An Incremental Viterbi Algorithm

Jason Bobbin
Defence Science and Technology Organisation
Command Control Communications and Intelligence Division
Edinburgh, South Australia, Australia
jason.bobbin@dsto.defence.gov.au

Abstract

This paper describes an incremental version of the Viterbi dynamic programming algorithm. The incremental algorithm is shown to dramatically reduce memory usage in long state sequence problems compared with the standard Viterbi algorithm while having no measurable impact on the algorithms runtime. In addition, the set of problems which the Viterbi algorithm can be applied is extended by the incremental algorithm to include problems of finding optimal paths in realtime domains. The Viterbi algorithm is widely used to find optimal paths in hidden markov models (HMM), and HMMs are frequently applied to both streaming data problems where realtime solutions can be of interest, and to large state sequence problems in areas like bioinformatics. In this paper we apply the incremental algorithm to finding optimal paths in a variant of the burst detection HMM applied to the novel problem of detecting user activity levels in digital evidence data derived from hard drives.

1 Introduction

Hidden markov models (HMMs) provide a flexible probabilistic modelling paradigm with a wide range of applications. They are designed to solve linear sequence 'labeling' problems, where the model assigns a sequence of labels, or states, to a linear sequence of inputs [3]. A common problem in many applications is to find the optimal sequence of states, called the optimal path, that explains a given sequence of observations. Given the probability of a state transition in the model, and the probability of the hidden state generating an observation, we can use the Viterbi dynamic programming algorithm to find the most likely state sequence which the model can use to generate a particular sequence of observations [3, 12].

HMMs have recently been successfully applied to finding patterns in streams of data, such as document streams [9, 7, 2, 17], web browsing [1], network traffic identification [16] and route planning applications [10, 11]. The Viterbi algorithm allows for optimal paths of the HMMs to be found in $\mathcal{O}(TN^2)$ time in general, where N is the number of different states and T is the number of discrete time steps or state sequences. Recently it has been shown that many problems, including those considered here, can be solved by the Viterbi algorithm in $\mathcal{O}(TN)$ time [5, 4] with a small modification. In all cases finding the optimal path requires $\mathcal{O}(TN)$ memory, and no part of the path is available before the completion of all calculations.

Most streaming HMM models are applied to data which could be available in realtime. To apply these models to realtime data streams presents a number of problems:

- how to provide feedback on the optimal path before the final observation is received in an online situation?
- how should the HMM models be applied to input streams which may not have a natural final observation?
- how to find the optimal state sequence when the size of the observation space prevents the Viterbi algorithm from being implemented with reasonable hardware constraints?

This last problem exists in non-online problem domains as well, such as bioinformatic sequence modeling, and finding bursts in large data collections. In bioinformatic modeling, large input sequences are typically split into multiple shorter sequences which then have their respective optimal paths reassembled to produce a suboptimal path of the entire sequence in reasonable memory [14].

The high memory requirements for the Viterbi algorithm stem from the requirement that the entire path matrix consisting of all optimal paths for a given state at a given time needs to be kept for the entire run. However, for the long state sequence problems the algorithm is now being applied to the entire matrix nearly always contains redundant information which is not required in order to find the optimal path sequence. This occurs because not all, or even

0-7695-3069-9/07 $25.00 © 2007 IEEE
DOI 10.1109/ICMLA.2007.49

most, solution path elements remain reachable when we reconstruct the optimal path. This fact has been exploited to produce algorithms which increase run time complexity in order to reduce memory usage [8, 14]. The modification we describe in this paper does not measurably affect the runtime, while still reducing the memory usage. The algorithm works through a bookeeping approach to tracking the reachable states at the start of a particular reachability window. When the algorithm detects only one state as being reachable at the start of the window, the optimal path up to that position can be returned.

The memory requirements for the proposed scheme still have a worst case of $\mathcal{O}(TN)$, although in practice this would not occur except in pathological examples. A very recently published result from Šrámek et al. uses a more complicated scheme to achieve a similar reduction in memory usage by exploiting the same property of the path matrix [14, 13]. The approach taken by Šrámek et al. requires a modest increase in running times that our approach does not [14].

2 An Incremental Viterbi Solution

In this section we briefly outline the Viterbi algorithm and the typical HMM structure which it is used on before presenting our modification.

