# Online Decoding of Markov Models under Latency Constraints

**Mukund Narasimhan** MUKUNDN@MICROSOFT.COM
**Paul Viola** VIOLA@MICROSOFT.COM
**Michael Shilman** SHILMAN@MICROSOFT.COM
Microsoft Corporation, One Microsoft Way, Redmond WA 98195

## Abstract

The Viterbi algorithm is an efficient and optimal method for decoding linear-chain Markov Models. However, the entire input sequence must be observed before the labels for any time step can be generated, and therefore Viterbi cannot be directly applied to online/interactive/streaming scenarios without incurring significant (possibly unbounded) latency. A widely used approach is to break the input stream into fixed-size windows, and apply Viterbi to each window. Larger windows lead to higher accuracy, but result in higher latency.

We propose several alternative algorithms to the fixed-sized window decoding approach. These approaches compute a certainty measure on predicted labels that allows us to trade off latency for expected accuracy dynamically, without having to choose a fixed window size up front. Not surprisingly, this more principled approach gives us a substantial improvement over choosing a fixed window. We show the effectiveness of the approach for the task of spotting semi-structured information in large documents. When compared to full Viterbi, the approach suffers a 0.1 percent error degradation with a average latency of 2.6 time steps (versus the potentially infinite latency of Viterbi). When compared to fixed windows Viterbi, we achieve a 40x reduction in error and 6x reduction in latency.

Appearing in *Proceedings of the 23$^{rd}$ International Conference on Machine Learning*, Pittsburgh, PA, 2006. 

## 1. Introduction

Many problems in information extraction can be reduced to segmenting/labeling sequences, including part-of-speech tagging (in natural language applications), phoneme tagging (in speech applications), and sequence alignment (in bioinformatics applications). Hidden state Markov Models are widely used for solving such problems. The hidden state corresponds to the label for each observation in the input sequence, and the Markov assumption specifies that the state corresponding to time step (or location) $n$ is independent of the state corresponding to time steps prior to $n-2$ given the state of time step $n-1$. Two such models are linear chain Hidden Markov Models (HMMs) (Rabiner, 1989; Leek, 1997) and Conditional Random Fields (CRFs) (Lafferty et al., 2001; Taskar et al., 2002; Berger et al., 1996; McCallum et al., 2000; Pinto et al., 2003). Both models have been widely used for solving problems dealing with semi-structured input sequences due to their simplicity and their effectiveness.

The Viterbi algorithm has been widely used for decoding such models. This algorithm requires computing a forward pass over the input sequence to compute probabilities/scores, followed by a reverse pass to compute the optimal state/label sequence. Therefore, all the data must be seen before any of the hidden states can be inferred, and hence it cannot be directly applied to real-time/reactive applications, or to applications where there are strong latency and/or memory constraints.

A simple example of such an application is malicious activity detection in network packet streams. A Markov Model can be trained on data from packet logs (which are annotated by experts to indicate interesting states), and then deployed in the network to detect attacks in real-time. This approach is potentially superior to hand coded rule-based systems because we can more easily update the models based on large transaction logs. However, in this setting, the input stream is

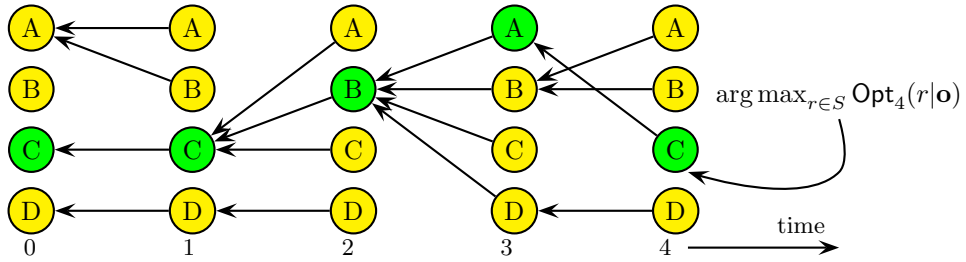


*Figure 1.* The Viterbi algorithm maintains a chain of back-pointers in order to compute the optimal path by a backward traversal. The optimal path is shown in green.

conceptually infinite, and we would like to know and react to the attacks as soon as possible. Therefore, the Viterbi algorithm is not very well suited for this problem. By generating the state labels greedily without looking at the later observations, we risk increasing the false alarm rate (which is also problematic in this setting because a high false alarm rate causes the human operator to ignore attack warnings).

What is desired is a system which will intelligently wait until it has seen enough of the input to generate labels with sufficient confidence (based on the specified costs for false positives and false negatives). False positives and false negatives can have a real monetary cost. For example, in credit card fraud monitoring, observing additional transactions before declaring the credit card as stolen can cost the credit card agency money (for the fraudulent transactions), while false alarms can cause the users to be unnecessarily annoyed (which can cost the company money if the customer decides to cancel his/her card). Such applications have been the target of a lot of research (Chan & Stolfo, 1998; Cox et al., 1997; Dumouchel & Schonlau, 1998; Fawcett & Provost, 1997; Lane & Brodley, 1997; Lee et al., 1998). There are many other types of on-line or reactive systems such as human activity detection and patient monitoring.

