Fast Likelihood Search for Hidden Markov Models

Yasuhiro Fujiwara
NTT Cyber Space Labs and University of Tokyo
Yasushi Sakurai
NTT Communication Science Labs
and
Masaru Kitsuregawa
University of Tokyo

Hidden Markov models (HMMs) are receiving considerable attention in various communities and many applications that use HMMs have emerged such as mental task classification, biological analysis, traffic monitoring, and anomaly detection. This paper has two goals; The first goal is exact and efficient identification of the model whose state sequence has the highest likelihood for the given query sequence (more precisely, no HMM that actually has a high-probability path for the given sequence is missed by the algorithm), and the second goal is exact and efficient monitoring of streaming data sequences to find the best model. We propose SPIRAL, a fast search method for HMM datasets. SPIRAL is based on three ideas; (1) it clusters states of models to compute approximate likelihood, (2) it uses several granularities and approximates likelihood values in search processing, and (3) it focuses on just the promising likelihood computations by pruning out low-likelihood state sequences. Experiments verify the effectiveness of SPIRAL and show that it is more than 490 times faster than the naive method.

Categories and Subject Descriptors: H.2.8 [**Database Applications**]: Data Mining

General Terms: Algorithms, Theory

Additional Key Words and Phrases: hidden Markov model, likelihood, upper bound

Portion of this article have appeared in the 2008 ACM conference of Knowledge Discovery and Data Mining (SIGKDD).
Authors' address: Y. Fujiwara (contact author), NTT Cyber Space Laboratories, 1-1 Hikarino-oka, Yokosuka-Shi, Kanagawa, 239-0847 Japan; e-mail: fujiwara.yasuhiro@lab.ntt.co.jp; Y. Sakurai, NTT Communication Science Laboratories, 2-4 Hikaridai, Seika, Soraku, Kyoto, 619-0237 Japan; e-mail: yasushi.sakurai@acm.org; M. Kitsuregawa, Institute of Industrial Science, University of Tokyo, 4-6-1 Komaba, Meguro-ku, Tokyo 153-8505, Japan; e-mail: kitsure@tkl.iis.u-tokyo.ac.jp

1. INTRODUCTION

The hidden Markov model is a ubiquitous tool for representing probability distributions over sequences of observations. Since HMMs, which assess sequential data as sequences of state transitions, are robust against noise, significant applications that use HMMs have emerged, including mental task classification, biological analysis, traffic monitoring, and anomaly detection. This paper has two goals. The first goal is efficient identification of the model whose state sequence has the highest likelihood for the given query sequence, exactly (i.e., an HMM that actually has a high-probability path for the given sequence is *never* missed by the algorithm.). The second goal is efficient monitoring of streaming data sequences to find the best model without exception. Although numerous studies have been published in various research areas [Siddiqi and Moore 2005; Esposito and Radicioni 2007], to the best of our knowledge, this is the first study to address the HMM search problem that guarantees answer exactness.

1.1 Problem Definition

HMMs have found their widest application in problems that have inherent temporality such as speech recognition or gesture recognition. HMM has a number of parameters whose values are set so as to best explain the training patterns for the known category. A given pattern is classified by the model with the highest posterior probability, likelihood, i.e. the one that best explains the given pattern.

Increasing the speed of computing HMM likelihood remains a major goal for the speech recognition community. This is because most of the total processing time (30-70%) in speech recognition is used to compute the likelihoods of continuous density HMMs where each state is modeled by a separate mixture of Gaussian densities [Gales et al. 1999]; the likelihood is computed using the Viterbi algorithm [Rabiner and Juang 1986]. Replacing continuous density HMMs with discrete HMMs is a useful approach to reducing the computation cost [Sagayama et al. 1995]. Unfortunately, the CPU cost still remains excessive, especially for large datasets, since all possible likelihoods are computed.

Therefore, this paper gives a solution by focusing on the following problem:

PROBLEM 1. *Given an HMM set, and a sequence $X = (x_1, x_2, \ldots, x_n)$, find the model whose state sequence has the highest likelihood, estimated with respect to X, from the set of HMMs.*

This problem is designed to handle human-generated queries. The system is a repository storing a large collection of models, and the target of the system is to identify the model that will best match the operator-given query sequence.

The focus on data engineering has recently shifted toward data stream applications [Abadi et al. 2003]. These applications handle continuous streams of input from external sources such as a sensor.

We address the following problem in this paper:

PROBLEM 2. *Given an HMM set, and a subsequence of data stream $X = (x_1, x_2, \ldots, x_n)$ where x_n is the most recent value, monitor incoming sequence X to identify the model whose state sequences has the highest likelihood, estimated with respect to X, from the set of HMMs.*

Problems 1 and 2 focus on finding the best model. However, we can handle range queries (find the models whose likelihoods exceed a given threshold) and K-nearest neighbor queries (find the highest K likelihood models) as described in later sections.

In order to simplify the presentation, in the remainder of the paper, we assume that the models have non-zero likelihood, and that they will never give exactly the same likelihoods. This assumption is made so that there is always just one best model for any given sequence. This assumption can be eliminated without much problem and is not pursued in this paper.

1.2 Problem Motivation

The problems tackled in this paper must be overcome to develop the following important applications.

1.2.1 *Applications of processing human-generated queries.* We show mental task classification and biological analysis as examples of the first problem.

Mental task classification. The Brain Computer Interface (BCI), which is mainly designed to help disabled people control personal computers using biofeedback, is a completely new approach in the field of neurology [G. Pfurtscheller and Neuper 1994]. Biofeedback is a coaching and training process that helps people learn how to change patterns of behavior, to take greater responsibility for their health and for their mental, physical, emotional and spiritual well-being. Since it is undesirable for disabled people to have to adapt to their computers, the basic idea behind BCI is for the computer to adapt rather than the person.

Electroencephalogram (EEG) signals are weak voltages resulting from the spatial summation of electrical potentials in the brain cortex, which can easily be detected by electrodes suitably placed on the scalp. They result from the superposition of three main types of brain potential: oscillatory, event-related, and slow potential shifts. Different components of the EEG signal have been widely demonstrated to have measurable correlations with the brain activity associated with specific mental tasks.

Mental task classification using EEG is an approach to understanding human brain functions. HMM processing is a major tool for EEG since it has the capability to classify probabilistic and statistical signals. In the classification, artifacts, such as body movement and respiration, are removed from the original signals by digital filtering, correlation analysis, or independent component analysis [Novak et al. 2004]. HMMs are prepared, and their parameters are trained using refined data. The HMMs are manually labeled and stored in a database. To classify a query sequence, it is fitted to all trained models, and is classified as belonging to the model with the highest likelihood [Zhong and Ghosh 2002].

Biological analysis. One of the most important contributions of biological sequences to the study of evolution is the discovery that sequences of different organisms are often related. Similar genes are conserved across widely divergent species, often performing a similar or even identical function; some functions are altered through the forces of natural selection [Mount 2001]. Thus, many genes are represented in highly conserved forms over a wide range of organisms. Sequence searches

against large databases have become a mainstay of bioinformatics, and sequencing projects in which the entire genomic DNA sequence of an organism is obtained have become quite commonplace [Durbin et al. 1999]. Search techniques can also be especially useful in determining the function of genes whose sequences have been established in the laboratory but for which there is no biological information. In these searches, the sequence of the gene of interest is compared with every sequence in a sequence database, and similar ones are identified. Alignments with the best-matching sequences are shown and scored. If the query sequence can be readily aligned with a sequence with known function, structure, or biochemical activity, the query sequence is predicted to have similar properties.

The primary advantage of HMMs is that they can be automatically trained using unaligned sequences. Therefore, HMMs have gained increasing acceptance by the computational biology community as a means of sequence modeling, multiple alignment, and profiling [Baldi et al. 1994]. HMMs can also be used to model protein families, or families of other molecular sequences such as DNA and RNA [Haussler et al. 1993]. When modeling proteins, we observe the amino acids in the query sequence of the protein. For all models in the databases, likelihoods are computed with the Viterbi algorithm. The query sequence is assigned to the family of the model that has the highest likelihood among those in the database.

1.2.2 *Applications that monitor data streams.* Traffic monitoring and anomaly detection are key examples of the second problem.

Traffic monitoring. Traffic congestion is an unpleasant fact of modern life. The existence of saturated freeways and congested main roads all over the world, reflects the fact that the existing road networks are unable to cope with the demand for mobility which will only increase in the future. Especially in densely populated regions, it is socially untenable to expand the existing infrastructure in order to handle the situation. On the other hand, mobility is vital for continued economic development.