The Viterbi algorithm is a recursive optimisation procedure which can find optimal state sequences of discrete time finite-state Markov processes [15, 6]. The algorithm originated in the signal processing literature for decoding transmissions, and is presently well known as a method for finding optimal path sequences in HMMs. An HMM λ can be expressed as a 5-tuple $\lambda = (S, O, \pi, A, B)$ as follows:

Over some discrete time period $t = \{1, 2, 3, \ldots T\}$, given a discrete set of (non-observable) states $S = \{1, 2, 3, \ldots, N\}$, a discrete set of observations $O = \{o_t\}$, and an initial state probability π of being in state i of π_i, a transition matrix A where element a_{ij} gives the probability of the HMM transitioning from state i to state j and, where X_t denotes the state at time t, we have a set of emission probabilities B describing the probability of observing output o_t when in state j, i.e. $b_j(o_t) = P(o_t \mid X_t = j)$.

Given a HMM λ as above the question which the Viterbi algorithm answers is how to find an optimal state sequence $Q = (q_1, q_2, q_3, \ldots q_T),\ q_i \in S$ which maximizes the expectation of seeing the particular observation sequence O [12]. The probability of seeing O given our model λ is:

$$\begin{aligned} P(O|\lambda) &= \sum_{\text{All } Q} P(O|Q,\lambda)P(Q|\lambda) \\ &= \sum_{q_1,q_2,\ldots q_n} \pi_{q_1} b_{q_1}(o_1) a_{q_1 q_2} b_{q_2}(o_2) \\ &\quad \ldots a_{q_{n-1}q_n} b_{q_n}(o_n) \end{aligned} \tag{1}$$

which leads to the well known forward procedure for calculating $P(O|\lambda)$ via the forward variable:

$$\alpha(i) = P(o_1 o_2 \ldots o_t, q_t = i|\lambda)$$

and the recursion relation [12] :

$$\alpha_{t+1}(j) = \left(\sum_{i=1}^{N} \alpha_t(i) a_{i,j} \right) b_j(o_{t+1})$$

When finding the optimal state sequence Q for λ we can see from Equation 1 that we will need to balance the likelihood of the chosen state at time t (q_t producing observation o_t), and the probability of seeing a transition from the previous state q_{t-1} to the currently chosen state q_t. These two criteria depend on the emission probability $b_{q_t}(o_t)$ and the transition probability $a_{q_{t-1}q_t}$.

The Viterbi algorithm operates as follows. We define the quantity $\delta_t(i)$ as

$$\delta_t(i) = \max_{q_1,q_2,\ldots q_{t-1}} P(q_1, q_2, \ldots q_t = i, o_1 o_2 \ldots o_t|\lambda) \tag{2}$$

i.e. the probability at time t of seeing observation o_t when in state $q_t = i$ given that we have made the optimal choice of state up to time t. We then have the recurrence relation

$$\delta_t(j) = \max_i \left((\delta_{t-1}(i)\ a_{ij}) \cdot b_j(o_t) \right) \tag{3}$$

from which we find the highest probability of seeing the observation sequence O when we choose the optimal sequence of states Q. This is done in $\mathcal{O}(TN^2)$ time. To retrieve the state sequence Q we keep track of the argument which maximizes Equation 3 in a matrix Ψ.

$$\begin{pmatrix} \cdots & \psi_{t,1} & \psi_{(t+1),1} & \psi_{(t+2),1} & \psi_{(t+3),1} & \cdots \\ \cdots & \psi_{t,2} & \psi_{(t+1),2} & \psi_{(t+2),2} & \psi_{(t+3),2} & \cdots \\ \cdots & \psi_{t,3} & \psi_{(t+1),3} & \psi_{(t+2),3} & \psi_{(t+3),3} & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \cdots & \psi_{t,N} & \psi_{(t+1),N} & \psi_{(t+2),N} & \psi_{(t+3),N} & \cdots \end{pmatrix}$$

Keeping this $N \times T$ matrix requires $\mathcal{O}(TN)$ memory.

The optimal state sequence is retrieved by noting the final state which achieves the highest probability of observing O, $q_T^* = \text{argmax}_{(1 \le i \le N)}(\delta_T(i))$ and evaluating $q_t^* = \psi_{(t+1),q_{t+1}^*}$ for $t = T-1, T-2, \ldots 1$. Figure 1 shows an example matrix Ψ with the optimal paths resulting from Equation 3 shown.