Researchers in speech (Seward, 2003) and gesture (Eickeler et al., 1998) recognition identify this problem and solve it by applying Viterbi to simple, fixed-size chunks of the input sequence. Viterbi is one of many ways to decode sequence data. Search based approaches, including approximations such as beam search and simulated annealing, have been used to accelerate sequential decoding (Daumé III & Marcu, 2005). However, accelerating the decoding process does not necessarily reduce latency: one still has to wait for the entire input to be present to run these faster decoders.

In this paper, we propose two algorithms to reduce the latency between the time the decoder receives the observation corresponding to a particular time step and the time at which the decoder generates the label for the observation.

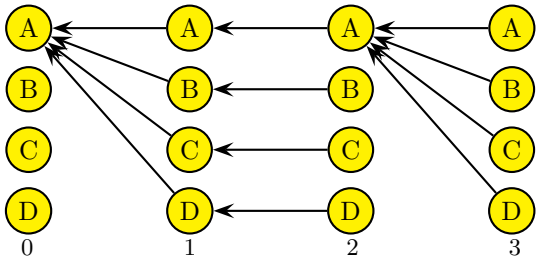


*Figure 2.* Decoding using a fixed window size can lead to arbitrarily bad performance.

## 2. Preliminaries and Notation

Let $S = \{s_1, s_2, \ldots, s_k\}$ be the set of states of the Markov model, and $S^n$ the set of all state sequences of length $n$. The probability of a state sequence $\mathbf{s} = s^{(0)}s^{(1)}s^{(2)}\ldots s^{(n-1)}$ given the observations $\mathbf{o}$ for a linear-chain CRF is proportional to the following unnormalized product:

$$\mathsf{P}_{[0,n]}(\mathbf{s}\,|\mathbf{o}) \propto \prod_{t=1}^{n} \phi_t\left(s^{(t-1)}, s^{(t)}, \mathbf{o}\right)$$

where the $\phi_t$ are potential functions of the (maximal) cliques of the underlying linear-chain graphical model, and are typically of the form

$$\phi_t\left(s^{(t-1)}, s^{(t)}, \mathbf{o}\right) = \exp\left(\sum_{f \in \mathcal{F}} \lambda_f f\left(t, s^{(t-1)}, s^{(t)}, \mathbf{o}\right)\right)$$

Each $f \in \mathcal{F}$ represents a feature and is a function of the last-state, the present-state, the (entire) observation sequence, and the position. Observe that such functions can ignore arguments, and so can represent dependencies between states and observations, between

neighboring states, or all three. The $\lambda_f$ are weights that are determined by some training procedure. For linear chain models, the observations also form a sequence, with one observation per time step. We will denote by $\mathbf{o}_{[0:t]}$ the set of observations corresponding to time steps 0 through $t$.

### 2.1. The Viterbi Algorithm for Decoding CRFs

The Viterbi algorithm for decoding CRFs is based on the fact that the score of optimal sequences of length $n$ ending in a particular state $r \in S$ can be computed efficiently by dynamic programming. The score $\mathsf{Opt}_t(r|\mathbf{o})$ of an optimal sequence of length $t+1$ ending in the state $r \in S$ is given by

$$\mathsf{Opt}_t(r|\mathbf{o}) = \max_{\mathbf{s} \in S^{t+1}, s^{(t)}=r} \mathsf{P}_{[0,t]}\left(\mathbf{s}\,|\mathbf{o}\right)$$

$\mathsf{Opt}_t(r)$ can be computed efficiently using the following recurrence relation:

$$\mathsf{Opt}_t(a|\mathbf{o}) = \max_{b \in S}\Big[\mathsf{Opt}_{t-1}(b|\mathbf{o}) \cdot \phi_t\left(b, a, \mathbf{o}\right)\Big]$$

Once we have $\mathsf{Opt}_t(r|\mathbf{o})$ for all $r \in S$, we may compute the score of an optimal sequence as

$$\max_{\mathbf{s} \in S^n} \prod_{t=1}^{n} \phi_{t-1}(s^{(t-1)}, s^{(t)}, \mathbf{o}) = \max_{r \in S} \mathsf{Opt}_n(r|\mathbf{o})$$

Note that the last state in the optimal sequence is $\mathsf{argmax}_{r \in S}\mathsf{Opt}_n(r|\mathbf{o})$. We can compute the entire optimal sequence by storing back pointers :

$$\mathsf{Prev}_t(a|\mathbf{o}) = \mathsf{argmax}_{b \in S}\Big[\mathsf{Opt}_{t-1}(b|\mathbf{o}) \cdot \phi_t\left(a, b, \mathbf{o}\right)\Big]$$

Once we have computed $\mathsf{Opt}_t(r|\mathbf{o})$ and $\mathsf{Prev}_t(r|\mathbf{o})$ for all $0 \leq t < n$, and every $r \in S$, we can find the optimal state sequence by following the chain of back pointers. So, in the example shown in Figure 1, if $\mathsf{argmax}_{r \in S}\mathsf{Opt}_4(r|\mathbf{o})$ is C, then following the trail of back pointers yields the darker (or green) states as the optimal state sequence.