The existing road network, therefore, has to be used more efficiently by the application of information systems that inform road users about the traffic conditions or provide route guidance. The basic requirement for this service is the precise processing of spatial and temporal data to yield accurate traffic status. Usually the traffic status is measured locally by various detection technologies, mostly inductive loops. In order to provide network-wide information, it is convenient to combine the measured data with a suitable traffic flow model [Helbing et al. 2000]. This data can be processed by information systems whose outputs allow the road users to organize their trips with regard to individual preferences.

Various algorithms have been advanced to explain traffic congestion based on fluid flow, cellular automata, and microscopic computer simulations. Unfortunately, all of these approaches have limitations, particularly the need to tune many parameters to each individual freeway.

HMM is a very cost-effective tool because it can extract the model parameters from actual traffic data and effectively identify traffic status [Bickel et al. 2001; Kwon and Murphy 2000]. Several kinds of information can be extracted from loop detector data such as the number of vehicles passing the location during a given

time interval (flow rate), the fraction of time that vehicles are over the loop detector (occupancy), and the vehicle velocities averaged over the time interval. Model parameters are estimated with this information and traffic status like congestion or free flow, which are defined beforehand. Once models are estimated, the traffic status can be identified with the likelihood for streaming data sequence from a road. For example, if a model estimated from past congested data shows the best likelihood, the road can be labeled as 'traffic jam'.

Anomaly detection. The widespread use of the Internet and computer networks has brought, with all its benefits, another kind of threat: that of people using illicit means to access, invade, and attack computers. Since we have become extremely dependent on the use of information services, the danger of crucial operations being seriously disrupted is frightening. What is worse, it is estimated than less than 4% of these attacks will be detected or reported [Barbará et al. 2001]. Therefore, anomaly detection in computer science is a key problem area because of its importance and the widespread interest in the subject [Denning 1998].

Automated modeling of human behavior is useful in the computer security domain of anomaly detection. In the user modeling facet of the anomaly detection domain, the task is to develop a model or profile of the normal working status of a computer system user and to detect anomalies as deviations from expected behavior patterns. A subset of hostile activities can then be detected through anomalous behaviors. For example, recursively searching system directory hierarchies by hand or browsing through another user's files is unusual behavior for many users and the presence of such activities may be indicative of an intruder who has penetrated the account.

Recently, one feature of the anomaly detection domain is the threat of replay attacks, in which an attacker monitors a system and records information such as user commands; these commands then later are replayed back to the system literally. Because user commands were, in fact, generated by a valid user, it seems perfectly normal to the detection sensors unless some check is made for events that are too similar to past events.

To avoid replay attacks, HMMs can be used to identify users by their command line behavior patterns [Lane 1999; Warrender et al. 1999]. First, command traces were gathered from UNIX users. The command traces were parsed with a recognizer for the shell command language to convert them into a format suitable for scanning by HMMs. Each whitespace-delimited command is considered to be a separate symbol. The feature selection step removes filenames and replaces them with the count of the number of file names occurring in the command line. Removal of filenames reduces the alphabet size by deleting excessively unique symbols and improves recognition accuracy. A model is trained with the observed behavioral patterns of the valid user. The likelihood of the incoming data sequence is evaluated with respect to the best model and sequences judged too sufficiently likely, the likelihood is too high, are labeled as anomalous, i.e. a possible replay attack.

In addition to the applications mentioned above, HMMs have been used in many fields such as scene classification for video analysis [Huang et al. 2005], isolated word recognition for speech processing [Rabiner and Juang 1986], gesture recognition in motion-based image processing and recognition [Eickeler et al. 1998], and

handwritten character recognition in optical character recognition [Hu et al. 1996]. Our proposed method is applicable to all of these areas.

1.3 Contribution

We propose a novel method called SPIRAL that offers fast likelihood searches. In order to reduce search cost, (1) we merge multiple states to compute approximate likelihood, (2) we compute the approximate likelihood with several levels of granularity, and (3) we prune low-likelihood state sequences that will not yield a fruitful model. SPIRAL has the following attractive characteristics based on the above ideas:

—**High-speed searching:** Solutions based on the Viterbi algorithm are prohibitively expensive for large HMM datasets. SPIRAL uses carefully designed approximations to efficiently identify the most likely model.

—**Exactness:** SPIRAL does not sacrifice accuracy; it returns the highest likelihood model without any omission.

—**No restriction on model type:** It achieves a high level of search performance regardless of model type.

In order to achieve high performance and to find the exact answer, SPIRAL first prunes many models with approximate likelihoods at low computation cost. The exact likelihood computations are limited to the minimum necessary, which yields a dramatic reduction in the total search cost. Our experiments compared the proposed method with the method based on the Viterbi algorithm. As expected, the experiments demonstrate the superiority of SPIRAL. Specifically, SPIRAL is more than 490 times faster.

This article is an extended version of the conference paper of Fujiwara et al. [2008]; there are three additions. First, we enhance the search algorithm to better handle given query sequences (Section 4); its effectiveness is shown using an additional experimental dataset (Section 7). We introduce an additional problem for data stream processing (Section 1) and then our solution (Section 5). Finally, we discuss some extensions to SPIRAL when implementing it in real applications (Section 8).

The remainder of this paper is organized as follows. Section 2 describes related work on HMMs and data engineering. Section 3 overviews some of the background of HMMs. Section 4 introduces SPIRAL and shows how it identifies the best model for query sequence. Section 5 explains our stream processing algorithm to support the identification of the best model. Section 6 gives a theoretical analysis of SPIRAL. Section 7 reviews the results of our experiments. In Section 8, we give a discussion on some topics in implementing SPIRAL. Section 9 is a brief conclusion.

2. RELATED WORK

The basic theory of the HMM was published by Baum and his colleagues in the the late 1960's and early 1970's, but it has been well understood and used in the speech recognition field only since the early 1980's [Rabiner and Juang 1986]. Recently, HMMs have been applied in various fields such as pattern recognition and time sequence clustering. The database community has published many studies on

time sequence matching and query processing of uncertain data, which are slightly related to this work. Although numerous studies have been published in various research areas, none of the described techniques meet the conditions listed in Section 1.

2.1 HMM

Computing HMM likelihood in reasonable time remains a major goal for the speech recognition community. Continuous density HMMs typically have 8-64 Gaussian components, and the likelihood of each component must be separately computed, which incurs high CPU cost. Hunt et al. studied a technique based on LDA (Linear Discriminant Analysis) for reducing the number of Gaussian components [Hunt and Lefebvre 1989]. It is well known that Gaussian models are statistically accurate if the input feature vector is near the Gaussian mean. Based on this idea, Bocchieri presented a method that computes the likelihoods of only the Gaussian neighbors, rather than the likelihood of all Gaussians [Bocchieri 1993]. Replacing continuous density HMMs with discrete HMMs is a useful approach to reducing the computation cost, since the likelihoods of a discrete HMM can be computed by looking them up in a scalar quantized probability table [Sagayama et al. 1995]. These techniques can be applied to complement our method. Unfortunately, it still incurs excessive CPU cost, especially for large datasets, since all possible likelihoods are computed.

The Beam search algorithm is a popular approach to reducing the computational expense of exhaustive dynamic programming search techniques such as the Viterbi algorithm and has been employed in many studies [Ney et al. 1992; F. Jelinek 1999]. The basic idea of Beam search is that a path passing through states whose likelihood is much less than the highest one is not be likely to become the best path in a dynamic programming search (Viterbi path in the Viterbi algorithm). Beam search defines a pruning beam width that identifies states that can be disregarded according to their likelihood. It is clear from the naivety of the pruning criterion that this reduction technique has an undesirable property; the best path may be lost.

Attention has been focused on HMMs as a clustering tool for time-series data where the HMMs are used to provide a similarity measure. Smyth et al. were the first to discuss k-clustering of time-series data with HMMs [Smyth 1996]. They proposed a method that automatically detects the number of clusters based on the data distribution as determined from cross-validated likelihoods. Subsequent work by Li et al. focused on the model selection issue (i.e. locating the HMM topology that best represents the data) and on the clustering structure issue (i.e. finding the most likely number of clusters) [Li and Biswas 1999; 2000]. Lae et al. similarly studied the k-clustering problem for sequence datasets [Law and Kwok 2000]. They applied rival-penalized competitive learning to the problem to improve clustering accuracy.

2.2 Databases

Most previous studies have targeted the indexing of time-series databases. Agrawal et al. studied whole sequence matching (similarity searches for equal length sequences) [Agrawal et al. 1993]. Faloutsos et al. and Moon et al. generalized whole

sequence matching to subsequence matching (similarity searches that focus on arbitrary length sequences) [Faloutsos et al. 1994; Moon et al. 2002]. These studies use Euclidean distance as the similarity distance measure.