The path matrix Ψ contains a lot of redundant information. If we consider the path matrix from some column τ onwards, which we call the start of our reachability window, then we propose to keep track of what paths in the path matrix Ψ remain reachable at any time $\tau + t$ in the future. We do this by defining a reachable set $\mathcal{R} = (r_1, r_2, \ldots, r_N)$ at time τ and initializing it to every reachable state at τ, $\mathcal{R} = (1, 2, \ldots, N)$.

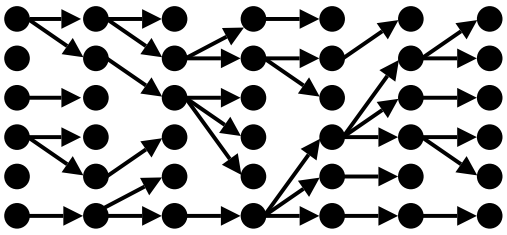

Figure 1. An example of the paths through the optimal state array Ψ

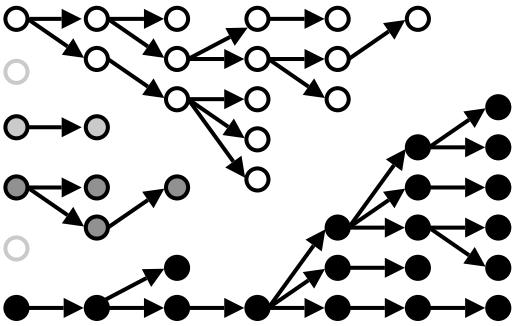

Figure 3. Representing the path matrix in Figure 1 as trees

At time step $\tau+1$ we create a new reachable set $\mathcal{R}'$ as we update the pointer matrix by setting $r'_i = r_{\psi_{(\tau+1),i}}$ as soon as we calculate the value $\psi_{(\tau+1),i}$, which is the argument which maximizes Equation 3,

$$\psi_{t,i} = \arg\max_i \left(\delta_{t-1}(i)\ a_{i,j}\right) \cdot b_j(o_t))$$

We keep track of whether there is more than one reachable state at time τ by keeping a boolean variable, *MultipleRootStates*, which we set to FALSE at the start of each new time step, and which we set to TRUE if $r'_i \neq r'_{i-1}$ as we update each r'_i value after the first,

$$\textit{MultipleRootStates} = \textit{MultipleRootStates}\ \text{OR}\ (r'_i \neq r'_{i-1}) \tag{4}$$

If *MultipleRootStates* is TRUE then we start calculating the next time step in the path matrix after setting $\mathcal{R} = \mathcal{R}'$, and so start computing a new reachable set vector $\mathcal{R}'$ for the new time step.

For example, the following 3 state section of a path matrix

$$\begin{pmatrix} \cdots & \psi_{\tau,1} & 1 & 2 & 1 & \cdots \\ \cdots & \psi_{\tau,2} & 1 & 2 & 2 & \cdots \\ \cdots & \psi_{\tau,3} & 2 & 3 & 2 & \cdots \end{pmatrix} \tag{5}$$

shows that at time $\tau+1$, if you want to be in state 1 the best state to have been in is 1, for state 2 it was also 1, and for state 3 it was 2, and so on for time steps $\tau+2$ and $\tau+3$. We set the reachable state set vector at time τ to be $(1, 2, 3)$. We then have the following sequence of reachable state set vectors $\mathcal{R}$ from time τ:

$$\begin{matrix} \tau \\ \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix} \end{matrix} \quad \begin{matrix} \tau+1 \\ \begin{pmatrix} 1 \\ 1 \\ 2 \end{pmatrix} \end{matrix} \quad \begin{matrix} \tau+2 \\ \begin{pmatrix} 1 \\ 1 \\ 2 \end{pmatrix} \end{matrix} \quad \begin{matrix} \tau+3 \\ \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \end{matrix} \tag{6}$$

At time $\tau+3$ we would have only one reachable root state remaining (state 1 in this example) from our original reachable set from time τ. Using this information allows us to retrieve the optimal state sequence before time τ, after which we no longer need the path matrix prior to time τ, which can be discarded. We reset our new reachability window to start at time, $\tau' = \tau + 4$, and continue.