## 3. Latency-Accuracy Tradeoffs

The primary disadvantage of the Viterbi algorithm in latency-sensitive applications is that the optimal state sequence cannot be computed until the entire input has been observed. In this section, we discuss four different approaches to generating the hidden states early, including the fixed size window heuristic, a lossless transitive-closure algorithm, and two new algorithms based on a dynamically-sized window.

### 3.1. Simple Chunking

A simple way of generating the hidden states with bounded latency is to fix a window (of length less than the desired latency), and apply the Viterbi algorithm on each window. One problem with this approach is illustrated in Figure 2, where a fixed window size can generate poor results. In this example, the optimal labels for time steps 0 and 1 are **A** and **A** regardless of the state scores. A window of size 3 however can lead to the selection of the incorrect path. This could result in 66% error rates (as we get 2 states wrong in each window of length 3). Even worse, the resulting state sequence does not yield a set of valid transitions in the model. We could have obtained the correct result with no increase in latency by simply changing the end points of the windows. Admittedly, this example is pathological, and unlikely to arise in real applications. However, it illustrates some of the problems that can arrise by a simple minded choice of window.

### 3.2. Pruning by Transitive Closure

In some cases it is possible to find the optimal state sequence without incurring the full latency of the Viterbi algorithm. One such example is shown in Figure 1. At time step 3, all back pointers point to the same state, and so regardless of the states in the optimal sequence after time step 2, the optimal states for time steps 0 through 2 must be **CCB**. In practice, it is not uncommon for a single state to dominate all others to therefore act as the parent of all subsequent states.

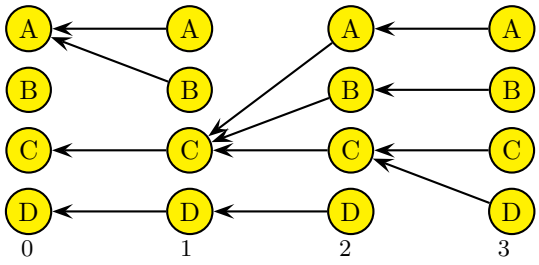


*Figure 3.* Unambiguous decoding without unique backpointers: At time step 3, it is clear that **C** is the correct state label for time steps 0 and 1.

However, we can go beyond this simple observation. In the trellis shown in Figure 3, we can uniquely decode states for time steps 0 and 1 once we observe the backpointers for every state at time 3, even though not all the back pointers point to the same state at any time step. In order to determine when cases like this occur, we need to maintain the transitive closure of the backpointers. Let us denote by $\overline{\mathsf{Prev}}_t(t_0, a|\mathbf{o})$ the state at time step $t_0$ that is reachable through a

chain of back pointers starting at state $a$ at time step $t$. For example, in the trellis shown in Figure 3, we have $\overline{\mathsf{Prev}}_3(0, A|\mathbf{o}) = C$ and $\overline{\mathsf{Prev}}_2(0, D|\mathbf{o}) = D$. This too can be computed efficiently using the following recurrence relation:

$$\overline{\mathsf{Prev}}_t(t_0, a|\mathbf{o}) = \overline{\mathsf{Prev}}_{t-1}(t_0, \mathsf{Prev}_t(a|\mathbf{o})|\mathbf{o})$$

The previous observation amounted to saying that if for some time step $t > t_0$, we have $\overline{\mathsf{Prev}}_t(t_0, a|\mathbf{o}) = s$ for every $a \in S$, then we may safely generate state $s$ as the label for time step $t_0$. Besides generating the labels early, the procedure has the advantage of not requiring backpointers for the time steps we have already emitted labels for (by contrast, Viterbi requires us to maintain backpointers all the way back to the time 0). The leads to a substantial saving of memory, while still generating exactly the same set of labels as Viterbi (eventually) will, only (potentially) earlier.

Note that there is a very simple implementation of transitive closure pruning which can be used in almost any Markov decoding application. One typically stores the information about states and paths in a matrix indexed by state and time (this matrix stores scores and backpointers). Alternatively one can store the information about each path as a linked list of states, where each state node stores the current path scores and a pointer to some state at the previous time step. With each time step the paths are extended to form longer paths. At some point in the computation, entire paths become dead ends: there is no extension of that path to the next time step (because the path probability is lower than all others). At this point the entire path can be removed and deallocated. By using a reference counting mechanism (or a more general garbage collection algorithm) deallocation is performed completely automatically.

