_t) & \text{if } 1 < n \leq D^* \end{cases} \quad (21)$$

where

$$F_t(i) = \sum_{n=1}^{D^*} fbin_t(i, n) p_i(\bar{d}) \quad (22)$$

$$F'_t(i) = \sum_{j=1, j \neq i}^N F_t(j) a_{ji} \quad (23)$$

$$\rho_t(i, n) = queue(i, n).pop * E_{t-1}(i, n) \quad (24)$$

$$queue(i, n).push(\rho_t(i, n-1)) \quad (25)$$

The explanation for push and pop operations, etc., begins with associating every bin with a queue  $queue(i, n)$ . The queue's size is equal to the number of substates grouped by this bin. At every time index, the oldest substate:  $f(i, d + \Delta d)$  will be shifted out of its current bin and pushed into its next bin, as shown in (24) and (25), where  $queue(i, n)$  stores the original probability of each substate in that bin when they were pushed in. So when one substate becomes old enough to move to the next bin, its

current probability can be recovered by first popping out its original probability, then multiplied by its “gain”—  $E_{t-1}(i, n)$ , as shown in (24). On the other hand, as long as substates stay inside a bin, their individual computations can be reduced to only one in the *fbin* term, as shown in (21).

Similarly, formulas (9), (10) and (11) used by backward algorithm can be replaced by:

$$bbin_t(i, n) = \begin{cases} e_t(O_t) \{bbin_{t+1}(i, n) - \rho_t(i, n) + B'_{t+1}(i)\} & \text{if } n = 1 \\ e_t(O_t) \{bbin_{t+1}(i, n) - \rho_t(i, n) + \rho_t(i, n + 1)\} & \text{if } 1 < n \leq D^* \end{cases} \quad (26)$$

where

$$B_t(i) = \sum_{n=1}^{D^*} bbin_t(i, n) p_i(\bar{d}) \quad (27)$$

$$B'_t(i) = \sum_{j=1, j \neq i}^N a_{ij} B_t(j) \quad (28)$$

$$\rho_t(t, n) = queue(i, n).pop * E_{t+\Delta d+1}(i, n) \quad (29)$$

$$queue(i, n).push(\rho_t(i, n + 1)) \quad (30)$$

For re-estimation, formula (12) now becomes:

$$\omega(t, i, \bar{d}) = F'_{t-1}(i) bbin_t(i, \bar{d}) p(\bar{d}) \quad (31)$$

while the other re-estimating formulas remain unchanged.

### B. The Viterbi algorithm in the adaptive HMMD formalism

The adaptive process used here is similar to the one for adaptive Baum-Welch training (with computation complexity also  $O(TNN + TND^*)$ ), where the following formulas are used:

$$E_t(i) = \begin{cases} 0 & \text{if } t = 1 \\ E_{t-1}(i) + \log e_i(O_t) & \text{if } 1 < t \leq T \end{cases} \quad (32)$$

$$\zeta_t(i) = \max_{j=1, j \neq i}^N \{m_{t-1}(j) + E_{t-1}(j) + \log a_{ji}\} \quad (33)$$

$$\zeta_t(i) = \zeta_t(i) - E_{t-1}(i) \quad (34)$$

$$\text{if } (bin(i, n).getlast()) \text{ is too old} \{bin(i, n).removelast();\} \quad (35)$$

$$\text{while } (bin(i, n).getfirst() < \zeta_t(i)) \{bin(i, n).removefirst();\} \quad (36)$$

$$bin(i, n).add(\zeta_t(i)) \quad 1 \leq n \leq D^* \quad (37)$$

$$t' = t + d(i, n) - 1 \quad 1 \leq n \leq D^* \quad (38)$$

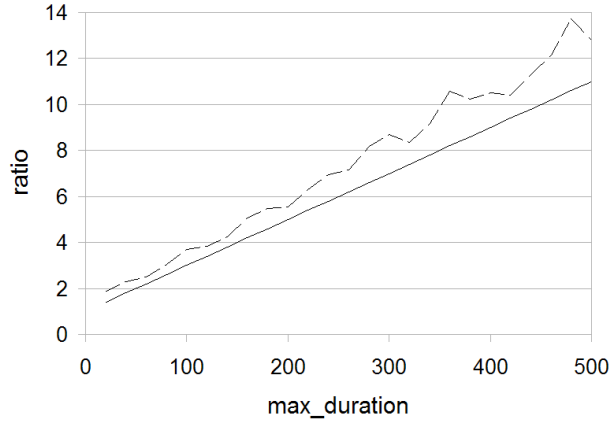
$$\text{if } (bin(i, n).getlast() + \log(p_i(\bar{d}))) < m_{t'}(i) \{m_{t'}(i) = bin(i, n).getlast() + \log(p_i(\bar{d}));\} \quad (39)$$