There are a number of advantages to keeping track of the reachable states in this way. The addition of the $\mathcal{R}$ vector requires very little memory overhead, and the vector is updated, along with the *MultipleRootStates* variable, at the same time as we update the path matrix Ψ via Equation 3. As we move to calculating the next timestep of the path matrix we simply check the *MultipleRootStates* variable to see if we can resolve the optimal path up to the start of our reachability window. The algorithms runtime complexity remains $\mathcal{O}(TN^2)$. In addition, where it is possible to embed the transition probabilities $a_{i,j}$ in an underlying parameter space such that the costs can be expressed in terms of a distance between parameter values corresponding to states, then we can use the fast algorithms for large state-spaces in conjunction with our memory reducing approach to provide $\mathcal{O}(TN)$ runtime [4].

Figure 2 shows the procedure for retrieving the optimal path once the *MultipleRootStates* variable is FALSE. The algorithm provides a mechanism for implementing a solution in realtime for unending streams of observations to the HMM. If we want to provide an estimate of the optimal path at a time point where the *MultipleRootStates* variable is TRUE we can provide the path segment in the current reachability window corresponding to the path ending in the most likely state at the current time point. This would then supply the optimal sequence up to the start of the current unfinished reachability window, and a best guess for the current unfinished window. This guess could later be modified when new data becomes available and the reachability window closes, i.e. the *MultipleRootStates* variable becomes FALSE.

The path matrix Ψ can be explicitly represented as a collection of rooted trees representing the optimal path, as shown in Figure 3. The reachability window ends when new paths are all being added to the same tree. At any point before then we can use the current reachability set to de-

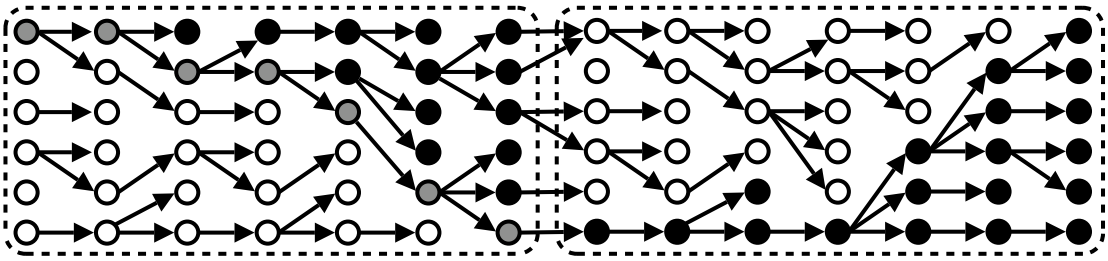

Figure 2. An example of finding the traceback path in the previous reachability window once the end of the current reachability window is found. The grey path shows the optimal set of state changes

cide which trees can be removed because they are no longer reachable. While this could allow us some marginal decreases in memory usage, it has been found empirically that for our problems the trees are reasonably short, and all trees are very short relative to our T. For long state sequence problems it is unlikely that the increased runtime complexity will be worth the marginal memory savings.

The recent work of Srámek et al. also exploits the same structural property of the pointer matrix Ψ to discover the reachable set of states at some time step t [14]. Their proposed algorithm maintains a compressed tree structure representing the reachable states, and a number of auxiliary data structures to enable the tree to be maintained at each new time point. This results in an $\mathcal{O}(N)$ computation at each time point to maintain the tree structures which they indicate leads to an experimentally validated 5% increase in run time for the general $\mathcal{O}(TN^2)$ algorithm. The technique which we have developed exploits this property of the pointer matrix using a simpler bookkeeping approach which is both maintained and checked in the same loop as the pointer matrix is calculated, and produces no measurable change in the running time of the algorithm.

There remains the question of how long we need to wait before there is only one state in the set of reachable states at time t. In general, this will depend on the data and the model. For example, if the probabilities of changing state $a_{i,j}$ are set to 0 then the worse case of never finding a single reachable state at any time point occurs. In long markov problems, however, it will nearly always be the case that any possible observation at some time point won't change the optimal states before some time in the distant past; there is an horizon of influence for new data which we are detecting with our reachability set. Once the reachability window is found, all possible paths to optimally account for future observations will have the initial path up to the start of the reachability window in common. It has been shown in a random walk model for a two-state HMM with uniformly random input of length T, that the time to wait for a single reachable state increases as $\log(T)$ [14]. In the typical markov models used for streaming data and large data sets, we can show experimentally that our technique significantly reduces memory usage with no measurable increase in running time.