### 3.3. Probabilistic Methods

So far, we have been very rigid about our decisions: We generate the (optimal) state for a particular time step when we are sure that it is the only state that is reachable through a sequence of back-pointers. The advantage of this rigidity is that the labels generated are the same as those generated by Viterbi. The disadvantage is that we could be passing up opportunities for early generation of some states. In contrast to the Chunked approach, we consider algorithms that dynamically choose where to break the input stream based on an uncertainty measure. Below, we present two alternative approaches, Online Step and Online Variable Window, that leverage this insight in slightly different ways.

#### 3.3.1. Online Step

The essence of the online step algorithm is that as we scan forward through the input sequence, the initial state becomes increasingly certain. At the same time, the latency, which is the average number of states that must be examined before the algorithm can output a single state, also increases. Once our certainty estimate reaches a dynamically-computed threshold in terms of the latency, we output this state and begin to estimate the next state in the sequence.

Assume that for each state $r \in S$, we can determine (or at least estimate) the probability that $r$ is the state at time step $t_0$ on an optimal (Viterbi) path. In this case, we may decide to generate a state for time step $t_0$ as long as there is only one state with sufficiently high probability (or equivalently if all the states but one have very low probability). In general, we cannot determine the exact probability that a given state is on the optimal path without all the observations. However, we may estimate this probability by using a limited number of future observations. Given some criteria for accuracy, it is possible in some cases to generate labels for past states well before the entire input is observed.

It should be noted that there may be a set of future observations for which some past state remains ambiguous. Hence, to get any guarantees on the performance of an online algorithm for this problem, we need to make some assumptions about the input. Of course, we will show (experimentally) that the assumption we make is a reasonable one.

Let $\tilde{\mathsf{P}}_T(t_0, a|\mathbf{o}_{[0:T]})$ be an estimate of the probability that state $a$ is on the Viterbi path at time step $t_0$ after observing the input up to time $T > t_0$. This probability can be computed as follows. Since we have observed the input up to time step $T > t_0$, we can compute $\mathsf{P}_T(b|\mathbf{o})$, the probability of being at state $b$ at time step $T$ for each state $b \in S$. If state $b$ is the state on the Viterbi path at time step $T$, then we must be in state $\overline{\mathsf{Prev}}(t_0, b|\mathbf{o}_{[0:T]})$ at time step $t_0$. Define

$$\tilde{\mathsf{P}}_T\left(t_0, a\,\middle|\mathbf{o}_{[0:T]}\right) = \sum_{\substack{b \in S \\ \overline{\mathsf{Prev}}_T(t_0, b|\mathbf{o}_{[0:T]}) = a}} \mathsf{P}_T(b|\mathbf{o}_{[0:T]})$$

a distribution on the states at time step $t_0$, conditioned on observations upto and including $T$. We can use this distribution as a proxy for the true distribution of the state $a$ being on the Viterbi path at time step $t_0$ (which could not be computed without observing the entire observation sequence). Clearly, if this distribution places significant probability only on one state, then it is reasonable to generate the label for time

step $t_0$, while it is less reasonable to do so if the distribution has significant mass at more than one state. We can quantify this with a measure of how peaky the distribution is.

One simple measure is the expected classification error rate under this distribution, which is 1 minus the probability of the highest scoring state. In an interactive setting, where the user is presented with more than one option, an alternative measure could be the residual uncertainty in the state label, such as the entropy of the distribution. If we were extremely intolerant to mistakes (i.e., if it is unreasonable to generate a state label unless we are absolutely certain of the label), then a suitable measure might take on the value 1 if there is more than one state with non-zero probability, and 0 if all the probability is concentrated on a single state. In fact, the last measure will lead to the same results as Pruning by Transitive Closure (Section 3.2). In all these cases, the higher the measure, the more costly it is to generate a state label for time step $t_0$.

Let $M_T(t_0|\mathbf{o}_{[0:T]})$ be the measure chosen such that $M_T(t_0|\mathbf{o}_{[0:T]})$ is small when the $\tilde{\mathsf{P}}_T\left(t_0, a\,|\mathbf{o}_{[0:T]}\right)$ is concentrated around a single state. Observe that $M_T(t_0|\mathbf{o}_{[0:T]})$ measures the uncertainty in a single state label (that at time step $t_0$) given the observations upto and including $T \geq t_0$. We will assume that this measure is bounded above as it is in the case of expected classification error rate (by 1) or the entropy of the probability distribution (which is bounded by the logarithm of the number of states).

We wish to pick a time $T > t_0$ so as to minimize this measure. On the other hand, we also want to minimize $T - t_0$, the number of additional observations we need to make before generating the state label. Combining these objectives, we seek a time step $T$ so that

$$f(T) = M_T(t_0|\mathbf{o}_{[0:T]}) + \lambda \cdot (T - t_0)$$

is minimized. The parameter $\lambda$ is set based on the accuracy/latency trade off desired. Larger values of $\lambda$ penalize latency more, and smaller values of $\lambda$ penalize errors more.