Even though many similarity functions have been proposed [Agrawal et al. 1995; Das et al. 1997] , DTW (Dynamic Time Warping) is the dominant distance measure; it provides scaling along the time axis. Yi et al. first studied DTW for very large datasets [Yi et al. 1998]. Keogh investigated a search method based on global constraints which limits the duration along the time axis [Keogh 2002]. The method guarantees no false negatives.

The focus of many studies has shifted toward raising the robustness of noisy data search. Actual values are estimated using PDFs (Probability Density Functions). Cheng et al. classified probabilistic queries in two dimensions into multiple types: aggregate/non-aggregate and entity-based/value-based queries. They then studied efficient processing for all types of queries [Cheng et al. 2003]. Probabilistic threshold queries were also investigated by Cheng et al.; probabilities are computed to determine whether they exceed a given threshold [Cheng et al. 2004]. Tao et al. introduced U-tree [Tao et al. 2005] for uncertain data values, and this can be applied to any type of PDF, such as Zipf and Poisson.

The challenges have made researchers re-think many parts of traditional database-management system design in the streaming context, especially with regard to query processing using correlated attributes [Deshpande et al. 2005], scheduling [Babcock et al. 2003], load shedding [Tatbul et al. 2003], and memory requirements [Arasu et al. 2002]. Various architectures for data stream management systems, such as Aurora [Abadi et al. 2003], Stream [Motwani et al. 2003], Telegraph [Chandrasekaran et al. 2003], and Gigascope [Cranor et al. 2003], have been presented. They are slightly related to our work.

3. HIDDEN MARKOV MODEL

HMM have been the mainstay of the statistical modeling techniques used in modern speech recognition systems. Variants of HMMs are still the most widely used technique in that domain, and are generally regarded as the most successful. In this section we explain the basic theory of HMMs.

3.1 Definitions

Unlike the regular Markov model, in which each state corresponds to an observable event, an HMM is used when the observation is a probabilistic function of the state. An HMM is composed of the following probabilities:

—**Initial state probability:** $\pi = \{\pi_i\}$
The probability of the state being u_i $(i = 1, \cdots, m)$ at time $t = 1$.

—**State transition probability:** $a = \{a_{ij}\}$
The probability of the state transiting from state u_i to u_j.

—**Symbol probability:** $b(v) = \{b_i(v)\}$
The probability of symbol v being output from state u_i.

We use the following urn-and-ball example to explain the basic HMM concept.

Symbols	Definitions
x_t	Value of sequence X at time t ($t = 1, \cdots, n$)
n	Sequence length of X
u_i	i-th state of an HMM ($i = 1, \cdots, m$)
m	Number of states
π=$\{\pi_i\}$	Initial state probability of u_i
a=$\{a_{ij}\}$	State transition probability from u_i to u_j
$b(v)$=$\{b_i(v)\}$	Symbol probability of symbol v in state u_i
P	Exact likelihood
$\tilde{P}$	Approximate likelihood

Table I. Definition of main symbols.

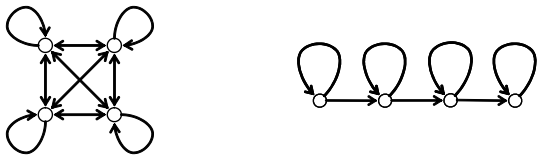
EXAMPLE 1. *Assume there are m urns that represent m states and in each urn there are balls of different colors. Also assume that the observation sequence of length n is created by randomly extracting a ball from a randomly selected urn. There can be multiple combinations of state (urn) sequences that correspond to the same observation sequence (sequence of different ball colors). This is where the "Hidden" concept lies, since the exact state transition sequence corresponding to one observation sequence is unknown. To find one certain state transition sequence, some restrictions need to be applied, such as "the state sequence that has the highest probability". In this example, the probability of extracting a certain ball color from each urn is $b(v)$. The urn selection probabilities are π and a.*

HMMs are classified by the structure of the transition probability a as shown in Figure 1, where the white circles represent states, and the arrows represent transitions. Ergodic HMMs, or fully connected HMMs, have the property whereby every state can be reached from every other state. As shown in Figure 1 (a), the $m = 4$ state model has the property whereby every a_{ij} coefficient is positive. Hence, for Figure 1 (a), we have

$$a = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix}. \quad (1)$$

Left-right HMM is another type of HMM; its state transitions have the property whereby, as time increases, the state number increases or stays the same. The fundamental property of all left-right HMMs is: (1) the state transition probability is $a_{ij} = 0$ for $j < i$ (that is, transitions to lower number states are prohibited). (2) For the initial state probabilities, $\pi_1 = 1$ (i.e., $\pi_i = 0$ for $i \neq 1$) since left-right HMMs always begin with the first state. (3) An additional constraint is that possible transitions are limited to a small number of states. For example, $a_{ij} = 0$ for $j \geq i + 2$ in Figure 1 (b), which means possible transitions are limited to two states. Overall, the state transition probabilities for Figure 1 (b) are given by

$$a = \begin{bmatrix} a_{11} & a_{12} & 0 & 0 \\ 0 & a_{22} & a_{23} & 0 \\ 0 & 0 & a_{33} & a_{34} \\ 0 & 0 & 0 & a_{44} \end{bmatrix}. \quad (2)$$



(a) Ergodic HMM (b) Left-right HMM

Fig. 1. HMM types.

3.2 The Viterbi algorithm

The well-known Viterbi algorithm is a dynamic programming algorithm for estimating the likelihood of sequence X. The maximum probability yielded by a single state sequence corresponds to that likelihood. The state sequence, which gives the likelihood, is called the Viterbi path. For a given model, the likelihood P of X is computed as follows,

$$P = \max_{1 \leq i \leq m} (p_{in}) \tag{3}$$

$$p_{it} = \begin{cases} \max_{1 \leq j \leq m}(p_{j(t-1)} \cdot a_{ji}) \cdot b_i(x_t) & (2 \leq t \leq n) \\ \pi_i \cdot b_i(x_1) & (t = 1). \end{cases}$$

where p_{it} is the maximum probability of state u_i at time t. The likelihood is computed based on the trellis structure shown in Figure 2, where states lie on the vertical axis, and sequences are aligned along the horizontal axis. The likelihood is computed using the dynamic programming approach that maximizes the probabilities from previous states (i.e., each state probability is computed using all previous state probabilities, associated transition probabilities, and symbol probabilities).

EXAMPLE 2. *Assume the following model and sequence.*

$$\pi = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, a = \begin{bmatrix} 0.5 & 0.5 & 0 \\ 0.5 & 0.25 & 0.25 \\ 0 & 0 & 1 \end{bmatrix},$$

$$b(1) = \begin{bmatrix} 1 \\ 0.75 \\ 0 \end{bmatrix}, b(2) = \begin{bmatrix} 0 \\ 0.25 \\ 0 \end{bmatrix}, b(3) = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

$$X = (1, 1, 2, 3).$$

From the Viterbi algorithm, we have

$$\begin{array}{llll} p_{11}{=}1, & p_{12}{=}0.5, & p_{13}{=}0, & p_{14}{=}0 \\ p_{21}{=}0, & p_{22}{=}0.75{\cdot}0.5, & p_{23}{=}(0.5)^2{\cdot}0.25, & p_{24}{=}0 \\ p_{31}{=}0, & p_{32}{=}0, & p_{33}{=}0, & p_{34}{=}(0.5)^2{\cdot}(0.25)^2. \end{array}$$

The state sequence (u_1, u_1, u_2, u_3) *gives the maximum probability. Consequently, we have* $P = (0.5)^2 \cdot (0.25)^2$.

The Viterbi algorithm generally needs $O(nm^2)$ time since it compares m transitions to obtain the maximum probability for every state, that is, it requires $O(m^2)$ in each time tick. The naive solution to identifying the best model for query sequence

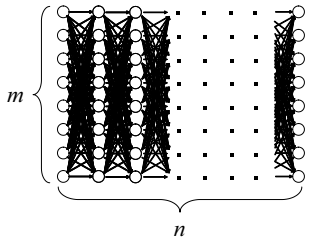


Fig. 2. Trellis structure.

would be to compute the likelihood for every model using the Viterbi algorithm, and then choose the most likely model (i.e., the model that shows the highest likelihood). This incurs excessive CPU time if the trellis structures have large numbers of states. Furthermore, the naive approach to monitoring data stream is to perform this procedure each time a sequence value arrives. However, considering the high frequency with which new values will arrive, more efficient algorithms are needed.

4. FINDING THE BEST MODEL: SPIRAL

In this paper, we mainly tackle two problems: processing human-generated query sequences and monitoring data streams. We start by focusing on search processing for a query sequence, and then discuss how to handle data stream in Section 5.