3 HMM models for profiling activity levels

The incremental Viterbi algorithm described in the previous section could be used to find near realtime solutions to streaming observation sequences where we use some heuristic to account for the unlikely event that the reachability windows does not close before our memory usage is exhausted. In this section, however, we outline the large state sequence HMMs which motivated the algorithms development. The observation sequences which we are interested in understanding with a HMM originate from computer hard drive files, and we are interested in understanding the patterns of user activities over time contained on those hard drives.

To do this we use our incremental algorithm to find optimal paths in our HMM for the extremely large sequences of observations which we are dealing with. We use a modified version of Kleinberg's burst detection HMM to detect activity levels as bursts of file time stamps in the time series we derive from the hard disk drives [9].

We model an individuals activity level as the hidden state in our HMM, and the rate at which we observe file events in our data sources are the observations. The burst model uses an exponential probability distribution to link states with observations (i.e. as our emission functions b), where the states index a discrete set of expected waiting times (α_i) between events in the probability distribution:

$$b_i(o_t) = \alpha_i e^{-\alpha_i o_t}$$

Given this emission probability we still have considerable freedom in assigning the expected rate of observations α_i when in state i. Kleinberg suggests using a geometric progressions starting at the mean rate of the observed data,

Table 1. A table of mappings between hidden state number and expected waiting times in the burst HMM

State	Expected waiting times
1	1 second
2	1 minute
3	1 hour
4	1 day
5	1 week
6	1 month

$\alpha_i = g \cdot s^i$ where g is the average waiting time (i.e., n/T where n is the number of events and T is the the total length of time between the first and last event), and s is a scale factor. We are often interested in looking at periods of lower than usual activity, or inverted bursts, and so a natural extension to Kleinberg's state model is to multiply g by a factor s^{-c} where c is the depth of inverted bursts we wish to consider. Another alternative mapping which we have found useful is to consider deterministic mappings between hidden state and expected waiting times, such as those in Table 1. This has the advantage of making the returned burst level sequence easier to understand, and the mapping is tunable to our problem domain.

Rather than maximize the product in Equation 3 we take the logarithm of this equation and convert the problem into one of minimizing a sum of cost functions, where the cost functions are inversely related to the probabilities of transition and emission.

The probability of changing state is assigned a value proportional to the number of events, $\propto 1/n$. The transition probabilities can be adjusted according to the number of state jumps if desired, and since we are looking for bursts of activity we set the costs associated with changing state to be smallest for remaining in the same state (implicitly defined as inversely related to $1 - P(\text{state change})$). It can be seen from Equation 3 that changing the matrix A by a constant positive $\kappa < a_{i,j}\ \forall i, j$ does not change the optimal path selection. This allows the burst model to have the intuitive property of having zero cost when staying in the same state. We also have some flexibility in considering the probabilities of changing state, and we define the costs associated with these probabilities in the minimization problem to be a function of the states in the transition, $\tau(i, j)$. For this work we set the transition costs as

$$\tau(i,j) = \begin{cases} n^{-\gamma} & \text{if } i \neq j \\ 0 & \text{if } i = j \end{cases}$$

and we set $\gamma = 1$ for the results in this paper. This cost function defines our state transition matrix, A, in the original problem formulation for the burst model. We can now derive the recurrence relation for the burst model [9]:

$$\delta_t(j) = -\ln b_i(o_t) + \min_i \left(\delta_{t-1}(i) + \tau(i,j)\right) \qquad (7)$$

which we can find an optimal path for using the incremental Viterbi algorithm described in the previous section.

The emission probabilities described address the issue of temporal bursts in time series data. We are also interested in finding bursts in the pattern of data labels which we observe in our time series. To do this, we look for changes in the number of events we need to observe before we see the next symbol we are interested in. This is similar to the 'batting average' model used by Aizen et al. to look at bursts in the number of people who purchase a particular product from a website [1]. In their case, they used the burst model described above with the exponential emissions probability density function replaced by an estimate of the bias that the current observed event will be of the same class as the events of interest. In their website example, this corresponded to an instantaneous estimate that a website hit would result in a product purchase.