Clearly if $M_T(t_0|\mathbf{o}_{[0:T]})$ can vary arbitrarily with $T$, then any deterministic (non-randomized) algorithm can do arbitrarily badly. Therefore, we make the following assumption on $M$ which is well supported by the data for the application we are interested in: $M_T(t_0|\mathbf{o}_{[0:T]})$ is a decreasing function in $T$ for any fixed $t_0$. Essentially what this says is that the more we are allowed to observe the future, the more confident we can be about the present state. This seems to be a reasonable assumption as it is true for over 98% of the examples in our data set. Under this assumption, we will show that the online algorithm shown in Figure 4 is the best possible deterministic online algorithm. We judge algorithms in terms of their competitive ratio. An algorithm for optimizing $f(t)$ is $\alpha$-competitive if for every possible input (under the specified assumptions), the ratio of the solution produced by the algorithm to the optimal solution is no more that $\alpha$ (upto a constant).

$t_0 \leftarrow 0; T \leftarrow 0;$
$\forall a \in S:\ \mathsf{pprev}(a) \leftarrow a$ $\quad //\overline{\mathsf{Prev}}_T(t_0, a\|\mathbf{o}_{[0:T]})$
$\forall a \in S:\ \mathsf{pp}(a) \leftarrow \phi_{t_0}(s^{(t_0)}, a)$ $\quad //\tilde{P}_T(t_0, a\|\mathbf{o}_{[0:T]})$
$\forall a \in S:\ \mathsf{p}(a) \leftarrow \phi_{t_0}(s^{(t_0)}, a)$ $\quad //P_T(t_0, a\|\mathbf{o}_{[0:T]})$
$m \leftarrow \mathsf{UncertaintyMeasure(pp)}$ $\quad //M_T(t_0\|\mathbf{o}_{[0:T]})$
**while** $t_0 \leq T$ **do**
  **if** $m < \lambda \cdot (T - t_0)$ or end-of-input **then**
    $\mathbf{s}^{(t_0)} \leftarrow \mathsf{argmax}_{a \in S} \mathsf{pp}(a)$
    $t_0 \leftarrow t_0 + 1;$
    $\forall a \in S:\ \mathsf{p}(a) \leftarrow \phi_{t_0}(s^{(t_0)}, a)$
    $\forall a \in S:\ \mathsf{pprev}(a) \leftarrow a$
    $t_1 \leftarrow t_0 + 1$
  **else**
    $T \leftarrow T + 1$
    $t_1 \leftarrow T$
  **endif**
  **for** $t \leftarrow t_1 \ldots T$ **do**
    $\forall a \in S : \mathsf{prv}(a) \leftarrow \mathsf{argmax}_{b \in S} \mathsf{p}(b) \cdot \phi_t(b, a, \mathbf{o}_{[0:T]})$
    $\forall a \in S : \mathsf{temp}(a) \leftarrow \mathsf{p}(\mathsf{prv}(a)) \cdot \phi_t(\mathsf{prv}(a), a, \mathbf{o}_{[0:T]})$
    $\mathsf{p} \leftarrow \mathsf{temp}$
    $\forall a \in S : \mathsf{pprev}(a) \leftarrow \mathsf{pprev}(\mathsf{prev}(a))$
    $\forall a \in S : \mathsf{pp}(a) \leftarrow \sum_{\{\, b \in S\, : \mathsf{pprev}(b) = a\}} p(b)$
  **endfor**
  $m \leftarrow \mathsf{UncertaintyMeasure(pp)}$
**endwhile**

*Figure 4.* The Online Step Algorithm

We have following performance guarantee for the algorithm show in Figure 4.

**Lemma 1.** *The algorithm shown in Figure 4 is 2-competitive under the assumption that $M_T(t_0|\mathbf{o}_{[0:T]})$ is a decreasing function in $T$ for any fixed $t_0$.*

*Proof.* Observe that since $M_T(t_0|\mathbf{o}_{[0:T]})$ is bounded from above, we will eventually generate a label for $t_0$ (no later than time step $t_0 + \frac{M_T(t_0|\mathbf{o}_{[0:T]})}{\lambda}$). This follows because $M_t(t_0|\mathbf{o}_{[0:t]}) \leq M_{t_0}(t_0|\mathbf{o}_{[0:t_0]})$, and so if we have not yet stopped, it must because $\lambda \cdot (t - t_0) < M_t(t_0|\mathbf{o}_{[0:t]}) \leq M_{t_0}(t_0|\mathbf{o}_{[0:t_0]})$, from which we get $t < t_0 + \frac{M_{t_0}(t_0|\mathbf{o}_{[0:t]})}{\lambda}$. This proves that the algorithm will always terminate (under the assumptions specified), even when the input sequence is infinitely long. To determine the competitive ratio of this algorithm, let us assume that the algorithm specified generates a label at time step $T$, while the actual optimal stopping time is $T_{\mathsf{opt}}$. If $T_{\mathsf{opt}} > T$, then cost associated with the optimal time step is $M_{T_{\mathsf{opt}}}(t_0|\mathbf{o}) + \lambda \cdot (T_{\mathsf{opt}} - t_0)$,