The explanation and usage for the above relations are as follows:

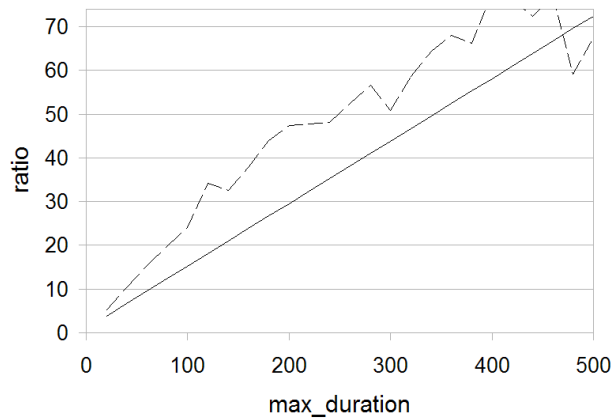
- First define the term  $E_t(i)$  as shown in (32). When a new substate transits (from one of other states) to state  $i$ , we store the difference between this substate’s current path probability at time  $t$  and the term  $E_{t-1}(i)$ , as shown in (34). Then if this substate’s probability is needed at any future time  $t' > t$ , its path probability at time  $t'$  can be computed as “the originally stored difference” plus  $E_{t'}(i)$ , as shown in the term  $m_{t-1}(j) + E_{t-1}(j)$  in (33). The benefit is that the add-computations on  $\log e_i(O_t)$  in (20) are saved (in (32)).
- If a substate at  $t - 1$  has a duration  $d - 1$ , then at time  $t$ , its duration increased to  $d$ . So at every time index we check to see if there is a substate too old to stay in the current bin, if so, it is removed from the current bin, as shown in (35).
- Before a new substate is pushed into the bin, we first scan the bin top-down and delete all substates whose path probability is less than the new substate. Because those deleted substates will never have a chance to be chosen as the “max” viterbi path, as shown in (36) and (37). As a result, paths inside a bin are always kept sorted, and the “max” one is always the last element of the bin. Thus, in (39) we can directly apply *getlast()* on each bin to get its “max”. The advantage of (36) is that the complexity of the “while” loop is only  $O(1)$ . (Suppose if more than one comparison is needed on average for every time index, then the bins will soon shrink to empty, a contradiction.)
- The max probability path of  $bin_t(i, n)$  can be pre-determined at time  $t - d(i, n) + 1$ , where  $d(i, n)$  is the duration length of first substate of  $bin_t(i, n)$ . This is because all substates of the same state  $i$  get the same increment of  $e_i(O_t)$  at very time index. Detailed steps are shown in (37), (38) and (39).

#### IV. EXPERIMENTAL RESULTS

In Figures 1 and 2 we present experimental results for the speed comparison between HMM and HSMM signal processing on  $N = 7$  and  $N = 50$  state models (reasonable cases encountered in gene-structure identification and channel current analysis, respectively). We expect an added expense of  $O[(D + N)/N]$  in such comparisons, and this is what is observed.



In Figure 1, the ratio of HMM and HMMD signal processing time is shown for  $N = 7$  state models as a function of the maximum state interval (maximum duration of self-transition). The dashed, irregular, line shows the experimental ratio that is observed, the straight line shows the corresponding theoretical value of  $(D + N)/N$ .



In Figure 2, the ratio of HMM and HMMD signal processing time is shown for  $N = 50$  state models as a function of the maximum state interval (maximum duration of self-transition). The dashed, irregular, line shows the experimental ratio that is observed, the straight line shows the corresponding theoretical value of  $(D + N)/N$ .