The problem with modeling our data with this approach is that we want to be able to account for instances when the bias can be very low or very high. When we replace the exponential emissions probability in Equation 7 with the corresponding term for the cost of the bias β we get the terms [1]:

$$\begin{array}{rl} -\log\beta & \text{if } o_t = 1 \\ -\log(1-\beta) & \text{if } o_t = 0 \end{array}$$

which become unbounded as β approaches 0 or 1. To resolve this we convert our input stream from a series of Bernoulli trials to a count of the number of events between successes. When we do this the appropriate probability density function becomes the geometric distribution, which we substitute for the exponential distribution (which is the continuous analogue of the geometric distribution) in Equation 7, i.e. we use the probability mass function:

$$b_i(o_t) = (1 - p_i)^{o_t} \cdot p_i$$

where p_i is our state indexed bias estimate. This resolves the boundedness problems with high and low biases, and significantly reduces the length of the state sequence in many problems.

Using the incremental Viterbi algorithm from the previous section we can apply these burst models in near real-time to find optimal burst patterns in streaming data as it becomes available. If we do this for multiple streams simultaneously we have the ability to look for correlations in burst activities which might be of interest in monitoring

network traffic or financial data, for example. The problems which we're going to apply the above models to require the incremental algorithm for a different reason - they are too large. Their large size makes the usual Viterbi procedure infeasible, particularly if we wish to perform the analysis on modest hardware.

4 Application

Digital evidence sources such as computer hard drives contain an enormous amount of data, much of it with associated meta data. One of the tasks for agencies which collect such data is to link the information in the data to activities of the users of the devices. In this section we look at discovering the usage level of the device by modeling time series derived from the devices with the burst model described in the previous section.

The author's macintosh hard drive contained 4,269,865 in May 2007, and each file had associated with it three timestamps (the creation, modification and access times), providing a time series of 12,809,595 events. We use the exponentially distributed emission probabilities described in the previous section to obtain a burst model of the time series corresponding to the author's home directory, which contains just over 1 million files.

A 2-state model, with one state corresponding to events expected once an hour, and another state expecting events once a day, is used to provide a state sequence capturing the inter-day patterns of the author's computer usage. The last 6 months of this model as well as the multistate model described by Table 1 and applied to the same observations, is shown in Figure 4. The 2-state HMM is retrievable from the multi-state HMM, and the optimal path sequence of the two HMMs present a multiple resolution break down of the burst activity, which could be described in an hierarchical model [9]. The 2-state model reveals the discrete state changes which occur when the author is at work on a per day basis. This model recovers the author's working week, as well as holidays and periods of absence. The multistate model provides extra states to the 2-state model revealing discrete activity changes at the daily and finer resolution. The models successfully label the time series with symbols representing the hidden state of the author's activity status.

Figure 4 shows the multi-state model from Figure 4 for the period May 7 through 17, 2007. User activities affect the creation of timestamps on hard drives in a number of different ways. For example, bursts in timestamps can be the result of moving large directories or unzipping files, depending on the operating system involved. In Figure 4 a time series anomaly on Saturday May 12 can be identified. The computer was not used by the author on this day, however it was awoken from sleep which resulted in it downloading emails. The last email downloads represent the discrete end of this state, which the burst model detects. The labelled sequence of activities from multiple hard drives could be used to find patterns in activities amongst groups of users of those hard drives.

In addition to examining temporal bursts in user activity, we are also interested in looking at patterns in the type of activities in the time series which are available and examining how these might relate to the users activities. We can use the geometric model to look at bursts in the proportion of files which have a property of interest. In Figure 4 we look at the proportion of file timestamps which are associated with files which hava 'java' in their name or path. The burst states correspond to a bias of i/N where i is the state and N is the maximum number of states, set to 10 in this case. The bias represents the probability that the next time event will be an event of interest. Once each file has a state associated with it from the geometric model, we have plotted the state against the timestamp of the file object in Figure 4. A state of 0 corresponds to a bias of $1/10^6$. As can be seen in the figure, state bias's near to 0 and 1 can give us quantitative information about our time series.