which increases with $T_{\mathsf{opt}}$. Therefore, from a competitive analysis point of view, the worst case occurs when $T_{\mathsf{opt}} = T+1$, and $M_{T_{\mathsf{opt}}}(t_0|\mathbf{o}_{[0:T_{\mathsf{opt}}]}) = 0$, in which case

$$\begin{aligned} f(T_{\mathsf{opt}}) &= f(T+1) \\ &= \lambda \cdot (T - t_0 + 1) \\ &> \lambda \cdot (T - t_0) \\ &= \frac{2 \cdot \lambda \cdot (T-t_0)}{2} \\ &> \frac{M_T(t_0|\mathbf{o}_{[0:T]}) + \lambda \cdot (T-t_0)}{2} \\ &= \frac{f(T)}{2} \end{aligned}$$

For all $t < T$, the worst case occurs if $M(T, t_0|\mathbf{o}_{[0:T]}) = M(t_0, t_0|\mathbf{o}_{[0:T]})$ (so the observations between $t_0$ and $T$ have provided no gain at all). In this case, the optimal solution is to pick $t = t_0$, and so the optimal value is $f(t_0) = M_{t_0}(t_0|\mathbf{o}_{[0:t_0]}) = M_T(t_0|\mathbf{o}_{[0:T]})$. Because the algorithm stops at time step $T$, and not before, we have

$$\lambda \cdot (T - t_0 - 1) \leq M_T(t_0|\mathbf{o}_{[0:T]}) = M_{t_0}(t_0|\mathbf{o}_{[0:t_0]}) \leq \lambda \cdot (T - t_0)$$

Therefore,

$$\begin{aligned} f(T) &= M_T(t_0|\mathbf{o}_{[0:T]}) + \lambda \cdot (T - t_0) \\ &= M_T(t_0|\mathbf{o}_{[0:T]}) + \lambda \cdot (T - t_0 - 1) + \lambda \\ &\leq 2 \cdot M_{t_0}(t_0|\mathbf{o}_{[0:t_0]}) + \lambda \\ &\leq 2 f(t_0) + \lambda \end{aligned}$$

Therefore, the algorithm picks a time which costs no more than 2 times the cost of any other time (upto a constant), and hence it is guaranteed to be 2-competitive. □

#### 3.3.2. Online Variable Window

The online variable window algorithm is similar to the step algorithm above, only rather than computing the certainty of a single state, it estimates the certainty corresponding to an entire time window of states. Once the certainty of that window surpasses a threshold with respect to the length of the window, the entire window is output and the process begins again.

As before, we cannot determine the optimal break points with absolute certainty unless we read in the entire input. In fact, any online algorithm can be made to behave arbitrarily badly by choosing the appropriate observation sequence. We will therefore, make some modest assumptions about the input sequence, and show that under these assumptions we can find the optimal online algorithm.

Let us begin by quantifying the "loss" in labeling accuracy using a smaller window instead of the entire observation sequence. Suppose we run the Viterbi algorithm on observations from time step $t_0$ to time step $T$, and let $s^{(T)}_{\mathsf{early}}$ be the state we pick for time step $T$ based on the observations up to time step $T$, and $s^{(T)}_{\mathsf{opt}}$ be the optimal state based on the entire input observation. We may not be able to actually compute $s^{(T)}_{\mathsf{opt}}$ because the input sequence may extend into the infinite future. To overcome this, we use $\mathsf{P}_T(s|\mathbf{o}_{[0:T]})$ as an estimate for the probability that $s$ is the state at time $T$ on the optimal Viterbi path. Let $a, b \in S$ be two potential states that can be generated at time step $T$. If we pick state $a$ for time step $T$, then the state for time step $T-1$ has to be $\mathsf{Prev}_T(a)$, and the state for time step $T-2$ has to be $\mathsf{Prev}_{T-1}(\mathsf{Prev}_T(a)) = \mathsf{Prev}^2_T(a)$ and so on. Therefore, the difference in the states in the sequence caused by choosing $a$ instead of $b$ is

$$\mathsf{Loss}_T(a, b|\mathbf{o}_{[0:T]}) = \sum_{k=0}^{T-t_0} 1_{\{\mathsf{Prev}^k_T(a) \neq \mathsf{Prev}^k_T(b)\}}$$

This loss function can be efficiently computed by this simple recurrence relation:

$$\begin{aligned} \mathsf{Loss}_T(a, b|\mathbf{o}_{[0:T]}) &= 1_{a \neq b} \\ &+ \mathsf{Loss}_{T-1}(\mathsf{Prev}_T(a|\mathbf{o}_{[0:T]}), \mathsf{Prev}_T(b|\mathbf{o}_{[0:T]}), \mathbf{o}_{[0:T]}) \end{aligned}$$