4.1 Ideas behind SPIRAL

Our solution is based on the three ideas described below.

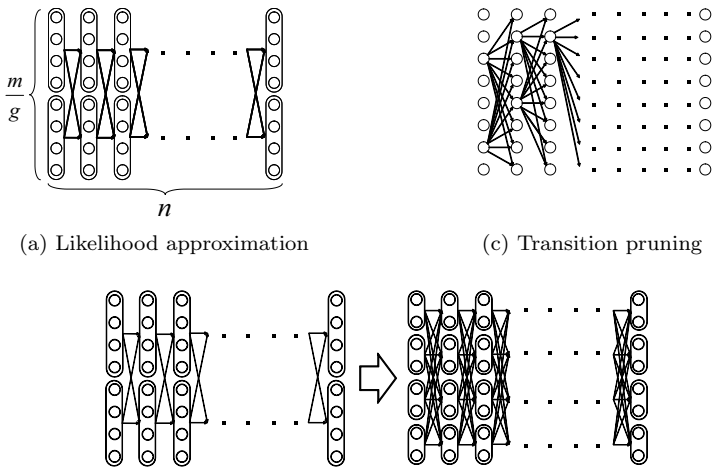
Likelihood approximation. We introduce approximations to reduce the high cost of the Viterbi algorithm solution. Instead of computing the exact likelihood of a model, we approximate the likelihood, thus low likelihood models are efficiently pruned.

The first idea is to reduce the model size. For given m states and granularity g, we create m/g states by merging 'similar' states in the model (See Figure 3 (a)), which requires $O(nm^2/g^2)$ time to obtain the approximate likelihoods instead of the $O(nm^2)$ time demanded by the Viterbi algorithm solution. We use a clustering approach to find groups of similar states, then create a compact model that covers the groups. We refer to it as the *degenerate* model.

This new idea has the following two major advantages. First, we can find a likely model without any omission even though we use approximations. Search omissions are avoided completely by upper bounding the likelihood. This means that we can safely discard unpromising models at low CPU cost.

The second advantage is that this idea does not depend on model type. We can estimate the approximate likelihoods for any model type since we do not use any probability constraints. The choice of model type depends on the user or application.

Multi-granularities. Instead of creating degenerate models at just one granularity, we propose using multiple granularities to optimize the trade-off between accuracy



(a) Likelihood approximation

(c) Transition pruning

(b) Multi-granularities

Fig. 3. Basic ideas behind SPIRAL.

and comparison speed. As the size of a model increases, accuracy improves (i.e., the upper bounding likelihood decreases), but the likelihood computation time increases. Therefore, we generate models at granularity levels that form a geometric progression: $g = 1, 2, 4, \ldots, m$, where $g = 1$ gives the exact likelihood while $g = m$ means the coarsest approximation. We then gradually increase the size of the models (i.e., we use a model with a smaller g), which improves the accuracy of the approximate likelihood, as the search progresses (See Figure 3 (b)).

Low-likelihood models (i.e., unlikely models) are pruned in the coarse-grained approximation stages, whereas fine-grained approximation is needed to more accurately compute high-likelihood models. Therefore, we apply fine-grained approximation only to the models that remain after coarse-grained approximation. Consequently, we can balance the competing goals of accuracy and computation time.

This approach reinforces the first idea by adjusting the granularity of a model according to its exact likelihood. That is, we can identify the best model among a large number of models efficiently since exact likelihood computations are minimized.

Transition pruning. Although our approximation technique is able to discard unlikely models, we still rely on exact likelihood computation to guarantee the correctness of the search results. Here we focus on reducing the cost of this computation.

The Viterbi path shows the state sequence from which the likelihood is computed. Even though the Viterbi algorithm does not compute the complete set of paths, the trellis structure includes an exponential number of paths. Clearly the exhaustive exploration of all paths is not computationally feasible, especially for a large number of states. We therefore ask the question, which paths in the structure are not

promising to explore? This can be answered by using a threshold (say θ).

Our search algorithm that identifies the best model maintains the candidate (i.e., best-so-far) likelihood before reporting the final likelihood. We use θ here as the best-so-far highest likelihood. θ is updated, i.e. increased, when a more promising model is found during search processing. Note, we assume that no two models have exactly the same likelihoods.

We exclude the unlikely paths in the trellis structure by using θ, since θ never decreases during search processing. If the upper bounding likelihood of paths that pass through a state is less than θ, that state cannot be contained in the Viterbi path, and we can safely discard them.

After pruning all viable paths the likelihood computation can be stopped early, and while m states may need to be searched for the next observation, the range of computation is less than that demanded by the Viterbi algorithm which requires $O(m^2)$ times in each time tick.

This technique can be applied to approximate likelihood computation as well as to exact computation. This means that we can compute the approximate likelihood more efficiently.

4.2 Likelihood approximation

Our first idea involves clustering states of the original models and computing upper bounding likelihoods to realize reliable model pruning.

4.2.1 *State clustering.* Attempts to minimize model complexity by aggregating states have been reported in the field of reinforcement learning [Singh et al. 1994]. We reduce the size of the trellis structure by merging similar states in order to compute likelihoods at low computation cost. To achieve this, we adopt a clustering approach. Given granularity g, we try to find m/g clusters from among the m original states. We first describe how to compute the probabilities of a new degenerate model, and then show our clustering method.

We merge all the states in a cluster and create a new state. For the new state, we choose the highest probability among the probabilities of the states to compute the upper bounding likelihood (described in Section 4.2.2). We obtain the probabilities of new state u_c by merging all the states in cluster $\mathcal{C}$ as follows:

$$\begin{aligned}
&\hat{\pi}_c = \max_{u_i \in \mathcal{C}}(\pi_i), \qquad \hat{a}_{cc} = \max_{u_i \in \mathcal{C},\ u_k \in \mathcal{C}}(a_{ik}),\\
&\hat{a}_{cj} = \max_{u_i \in \mathcal{C}}(a_{ij}) \quad \text{for any} \quad u_j \notin \mathcal{C},\\
&\hat{a}_{jc} = \max_{u_i \in \mathcal{C}}(a_{ji}) \quad \text{for any} \quad u_j \notin \mathcal{C},\\
&\hat{b}_c(v) = \max_{u_i \in \mathcal{C}}(b_i(v)).
\end{aligned} \tag{4}$$

We use the following example to explain state clustering.

EXAMPLE 3. *We use the model of Example 2. Let two clusters $\mathcal{C}_1$ and $\mathcal{C}_2$, and the original states u_1 and u_2 be elements of $\mathcal{C}_1$; u_3 is in $\mathcal{C}_2$. We obtain the new*

state probability by taking the maximum value of the original probabilities:

$$\hat{\pi}_1 = max(\pi_1, \pi_2), \quad \hat{a}_{11} = max(a_{11}, a_{12}, a_{21}, a_{22}),$$
$$\hat{a}_{12} = max(a_{13}, a_{23}), \quad \hat{a}_{21} = max(a_{31}, a_{32}),$$
$$\hat{b}_1(1) = max(b_1(1), b_2(1)), \quad \hat{b}_1(2) = max(b_1(2), b_2(2)),$$
$$\hat{b}_1(3) = max(b_1(3), b_2(3)).$$

Thus, we have

$$\hat{\pi} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \hat{a} = \begin{bmatrix} 0.5 & 0.25 \\ 0 & 1 \end{bmatrix},$$
$$\hat{b}(1) = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \hat{b}(2) = \begin{bmatrix} 0.25 \\ 0 \end{bmatrix}, \hat{b}(3) = \begin{bmatrix} 0 \\ 1 \end{bmatrix}.$$

We use the following vector of features F_i to cluster state u_i.

$$F_i = (\pi_i; a_{i1}, \ldots, a_{im}, a_{1i}, \ldots, a_{mi}; b_i(v_1), \ldots, b_i(v_s)). \tag{5}$$

where s is the number of symbols. We choose this vector to reduce approximation error. The highest probabilities are the probabilities of a new state. Therefore, the greater the difference in probabilities possessed by the two states, the greater the difference in the vectors becomes. Thus a good clustering arrangement can be found by using this vector.

In our experiments, we used the well-known k-means method to cluster states [1] where the Euclidean distance is used as a distance measure. However, we can exploit BIRCH [Zhang et al. 1996] instead of the k-means method, the L1 distance as a distance measure, or SVD to reduce the dimensionality of the vector of features. The clustering method is completely independent of SPIRAL, and is beyond the scope of this paper.