In Figure 3 we show state-decoding on synthetic data that is representative of a biological-channel two-state ion-current decoding problem. For this problem 120 data sequences were generated that have two states with channel blockade levels set at 30 and 40 pA (a typical scenario in practice). Every data sequence has 10,000 samples. Each state has emitted values in a range from 0 to 49 pA. The maximum duration of states is set at 500. The mean duration of the 40 pA state is given as 200 samples (typically have 1 sample every 20 microseconds in actual experiments), while the 30 pA level has mean duration set

at 300 samples. The task is to train using 100 of the generated data sequences and attempt state-decoding on the remaining 20 data sequences. Example sequences are shown in Fig. 3, along with their decoding when an HMM or an HMMD is employed. The performance difference is stark: the exact and adaptive HMMD decodings are 97.1% correct, while the HMM decoding is only correct 61% of the time (where random guessing would accomplish 50%, on average, in a two-state system). It is found that the HMMD outperforms the HMM even when the data is generated by a geometric distribution – this is because the theoretical geometric distribution curve is smooth, while the sample-based version correctly identifies step discontinuities. Three parameterized distributions were examined in this way: geometric, Gaussian, and Poisson. Distributions both segmented and “messy” were also examined. In all cases the HMMD performed robustly, similar to the above, and in all cases the adaptive HMMD optimization performed comparably to the more computationally expensive exact HMMD. In fact, in many tests an adaptive HMMD with very few states, often just  $D^* = 3$  or 4, is able to perform comparably to the exact HMMD (96.6% correct decoding for the adaptive HMMD compared to 96.8% correct decoding for the exact HMMD, on a comparative test with  $D^* = 3$ ). This remarkably improved performance compared to the HMM performance ( $\sim 61\%$  correct) is attributed to the benefit of having a better fit to the “tail” of the distribution (i.e., the tail bin described earlier).

A variety of self-tuning protocols could be adopted in the adaptive HMMD learning process. For simplicity in the test problem examined in Figure 3 we do the first 1/1000 of training on the fine-granularity  $D$ -states and thereby estimate the length distribution crudely, but enough to estimate some very slowly changing regions (where counts are sufficiently high for this to be trusted), we then conduct the next round of training on another 1/1000 of the training data, but now with slightly fewer states because a very small amount of granularity in states is introduced (where not all bins simply have a single length state), and make use of the relations described in Section IV. We then iterate this granularization process, such that by the time 1/10 of the data has been examined the granularity reduction is accomplished, and the remaining 90% of training is performed at the optimized (dynamic mesh) granularity. All sums, recursions, max operations, etc., shift to expressions according to the granularity. In this way we get the best of all possibilities, a  $D$ =huge number HMMwD that dynamically shifts to a strong representation of the length distribution information using  $D^*$  bins.