Therefore, the expected loss of picking the state $a$ at time step $T$ is

$$\mathsf{Loss}_T(a|\mathbf{o}_{[0:T]}) = \sum_{b \neq a} \mathsf{P}_T(b|\mathbf{o}_{[0:T]}) \cdot \mathsf{Loss}_T(a, b|\mathbf{o}_{[0:T]})$$

This measures the expected number of incorrectly generated states by the choice of $a$ at time step $T$. According to this loss function, the choice of state $s^{(T)}_{\mathsf{early}}$ at time $T$ that leads optimal loss for this window is given by

$$s^{(T)}_{\mathsf{early}} = \arg\min_{a \in S} \mathsf{Loss}(a|\mathbf{o}_{[0:T]})$$

Observe that this is a measure of loss for using the window $[t_0, T]$. As before, we want to choose a window $[t_0, T]$ that minimizes $\mathsf{Loss}_T + \lambda(T - t_0)$, which we can do using the same algorithm.

## 4. Experiments

For our experiments, we chose the problem of automatically spotting and extracting contact information in large documents. To run this application on medium to large sized documents in a resource limited device (such as a cell phone or a PDA), we

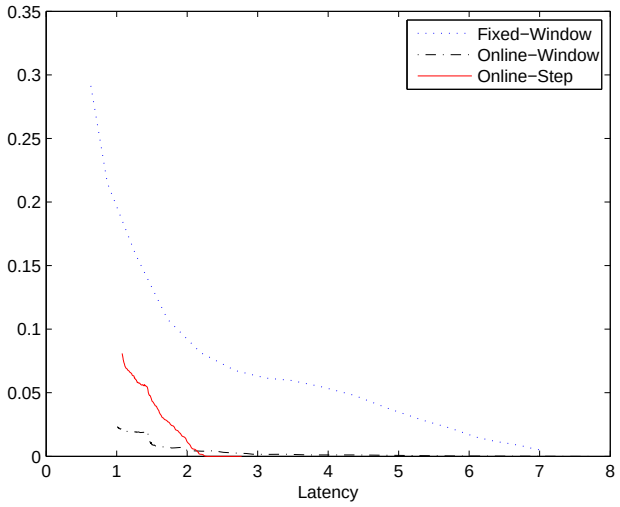


*Figure 5.* This plot compares the latency-error-rates for the three algorithms The Y-axis represents the decrease in accuracy over the Viterbi (optimal offline) algorithm.

need to be able to control the memory usage (and hence latency). We used a CRF that was trained on a set of 6399 documents (in this case email messages), and the test set consists of 6443 documents. The documents were labeled with 21 labels (NotContact, BeginContact, EndContact, FirstName, MiddleName, LastName, NickName, Suffix, Title, JobTitle, CompanyName, Department, AddressLine, City, State, Country, PostalCode, PhoneNumber, Email, InstantMessage, WebPage). More details about this application can be found in (Kristjansson et al., 2004).

The CRF achieves an accuracy rate of 98.91%. A baseline classifier (constructed by boosting a collection of decision trees of depth 3) achieves an accuracy rate of 81.34%. The baseline classifier uses only local features, based on the previous, current and next observations. These two classifiers serve as guideposts. On one hand, we have the CRF decoded using the Viterbi algorithm, which bases its decision on the entire input stream, and achieves an accuracy of 98.91%. On the other hand, we have the boosted classifier which only requires the previous and next observations, but achieves an accuracy of 81.34%. The algorithms that we have proposed in this paper lie somewhere in the middle. But as we will show, the accuracy is very close to that of the CRF, while still using only a very limited amount of future observations for each time step. Figure 5 shows the latency-accuracy trade-off performance of the two online algorithms. Both algorithms have one parameter: $\lambda$. This parameter allows us to tradeoff between latency and accuracy. Hence for each value of $\lambda$, we get one value for latency, and one value for accuracy. As we vary $\lambda$, we trace out various points of the latency-accuracy curve. $\lambda = 0$ gives the same accuracy of the CRF (it just allows for opportunistic pruning), and in the worst case, this could require us to look at the entire input stream. As we increase $\lambda$, the latency is bounded above, and decreases very sharply. In fact, we can achieve a degradation of 0.1% (i.e., only 0.1% of the labels generated differ from that of the Viterbi solution) with a latency of only 2.36 tokens. Of course, 2.36 is only the average lookahead required, so it is possible that the we could actually require a very large amount of lookahead for some cases. However, as is shown in Figure 6, the distribution of number of tokens of lookahead required is peaked around 2, and the percentage of tokens that require a lookahead of more than 10 is less than 0.1%. This distribution was generated by picking a value of $\lambda$ so that the accuracy is very close to the full information case (only 0.1% case decrease in error). Figure 5 compares the two algorithms we have presented in this paper with a fixed-window approach (for various values of window size). As can be seen, the improvement is substantial.