4.2.2 *Upper bounding likelihood.* We compute approximate likelihood $\hat{P}$ from degenerate models that have $\hat{m}(= m/g)$ states. Given a degenerate model, we compute its approximate likelihood as follows:

$$\hat{P} = \max_{1 \le c \le \hat{m}} (\hat{p}_{cn}) \tag{6}$$
$$\hat{p}_{ct} = \begin{cases} \max_{1 \le j \le \hat{m}} (\hat{p}_{j(t-1)} \cdot \hat{a}_{jc}) \cdot \hat{b}_c(x_t) & (2 \le t \le n) \\ \hat{\pi}_c \cdot \hat{b}_c(x_1) & (t = 1). \end{cases}$$

where $\hat{p}$ is the maximum probability of states.

THEOREM 4.1. *For any HMM model, the following inequality holds.*

$$P \le \hat{P}. \tag{7}$$

Proof: For each original state u_i $(1 \le i \le m)$, we have

$$p_{i1} \le \hat{\pi}_c \cdot \hat{b}_c(x_1) = \hat{p}_{c1}, \quad (1 \le c \le \hat{m}).$$

If $2 \le t \le n$, we have

$$p_{it} \le \max_{1 \le j \le \hat{m}} (\hat{p}_{j(t-1)} \cdot \hat{a}_{jc}) \cdot \hat{b}_c(x_t) = \hat{p}_{ct}.$$

[1] We repeat the clustering procedure until there are no more changes.

These equations mean that any path in the degenerated trellis structure gives the upper bounding likelihood for the corresponding path in the original trellis structure. The Viterbi path gives the maximum probability yielded by the original trellis structure, this property of the degenerated trellis structure is also true for the Viterbi path. That is,

$$P = \max_{1 \le i \le m} (p_{in}) \le \max_{1 \le c \le \hat{m}} (\hat{p}_{cn}) = \hat{P}.$$

which completes the proof. □

Theorem 4.1 provides SPIRAL with the property of finding the exact answer. We provide a proof of this property in Section 6.

4.3 Multi-granularity

In the previous section, we presented an algorithm that computed the approximate likelihood of a degenerate model with a single level of granularity. However, we can also exploit multiple granularity. Here, we describe the gradual refinement of the likelihood approximation with multiple granularity.

We use $h + 1$ distinct granularities that form a geometric progression $g_i = 2^i$ $(i = 0, 1, 2, \ldots, h)$. We, therefore, generate trellis structures of models that have $\lfloor m/g_i \rfloor$ states. g_h represents the smallest (coarsest) model[2] while g_0 corresponds to the original model, which gives the exact likelihood. g_i becomes geometrically smaller to give larger structures, which improves the approximation accuracy.

In search processing to identify the best model, we first compute the coarsest structure for all models. We then obtain the candidate and the exact likelihood θ. If a model has an approximate likelihood smaller than θ, that model is pruned with no further computation. Otherwise, we compute a finer-grained structure for that model, and check whether the approximate likelihood is smaller than θ. We iterate this check until we reach g_0. For example, if the original HMM has 16 states, SPIRAL computes the likelihoods of models that have $1, 2, 4, 8, 16$ states in this order until the model is pruned. Consequently, we can prune unlikely models with appropriate granularity according to exact likelihood.

Later we describe search algorithms based on these approaches.

4.4 Transition pruning

We introduce an algorithm for computing likelihoods efficiently based on the following lemma:

LEMMA 4.2. *Likelihoods of a state sequence are monotonic non-increasing.*

Proof: In Equations (3) and (6), likelihoods are computed by dynamic programming to maximize the probabilities from previous states. This procedure ensures that the likelihood of a state is less than or equal to the likelihoods of any transited states. □

We exploit the above lemma in pruning paths in the trellis structure. We introduce e_{it}, which indicates a conservative estimate of likelihood p_{it}, to prune unlikely

[2]Note that the coarsest granularity is $g_h = 2^{\lfloor \log_2 m \rfloor}$. The coarsest model has one state.

paths as follows:

$$e_{it} = \begin{cases} p_{it} \cdot (a_{max})^{n-t} \cdot \prod_{j=t+1}^{n} b_{max}(x_j) & (1 \leq t \leq n-1) \\ p_{in} & (t = n) \end{cases} \tag{8}$$

where a_{max} and $b_{max}(v)$ are the maximum values of the state transition probability and symbol probability, respectively:

$$a_{max} = \max_{i,j}(a_{ij}) \qquad (i = 1, \ldots, m; j = 1, \ldots, m) \tag{9}$$

$$b_{max}(v) = \max_{i} b_i(v) \qquad (i = 1, \ldots, m) \tag{10}$$

The estimate is exactly the same as the maximum probability of u_i when $t = n$. Estimate e_{it}, the product of the series of the maximum values of the state transition probability and symbol probability, has the upper bounding property assuming the Viterbi path passes through u_i at time t.

THEOREM 4.3. *For paths that pass through state $u_i (i = 1, \ldots, m)$ at time $t (1 \leq t \leq n)$, the following inequality holds for state $u_j (j = 1, \ldots, m)$ at time n.*

$$p_{jn} \leq e_{it} \tag{11}$$

Proof: If $1 \leq t \leq n-1$, for a state sequence that passes through state u_i at time t, the following equation holds at time $t+1$ for any state $u_j (1 \leq j \leq m)$ from Lemma 4.2:

$$p_{j(t+1)} = p_{it} \cdot a_{ij} \cdot b_j(x_{t+1}) \leq p_{it} \cdot a_{max} \cdot b_{max}(x_{t+1})$$

Similarly, the following equation holds at time $t+2$:

$$p_{j(t+2)} \leq p_{it} \cdot (a_{max})^2 \cdot \prod_{k=t+1}^{t+2} (b_{max}(x_k))$$

Consequently, given a state sequence that passes through state u_i at time t, the following equation holds at time n for any state $u_j (1 \leq j \leq m)$:

$$p_{jn} \leq p_{it} \cdot (a_{max})^{n-t} \cdot \prod_{k=t+1}^{n} b_{max}(x_k) = e_{it}$$

If $t = n$, $p_{in} = e_{in}$. which completes the proof. □

This property enables SPIRAL to search for models exactly, the proof of which is given in Section 6.

In search processing, if e_{it} gives a value smaller than θ (i.e, the best-so-far highest likelihood in the search processing for the best model), state u_i at time t for the model cannot be contained in the Viterbi path. Accordingly, unlikely paths can be pruned with safety. Figure 4 shows the algorithm for the likelihood computation. The algorithm prunes unlikely paths in the trellis structure with θ. We can similarly apply this algorithm to the approximate likelihood computation as well as to the exact computation. Threshold θ is updated when searching the model. We show this algorithm in the next section.

We use two arrays $\mathcal{S}$ and $\mathcal{S}'$ for dynamic programming to keep track of transitions. We also use θ to improve the efficiency. The algorithm initializes the first array,

```
Algorithm Pruning
  input: sequence X, threshold θ
  output: estimate e
  add all states to S';
  for t := 1 to n do
      S := ∅;
      for i := 1 to m do
          compute e_it for X from the transitions of S';
          if e_it ≥ θ then
              add u_i to S;
      end for
      S' := S;
      if S = ∅ then   // terminate the likelihood computation
          return max_{1≤i≤m}(e_it);
  end for
  return max_{1≤i≤m}(e_in)
```

Fig. 4. Likelihood computation algorithm. The algorithm prunes unlikely paths against given threshold θ.

$\mathcal{S}$, every time tick. If the likelihood estimate of u_i at t (i.e., e_{it}) does not exceed θ, the state is not included in $\mathcal{S}$, which means we do not need to take the state into account at $t+1$. If $\mathcal{S}$ is empty, we terminate the likelihood computation since the given model cannot yield a search result. We can optionally compute the state sequence by backtracking to the maximum probability if the user requires it.

Now let us use the following example to explain how to prune transition paths.

EXAMPLE 4. *We use the model of Example 2 and set $\theta = 0.1$. The path through u_2 at $t = 1$ is not promising since $e_{21} = p_{21} \cdot (1^3) \cdot (1 \cdot 0.25 \cdot 1) = 0$ is lower than θ. Therefore, we do not take this path into account when we compute the probabilities at $t = 2$. Similarly, we exclude the path through u_3 at $t = 1$, the path through u_2 at $t = 2$, and the path through u_3 at $t = 2$. At $t = 3$, for all states, the likelihood estimates are lower than θ, so we terminate the likelihood computation and determine that this is not a likely model.*

4.5 Search algorithm

We show our approach to finding the best model for query sequence in this section. Our approach is that we (1) prune low-likelihood models by using the approximate likelihood, which guarantees the exact answer, and then (2) ensure that the model selected can be the answer by exact likelihood computation, while minimizing the number of exact likelihood computations.