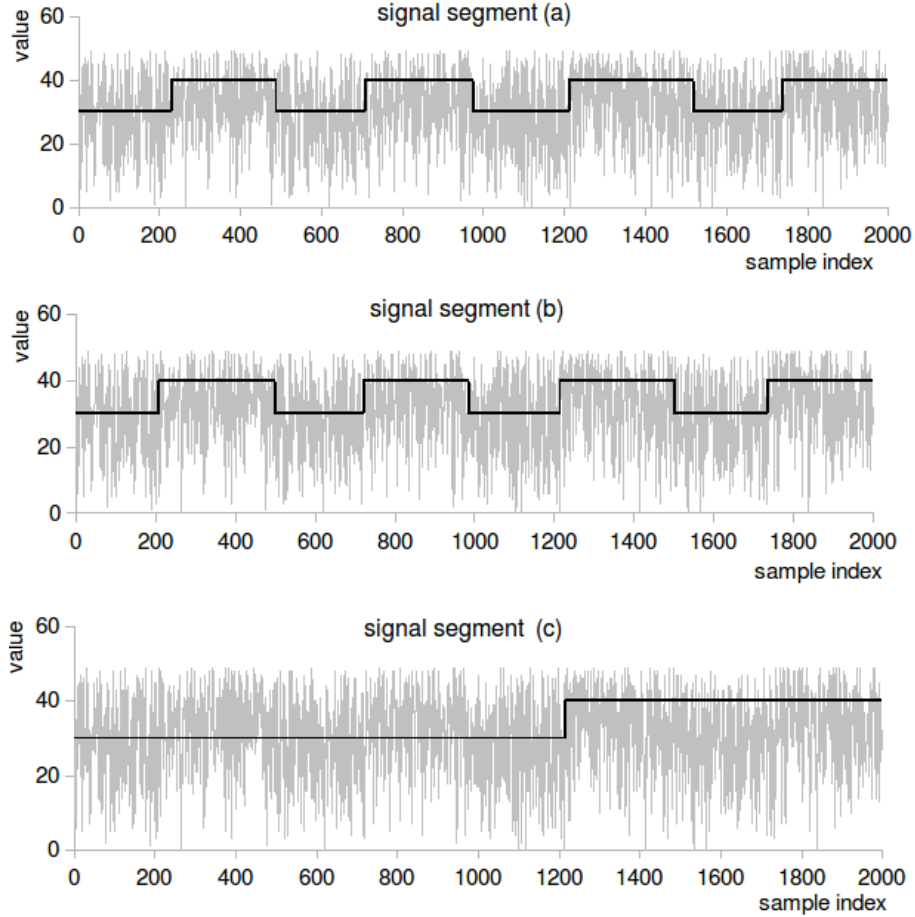


Fig.3. In the figure we show state-decoding results on synthetic data that is representative of a biological- channel two-state ion-current decoding problem. Signal segment (a) (at the top) shows the original two-level signal as the horizontal line, while the noised version of the signal is shown as the gray line. Signal segment (b) (in the middle) shows the noised signal in gray and the two-state denoised signal according to the HMMD decoding process (whether exact or adaptive), which is stable (97.1% accurate) allowing for state-lifetime extraction (with the concomitant chemical kinetics information that is thereby reliably obtained in this channel current analysis setting). Signal segment (c) (at the bottom) shows the standard HMM signal resolution, and its failure to properly resolve the desired level-lifetime information.

## V. CONCLUSION

In this paper we describe an explicit hidden Markov model with Duration (HMMD) with order of computation  $O(TNN + TND)$ , where  $T$  is the period of observations,  $N$  is the number of states,

and  $D$  is the maximum interval between state transitions ( $D$  is typically  $> 500$ ). We present an adaptive self-tuning explicit hidden Markov model with Duration approach that can be used to accomplish HMMD computations at comparable order to the standard HMM, where the order of computation is  $O(TNN + TND^*)$ , where  $D^*$  is typically  $\sim 50$ ,  $T$  is the period of observations, and  $N$  is the number of states. The adaptive reduction in computational expense is accomplished at no appreciable loss in accuracy over the explicit (exact) HMMD, and also provides a generalization to arbitrarily large intervals of state self-transitions (where  $D_{max} \gg D$ ). The latter generalization is of particular importance in a number of scientific endeavors where anomalously “long-tailed” distributions are found to exist – such as occurs for exons and introns (and sometimes extremely-so for introns). By means of the ESTEAHMMD algorithm we are able to reduce the computational expense of an explicit HMMD to just the  $O(TNN)$  cost of the standard HMM if  $N > 50$ ; as is often the case in power signal analysis and meta-state gene-structure identification applications.

#### ACKNOWLEDGEMENTS

Funding for this research was provided by an NIH K-22 grant (5K22LM008794, SWH PI).

#### REFERENCES

- [1] L.R. Rabiner, “A tutorial on hidden Markov models and selected application in speech recognition,” Proc. IEEE, vol. 77, pp. 257-286, Feb. 1989.
- [2] A. Krogh, S. Mian, D. Haussler. “A hidden Markov model that finds genes in E. coli DNA,” Nucleic Acids Research. 1994;22( 22):4768-78.
- [3] J.D. Ferguson, “Variable duration models for speech,” in Symp. Application of Hidden Markov models to Text and Speech, Oct. 1980, pp. 143-179.
- [4] P. Ramesh and J.G. Wilpon, “Modeling state durations in hidden Markov models for automatic speech recognition,” in Proc. Int. Conf. Acoust., Spech, Signal Process., 1992, pp. 381-384.
- [5] S.-Z. Yu and H. Kobayashi, “An efficient forward-backward algorithm for an explicit-duration hidden Markov model,” IEEE Signal Process. Lett., vol. 10, no. 1, pp.11-14, Jan. 2003.
- [6] M.T. Johnson, “Capacity and complexity of HMM duration Modeling Techniques,” IEEE Sig. Process. Lett, Vol 12, No. 5, pp. 407-410, May 2005.

Stephen Winters-Hilt. Ph.D. Computer Science, UCSC, Santa Cruz, CA, USA, 2003. Ph.D. Physics, U. Wisconsin, Milwaukee, WI, USA, 1997. M.S. Applied Physics, California Institute of Technology, CA, USA, 1990. B.S. Electrical Engineering & Physics, California Institute of Technology, CA, USA, 1987. He is currently an Assistant Professor with the Computer Science Department at the University of

New Orleans, New Orleans, LA, 70148, USA and is Principal Investigator at Childrens Hospital, New Orleans, LA, 70118, where he is Director of the Nanopore Biophysics Lab.

Zuliang Jiang. B.S. Computer Science, Xiamen University, China, 2006. He is currently a Ph.D. Candidate with the Computer Science Department at the University of New Orleans, New Orleans, LA, 70148, USA.