There is however, no clear cut winner between the Online Variable Window algorithm and the Online Step algorithm. The Online Step algorithm achieves a lower error rate in the (relatively) larger latency region, but has higher error rate in the smaller latency region. We believe that the reason for this is as follows. When we allow a reasonably long latency, emiting a single step at a time seems to be the right thing to do because we only make incremental decisions, not comitting to more than one label when we do not have to. Observe when we decode an entire window of length $k$ at a time, the latency for the labels for the first time step is $k$, while the latency for the label for the last time step is 0. Therefore, the Online Window algorithm allows for some states to take much more time while still keeping the average low. This appears to be significant when the allowable latency is small.

## 5. Conclusions

To our knowledge, this work is a first attempt at a principled and effective method for low-latency optimal decoding of sequential input streams. Both the Online Variable Window algorithm and the Online Step algorithm achieve substantially higher performance than the alternatives. Using these algorithms, we can achieve virtually the same accuracy as the

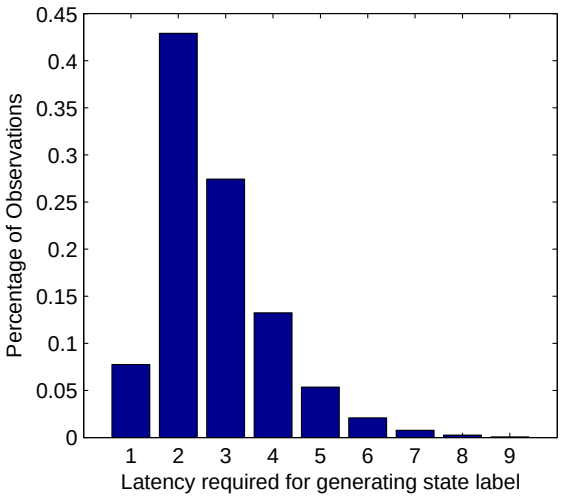


*Figure 6.* This plot shows the number of additional observations required for generating the state label for decoding with 0.1% error for the Online Step algorithm.

Viterbi algorithm, while using only a constant (small) amount of space, regardless of the length of the input. We have also shown formally that these algorithms are optimal under the conditions specified in this paper, and have experimentally verified that the conditions hold for over 98% of the examples in our data set.

## References


Berger, A., Pietra, S. D., & Pietra, V. D. (1996). A maximum entropy approach to natural language processing. *Computational Linguistics*, *22*, 39–71.

Chan, P., & Stolfo, S. (1998). Toward scalable learning with non-uniform class and cost distributions: A case study in credit card fraud detection. *Knowledge Discovery and Data Mining* (pp. 164–168).

Cox, K., Eric, S., Willis, G., & Brachman, R. (1997). Visual data mining: Recognizing telephone call-fraud. *Data Mining and Knowledge Discovery* (pp. 225–31).

Daumé III, H., & Marcu, D. (2005). Learning as search optimization: Approximate large margin methods for structured prediction. *International Conference on Machine Learning (ICML)*. Bonn, Germany.

Dumouchel, W., & Schonlau, M. (1998). A fast computer intrusion detection algorithm based on hypothesis testing of command transition probabilities. *Proceedings of the fourth international conference on Knowledge Discovery and Data Mining*.

Eickeler, S., Kosmala, A., & Rigoll, G. (1998). Hidden markov model based continuous online gesture recognition. *Int. Conference on Pattern Recognition (ICPR)* (pp. 1206–1208). Brisbane.

Fawcett, T., & Provost, F. (1997). Adaptive fraud detection. *Data Mining and Knowledge Discovery* (pp. 291–316).

Kristjansson, T., Culotta, A., Viola, P., & McCallum, A. (2004). Interactive information extraction with constrained conditional random fields. *Proceedings of the 19th AAAI*.

Lafferty, J., McCallum, A., & Pereira, F. (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data. *Proceedings of the 18th ICML*. San Francisco: Morgan Kaufmann.

Lane, T., & Brodley, C. (1997). Approaches to online learning and concept drift for user identification in computer security. *Data Mining and Knowledge Discovery* (pp. 259–263).

Lee, W., Stolfo, S., & Mok, K. (1998). Mining audit data to build intrusion detection models. *International Conference on Knowledge Discovery and Data Mining*.

Leek, T. (1997). Information extraction using hidden markov models. *Masters thesis, UC San Diego*.

McCallum, A., Freitag, D., & Pereira, F. (2000). Maximum entropy markov models for information extraction and segmentation. *Proc. 17th International Conference on Machine Learning*.

Pinto, D., McCallum, A., Wei, X., & Croft, W. (2003). Table extraction using conditional random fields. *Proceedings of the 26th ACM SIGIR* (pp. 235–242).

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

Seward, A. (2003). Low-latency incremental speech transcription in the synface project. *EUROSPEECH*.

Taskar, B., Abeel, P., & Koller, D. (2002). Discriminative probabilistic models for relational data. *Proceedings of the 18th Conference on Uncertainty in Artificial Intelligence*.