SPIRAL exploits the exact likelihood of the candidate model to prune other models. We present the simplest way of finding the candidate in [Fujiwara et al. 2008], which computes likelihoods of the models one by one. That is, approximate likelihoods of a model are computed while gradually increasing the model size, and if the approximate likelihood is smaller than θ, the model is pruned since it cannot be a qualifying model. The exact likelihood is computed only when the approximate likelihood of the finest model is greater than or equal to θ, and if the exact likelihood is larger than θ, the candidate and θ are updated. However, this approach can compute many models of finer granularities. For example, as the

```
Algorithm Searching
  input: query sequence X
  output: the best model M_best
  θ := 0;
  add all models to M;
  for i := h to 0 do
      θ' := 0;
      for each model M ∈ M do
          compute P_i for M;
          if P_i ≥ θ' then   //select candidate
              M_max := M;
              θ' := P_i;
          end if
      end for
      compute P_0 for M_max;
      if P_0 ≥ θ then   //update the candidate
          M_best := M_max;
          θ = P_0;
      end if
      for each model M ∈ M do  //prune models
          if P_i < θ then
              subtract M from M;
  end for
  return M_best;
```

Fig. 5. Algorithm for processing query sequences.

worst-case scenario, if models are checked in increasing order of exact likelihood, the candidate does not give efficient likelihood to prune models, and this approach could not find the best model efficiently.

In this paper, we present a sophisticated search algorithm to overcome the above problem. We compute likelihoods of all models with a fixed granularity to identify the candidate, and then increase the model size after selecting the candidate. More concretely, SPIRAL first computes approximate likelihoods of the coarsest structure for all models, and then chooses the best one (the candidate) in terms of the coarsest approximate likelihood. SPIRAL prunes models with this candidate and computes approximate likelihoods of the second coarsest structure for all remaining models. The new candidate model is selected using the second coarsest approximations. SPIRAL iterates this procedure until it computes the exact likelihoods of the remaining models.

This approach has the effect of finding good candidates efficiently as the model size increases by pruning unpromising models. Since good candidates can be computed with larger models, this procedure can reduce the computation cost for finer granularities by pruning low likelihood models at low granularities; there are fewer finer models to compute.

Figure 5 shows the search algorithm of SPIRAL. In this figure, P_i indicates the likelihood of granularity g_i, and $\mathcal{M}$ represents the set of models. We first compute the approximate likelihoods of g_h for all models, and then choose the best candidate. We obtain the initial value of θ by computing the exact likelihood of the best candidate. If the approximate likelihood is smaller than θ, we prune the model since

it cannot be a qualifying model. We continue to compute approximate likelihoods while gradually enhancing the accuracy. We compute the exact likelihood of the model that gives the maximum approximate likelihood at each level of granularity, and we update the candidate and θ to find the best model efficiently, provided P_0 is larger than θ.

Although we described only a search algorithm that can identify the model that has the highest likelihood, this approach can be applied to range queries and K-nearest neighbor queries. For range queries, we would utilize a search threshold as θ, instead of the best likelihood used in the above search algorithm (i.e., we do not use the candidate). The best K-th likelihood would be utilized instead of the best likelihood for K-nearest neighbor queries.

5. STREAMING ALGORITHM

The previous section explained how our approach can be used to search for the best model relative to a given query sequence; the implicit assumption was that the algorithm handles static data. In this section, we show that SPIRAL can be used to monitor data streams.

5.1 Background

Over the past few years, a great deal of attention in the networking and mobile-computing communities has been directed toward building networks of collections of sensors scattered throughout our environment. Researchers at several universities have embarked on projects to produce small, wireless, battery powered sensors and low level networking protocols [Kahn et al. 1999]. These attempts have brought us close to the vision of ubiquitous computing in which computers and sensors assist in every aspect of our lives. To fully realize this vision, however, it will be necessary to process the data structure in the form it occurs in; i.e. as a stream of data values.

In data stream processing, the time interval of interest is generally called as the *window* and there are three temporal spans for which the values of data stream need to be calculated [Ganti et al. 2000; Gehrke et al. 2001]:

—**Landmark window model:** In this temporal span, data streams are computed based on the values between a specific time point, called the landmark, and the present.

—**Sliding window model:** Given sliding window length, n, and the current time point, the sliding window model would compute the subsequence from the prior $n-1$ time to the current time.

—**Damped window model:** In this model, recent data values are more important than earlier ones. That is, in a damped window model, the weights of data decrease exponentially into the past.

This paper focuses on the sliding window model, which is illustrated in Figure 6, because it is used most often and is the most general model [Zhu and Shasha 2002; Gao and Wang 2005].

We consider the data stream as time-ordered series of tuples (time point, value). Each stream has a new value available at each time interval, e.g. every second. If a stream has no value at a time point, a value would be assigned to that time

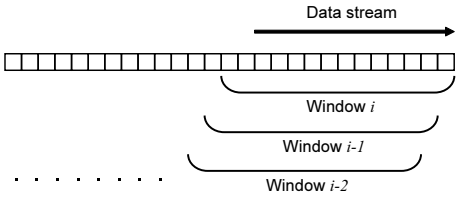


Fig. 6. Sliding window model.

point based on interpolation. If there are several values during a time point, then a summary value would be assigned to that time point, but this is beyond the scope of this work. We assume the most recent sample is always taken at time n. Hence, a streaming sequence takes the form of $(\ldots, x_1, x_2, \ldots, x_n)$. Likelihoods are computed only with n values from the streaming sequence, so we are only interested in subsequences of the streaming sequence from x_1 to x_n. Streaming sequence length must not be shorter than n, but this is not a severe restriction since the streaming sequence is longer than n after being received for some period. Lifting this restriction is not difficult, and is not pursued in this paper.

In the previous section, we assumed that we have a fixed number of HMMs and the operator-specified sequence. In this section, we assume that we have a fixed number of HMMs and a subsequence of the data stream, the problem we deal with here is to find the model which has the highest likelihood for each subsequence extracted from the data stream. The naive method for monitoring data streams is to compute the likelihood of the model with the Viterbi algorithm every time a sequence value arrive. However, this approach is not feasible due to the fact that data streams are likely to have high bit rates.

5.2 Our solution

Our approach for data stream processing mainly follows the approach for static sequences mentioned in Section 4.5. Our search procedure, however, is carefully designed to process high bit rate data streams, which is based on the following observation of data streams:

—There is little difference between subsequences before and after the arrival of a data value.
In the case of a data stream, incoming data values are continually being added to the already received data. The sliding window model is only interest in the subsequence of the latest n values. Therefore, there is little difference in the subsequences before and after the arrival of the latest value, even though new data values will arrive at high frequency.

Our first basic conclusion from this observation is that model granularity can be efficiently decided by referring to the immediately prior granularity. From this observation, it is to be expected that the likelihood of the model examined for the subsequence changes little, and that we can prune the models efficiently by continuing to use the prior granularities. That is, in the present time tick, the initial granularity is set relative to the finest granularity of the previous time tick

at which model likelihood was computed. If model pruning was conducted at the coarsest granularity, we use this granularity in the next time tick, otherwise we use the granularity level that is one step down (coarser) as the initial granularity.

EXAMPLE 5. *If the original HMM has 16 states and the model was pruned with the 1 state model (granularity g_4, coarsest), we adopt the 1 state model (granularity g_4) as the initial model in the next time tick; if the model is pruned using the approximate likelihood of 16 states (granularity g_0) , we select the 8 states model (granularity g_1) as the initial model.*

If the model is not pruned at the initial granularity, the approximate likelihood of a finer-grained structure is computed to check for model pruning against the given θ. This procedure is the same as the algorithm for handling static sequences.

This procedure enables SPIRAL to automatically change the granularities in accordance with the stream trend. That is, if the model likelihoods show a declining trend, the granularity is made coarser; if the likelihoods are rising, SPIRAL computes the likelihoods of finer-grained models.

Our second basic conclusion is to select the best model of the previous time as the initial candidate. The search algorithm for static sequences (which described in Section 4.5) selects the initial candidate based on the approximate likelihood of the coarsest model to find the best model. However, from the observation, the best model of the last time tick is likely to be the best model again. Therefore, we first compute the exact likelihood of the best model of one time tick before to obtain the initial θ needed to identify the best model effectively.

5.3 Search algorithm

Figure 7 depicts our approach to data stream processing. In this figure, $\mathcal{M}_i$ represents the set of models for which we compute the likelihood of granularity g_i, and $\mathcal{M}'_i$ represents the set of models computed with the finest granularity in the previous time tick, g_i. SPIRAL first computes the initial value of θ based on the best model of the last time. And SPIRAL sets the initial granularity. If a model is pruned at the coarsest granularity at the last time tick, it uses this granularity as the initial granularity. Therefore, we add $\mathcal{M}'_h$ to $\mathcal{M}_h$ in Figure 7. The one step lower granularity is used as the initial granularity if the model was not pruned at the coarsest granularity; 'add $\mathcal{M}'_{i-1}$ to $\mathcal{M}_i$' represents this procedure.