Each of the models above would require considerable memory resources if solved with the Viterbi algorithm. Table 2 shows the amount of memory which was required when using the incremental Viterbi algorithm on these problems. The time to execute the incremental Viterbi algorithm was found to be slightly less or equal to the usual Viterbi algorithm in all cases, presumably due to the overhead of large amount of memory management.

5 Conclusion

In this paper we have described an incremental Viterbi algorithm suitable for finding optimal path sequences in a wide range of HMMs. The algorithm was shown to dramatically reduce the usual $\mathcal{O}(TN)$ memory requirements of the Viterbi algorithm while still finding the optimal path sequence for a burst detection HMM. The incremental algorithm did not increase the $\mathcal{O}(TN^2)$ Viterbi runtime, and can be implemented to find paths in problems where the $\mathcal{O}(TN)$ runtime complexity large state-space algorithms can be used. In addition to enabling the memory-efficient solutions to large state sequence problems the incremental algorithm has the potential to provide realtime optimal paths in online problems.

We used the incremental algorithm to find optimal burst sequences in large time series derived from digital evidence sources for the purpose of linking computer user activities to patterns in the evidence. Kleinberg's burst algorithm is modified and used for this purpose [9]. Digital evidence sources typically provide extremely large data sets, and the incremental algorithm is shown to be suited to finding optimal paths in HMMs applied to such problems.