This approach is also applicable to range queries and K-nearest neighbor queries against streaming sequences. We can handle range queries by applying a search threshold as θ to the above algorithm. For K-nearest neighbor queries, we can efficiently select the initial candidate to prune models by computing the exact likelihoods of the K models at the previous time tick.

6. THEORETICAL ANALYSIS

In this section we provide a theoretical analysis that confirms the accuracy and complexity of SPIRAL. Note that our theoretical analysis covers both cases; handling query sequences and data streams. Let m be the number of states, n the sequence length, and s the number of symbols.

```
Algorithm Monitoring
  input: subsequence X of stream, set of models M',
         the previous best model M'_best
  output: the best model M_best
  compute P_0 for M'_best;  //set initial candidate
  θ := P_0;
  M_best := M'_best;
  add M'_h to M_h;  //set initial granularity
  for i := h to 1 do
      add M'_{i-1} to M_i;
  end for
  for i := h to 0 do
      θ' := 0;
      for each model M ∈ M_i do
          compute P_i for M;
          if P_i ≥ θ' then  //select candidate
              M_max := M;
              θ' := P_i;
          end if
      end for
      compute P_0 for M_max;
      if P_0 ≥ θ then  //update the candidate
          M_best := M_max;
          θ = P_0;
      end if
      for each model M ∈ M_i do  //compute finer models
          if P_i ≥ θ then
              add M to M_{i-1};
              subtract M from M_i;
          end if
      end for
      M'_i := M_i;
  end for
  M'_best := M_best;
  return M_best;
```

Fig. 7. Algorithm for monitoring data streams.

6.1 Accuracy

We first show the following lemma to show that SPIRAL identifies the best model accurately in this section:

LEMMA 6.1. *The threshold θ is monotonic non-decreasing in the search process of SPIRAL.*

Proof: To find the best model in the search process, we first find a candidate model based on the coarsest approximate likelihood or the best model at the last time tick, and set the initial θ from the model. We maintain the candidate as the best result; when we find a model higher likelihood, its exact likelihood is greater than θ, the candidate is replaced by the new model. This makes θ larger. Therefore, θ keeps increasing (note each model has different likelihood as mentioned in Section 1). □

We can prove that SPIRAL finds the best model accurately (without fail) as follows:

LEMMA 6.2. *SPIRAL guarantees the exact answer when identifying the model whose state sequence has the highest likelihood.*

Proof: Let M_{best} be the best model in the dataset and θ_{max} be the exact likelihood of M_{best} (i.e., θ_{max} is the highest likelihood). Also let P_i be the likelihood of model M for granularity g_i and θ be the best-so-far (highest) likelihood in the search process. From Theorems 4.1 and 4.3, we obtain $P_0 \leq P_i$, for any granularity g_i, for any M. For M_{best}, $\theta_{max} \leq P_i$ holds. In the search process, since θ is monotonic non-decreasing (Lemma 6.1) and $\theta_{max} \geq \theta$, the approximate likelihood of M_{best} is never lower than θ. The algorithm discards M if (and only if) $P_i < \theta$. Therefore, the best model M_{best} cannot be pruned erroneously during the search process. □

6.2 Complexity

We first discuss the complexities of the Viterbi algorithm and then that of SPIRAL.

LEMMA 6.3. *Given a sequence and model, the Viterbi algorithm requires $O(m^2 + ms)$ space and $O(nm^2)$ time to compute the likelihood.*

Proof: The Viterbi algorithm keeps m values for the initial state probability, m^2 values for the state transition probability, and ms values for the symbol probability. Thus, it needs $O(m^2 + ms)$ space. To compute the likelihood, the Viterbi algorithm computes the maximum probability from all m previous states for every state in each time tick. Therefore, it requires $O(nm^2)$ time. □

LEMMA 6.4. *SPIRAL requires $O(m^2 + ms)$ space to compute the likelihood.*

Proof: SPIRAL keeps $m/2^i$ $(i = 0, 1, 2, \ldots, \log m)$ values for the initial state probability for granularity g_i. Since $\sum_{i=0}^{\log m} m/2^i = 2(1-1/2^{\log m})m \approx 2m$, SPIRAL needs $O(m)$ space for the initial state probability. Similarly, SPIRAL requires $O(ms)$ space for the symbol probability and $O(m^2)$ space for the state transition probability. Consequently, the space complexity of SPIRAL is $O(m^2 + ms)$. □

LEMMA 6.5. *SPIRAL requires at least $O(n)$ time and at most $O(nm^2)$ time to compute the likelihood.*

Proof: When the search algorithm uses the coarsest approximation, the likelihood computation requires $O(n)$ time. SPIRAL needs $O(nm^2/4^i)$ $(i = 0, 1, 2, \ldots, \log m)$ time for granularity g_i. Thus, for the worst case scenario when the algorithm uses the trellis structures of all granularities, SPIRAL requires $O(nm^2)$ time since $\sum_{i=0}^{\log m} nm^2/4^i \approx 4/3nm^2$. □

Lemmas 6.3, 6.4 and 6.5 show theoretically that SPIRAL needs the same order of memory space as the Viterbi algorithm, while SPIRAL can be up to m^2 times faster. In practice, the search cost depends on the granularity used by SPIRAL for the likelihood approximation. In the next section, we show the effectiveness of our approach by presenting the results of our extensive experiments.

7. EXPERIMENTAL EVALUATION

We performed experiments to demonstrate SPIRAL's effectiveness. We compared SPIRAL to the Viterbi algorithm. We refer to the Viterbi algorithm implementation as *Viterbi* hereafter.

7.1 Experimental data and environment

We used the following four standard datasets in the experiments.

—*EEG*:
This dataset was taken from a large study that examined the EEG correlates of the genetic predisposition to alcoholism downloaded from the UCI website [3]. It contains measurements from 64 electrodes placed on subjects' scalps that were sampled at 256 Hz (3.9-msec epoch) for 1 second. In our experiments, we quantized EEG values in 1 microvolt steps, resulting in 506 elements. We computed the probabilities of models from a dataset of 6 subjects (co2a0000365, co2a0000368, co2a0000369, co2c0000338, co2c0000339, and co2c0000340), and we extracted queries from a dataset of 2 subjects (co2a0000364 and co2c0000337).

—*Chromosome*:
We used DNA strings of human chromosomes 2, 18, 21, and 22, which were obtained from the well-known NCBI website [4]. These DNA strings are composed of the letters {A,C,G,T,N} where N is *unknown*. We treat N as a different symbol, resulting in a symbol size of 5. In our experiments, a query dataset is obtained from chromosome 2, and models are trained using the rest of the dataset.

—*Traffic*:
This dataset contains loop sensor measurements of the Freeway Performance Measurement System found on the UCI website. This loop sensor dataset was collected in Los Angeles from 10 April 2005 to 1 October 2005 (5 minute count aggregates), and the symbol size is 91. To train the models, we extracted sequences from the sensor measurements from April 10th to September 23th. We similarly extracted query sequences from sensor measurements from September 24th to October 1st.

—*UNIX*:
We exploited the command histories of 8 UNIX computer users at a university over 2 year period downloaded from the UCI website. This data is drawn from tcsh(1) history files. The data was parsed and sanitized to remove filenames, user names, directory structures, web addresses, host names, and other possibly identifying items resulting in a symbol size of 2360. We obtained a query dataset from user0 and models were trained with the rest of the dataset.

The models were trained by the Baum-Welch algorithm [Levinson et al. 1982]. In our experiments, sequence length is 256 and possible transitions of left-right model are restricted only to 2 states, which is typical in many applications.

We evaluated the search performance mainly through wall clock time. All experiments were conducted on a Linux quad 3.33 GHz Intel Xeon server with 32GB of

[3] http://archive.ics.uci.edu/ml/
[4] http://www.ncbi.nlm.nih.gov

main memory. We implemented our algorithms using GCC. Each result reported here is the average of 100 trials.

7.2 Results of identifying the best model for query sequence

We examined the effectiveness of SPIRAL in finding the highest likelihood model for query sequence.

7.2.1 *Search cost.* We assessed the search time needed for SPIRAL and Viterbi. We conducted trials with various numbers of states and models because differences in these numbers are expected to strongly impact the time taken by Viterbi to process HMM datasets.