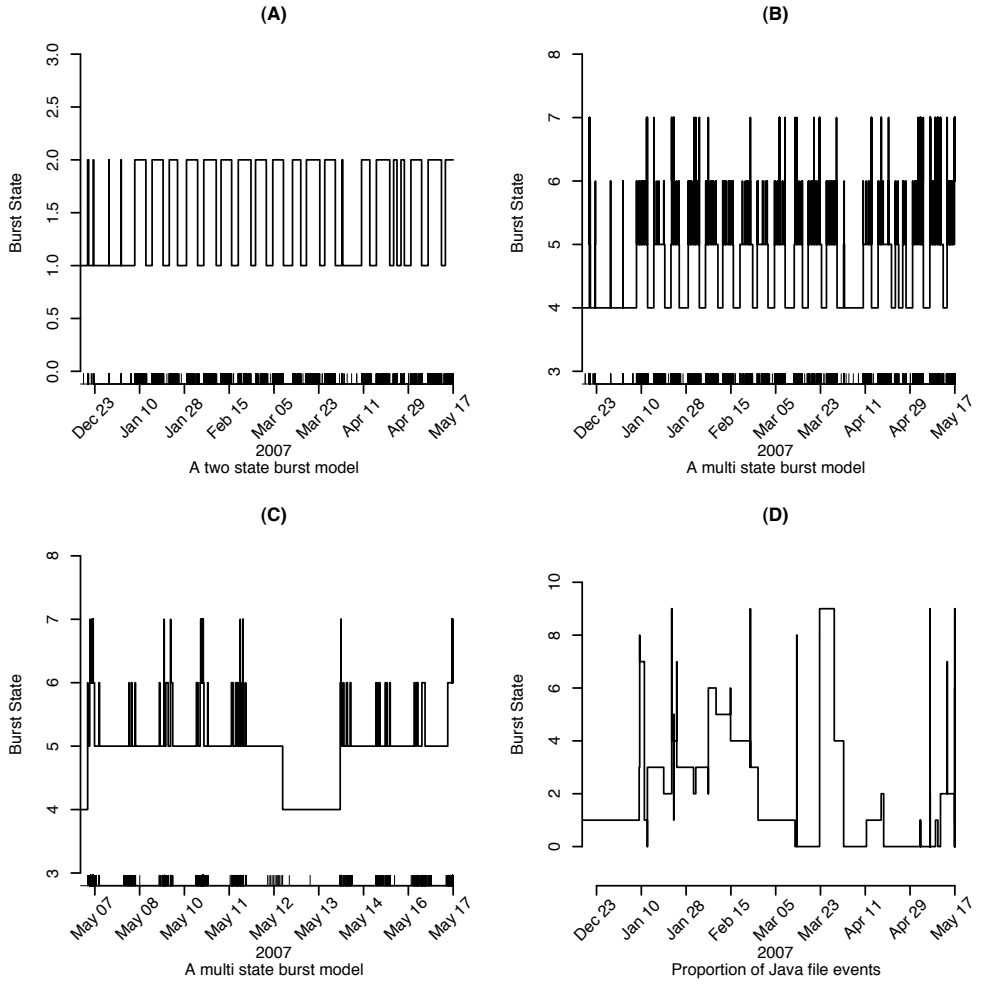

Figure 4. Results from applying the burst finding HMM using the incremental Viterbi algorithm. (A) shows the last 6 months of the two state burst model applied to the author's hard drive time stamps. (B) shows the last 6 months of the multi-state model applied to the author's hard drive time stamps. (C) shows the last few weeks of the burst activities in (B), showing the anomaly on Saturday 12th May. (D) shows a geometric burst HMM applied to finding the proportion of Java files in the author's hard drive time stamp events.

Table 2. Table of maximum path matrix sizes when applying the incremental viterbi algorithm used in Figure 4.

HMM Model	Discrete Time Points	State Count	Maximum Path Matrix Columns Required
2-State Exponential Burst Model	3,207,671	2	27
Multi-State Exponential Burst Model	3,207,671	6	140
Multi-State Geometric Burst Model	465,984	10	1,145

Additional research directions include applying the algorithm in realtime situations and attempting to characterize the problem structures to predict the likely width of the reachability window in the path matrix.

References

[1] J. Aizen, D. Huttenlocher, J. Kleinberg, and A. Novak. Traffic-based feedback on the web. *Proceedings of the National Academy of Sciences*, 101:5254–5260, 2004.

[2] C. Cui and H. Kitagawa. Topic activation analysis for document streams based on document arrival rate and relevance. *ACM Symposium on Applied Computing*, pages 1089–1095, 2005.

[3] S. R. Eddy. What is a hidden markov model? *Nature Biotechnology*, 22(10):1315–1316, 2004.

[4] P. Felzenszwalb, D. Huttenlocher, and J. Kleinberg. Fast algorithms for large-state-space hmms with applications to web usage analysis. In *Advances in Neural Information Processing Systems (NIPS)*, 2003.

[5] P. F. Felzenszwalb and D. P. Huttenlocher. Efficient matching of pictorial structures. *Proceedings of the IEEE Computer Vision and Pattern Recognition Conference*, pages 66–73, 2000.

[6] G. D. Forney. The viterbi algorithm. *Proceedings of the IEEE*, 61(3):268–278, 1973.

[7] T. Fujiki, T. Nanno, Y. Suzuki, and M. Okumura. Identification of bursts in a document stream. In *Proceedings of the First International Workshop on Knowledge Discovery in Data Streams*, 2004.

[8] J. A. Grice, R. Haghey, and D. Speck. Reduced space sequence alignment. *Computer Applications in the Biosciences*, 13(1):45–53, 1997.

[9] J. Kleinberg. Bursty and hierarchical structure in streams. In *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2002.

[10] J. Krumm, J. Letchner, and E. Horvitz. Map matching with travel time constraints. In *Society of Automotive Engineers (SAE) 2007 World Congress*, April 2007. Paper 2007-01-1102.

[11] J. Krumm, L. Williams, and G. Smith. SmartMoveX on a graph - an inexpensive active badge tracker. In *Fourth International Conference on Ubiquitous Computing (UbiComp 2002)*, pages 299–307, September/October 2002.

[12] L. R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, February 1989.

[13] R. Šrámek. The on-line Viterbi algorithm. Master's thesis, Department of Computer Science, Faculty of Mathematics, Physics and Informatics, Comenius University, Bratislava, 2007.

[14] R. Šrámek, B. Brejová, and T. Vinař. On-line Viterbi Algorithm and Its Relationship to Random Walks. *ArXiv e-prints*, 704, Mar. 2007.

[15] A. J. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, 13(2):260–269, 1967.

[16] C. V. Wright, F. Monrose, and G. M. Masson. On inferring application protocol behaviors in encrypted network traffic. *Journal of Machine Learning Research*, 7:2745–2769, 2006.

[17] J. Yi. Detecting buzz from time-sequenced document streams. In *EEE '05: Proceedings of the 2005 IEEE International Conference on e-Technology, e-Commerce and e-Service (EEE'05)*, pages 347–352, Washington, DC, USA, 2005. IEEE Computer Society.