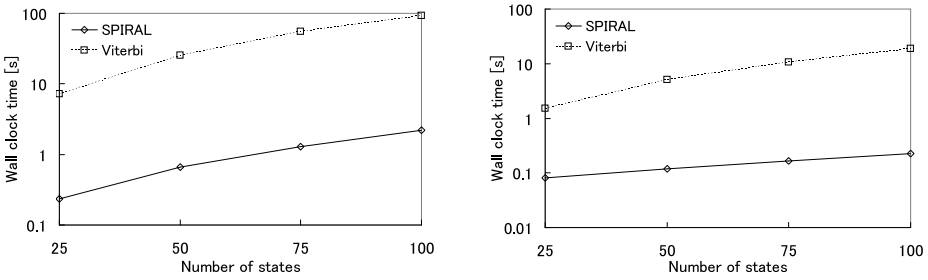
Wall clock time versus number of states. Figure 8 compares SPIRAL and Viterbi in terms of the wall clock time for various numbers of states m for 10,000 models. These figures show that SPIRAL offers greatly increased speed; Viterbi requires $O(nm^2)$ time for computing likelihoods while SPIRAL requires $O(nm^2/g^2)$ for computing approximate likelihoods. SPIRAL requires $O(nm^2)$ time to compute exact likelihoods for models that cannot be pruned through approximation. This cost, however, has no effect on the experimental results. This is because a significant number of models are pruned by approximation. Our method is much faster than the Viterbi algorithm implementation under all the conditions examined. Specifically, SPIRAL is more than 280 times faster for ergodic HMM and more than 80 times faster for left-right HMM.

Wall clock time versus number of models. Figure 9 shows the wall clock time as a function of the number of models, where the number of states is $m = 100$. SPIRAL is superior to the Viterbi algorithm as in the case of changing the number of states. Even if SPIRAL first computes the likelihoods of all models with the coarsest granularity to find the initial candidate, this cost does not alter the search cost since the coarsest approximation requires only $O(n)$ time for a degenerate model which has only one state. SPIRAL exploits the exact likelihood of the candidate model to prune other models, and new candidates are selected based on approximations of finer granularity. This ensures that SPIRAL compute fewer models as model size increases.

7.2.2 *Effect of likelihood approximation with Multi-granularities.* SPIRAL first prunes low-likelihood (unpromising) models using approximations of multiple granularities. The number of exact likelihood computations and fine-grained approximations are factors influencing the search cost. Accordingly, we evaluated the number of exact computations and approximations needed in SPIRAL. Figure 10 shows the number of computations. The number of states and models in this figure is 100 and 10000, respectively.

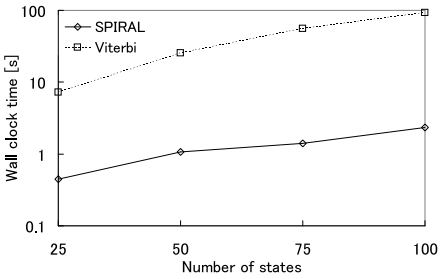
This figure indicates that SPIRAL has strong pruning power; it excludes most of the unlikely models with approximations of $g = 64$, $g = 32$, and $g = 16$. Owing to this approximation quality, SPIRAL achieves excellent search performance as shown in Figures 8 and 9.

This idea is especially effective with models of many states. This is because the more states the models have, the more granularities there are for finding the best model.

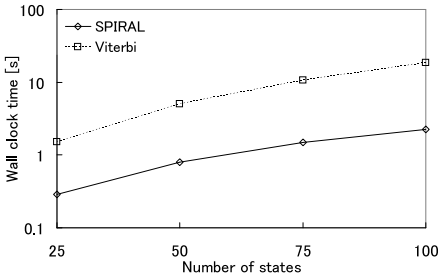


(a-1) *EEG*, ergodic

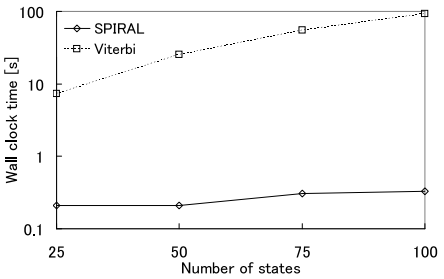
(a-2) *EEG*, left-right



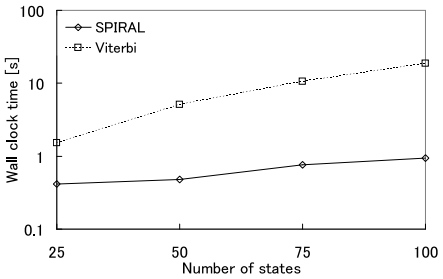
(b-1) *Chromosome*, ergodic



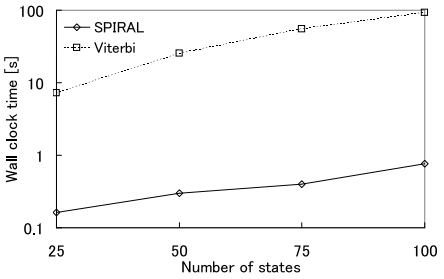
(b-2) *Chromosome*, left-right



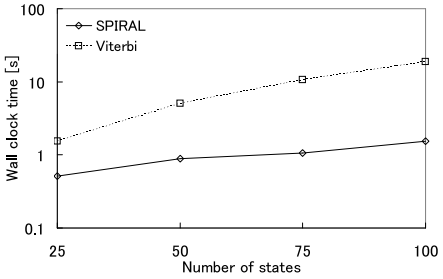
(c-1) *Traffic*, ergodic



(c-2) *Traffic*, left-right

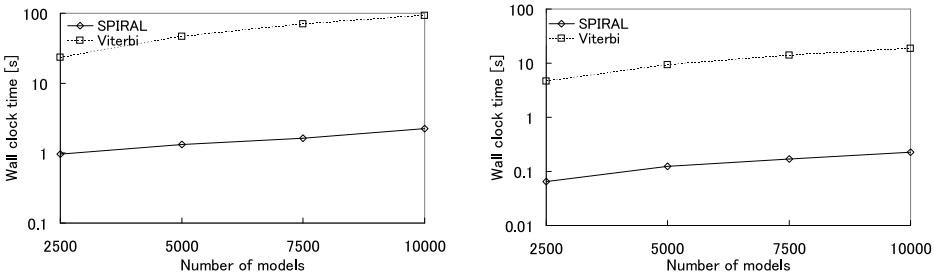


(d-1) *UNIX*, ergodic



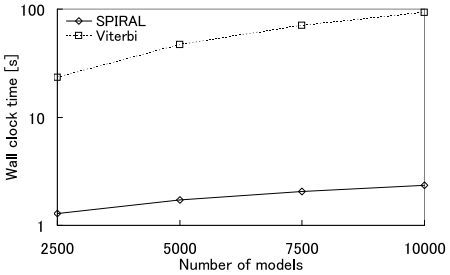
(d-2) *UNIX*, left-right

Fig. 8. Wall clock time versus number of states.

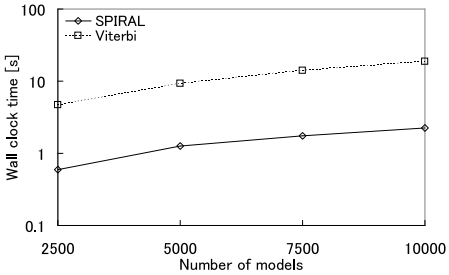


(a-1) *EEG*, ergodic

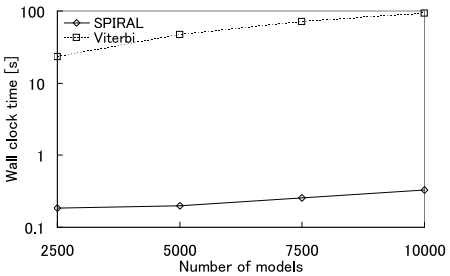
(a-2) *EEG*, left-right



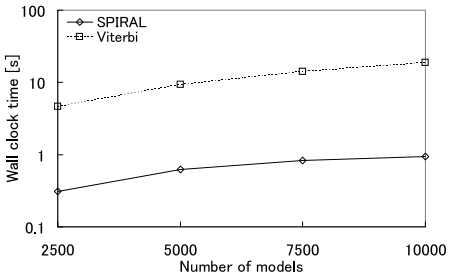
(b-1) *Chromosome*, ergodic



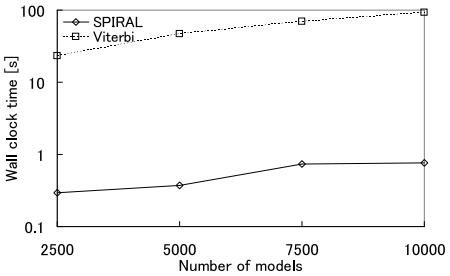
(b-2) *Chromosome*, left-right



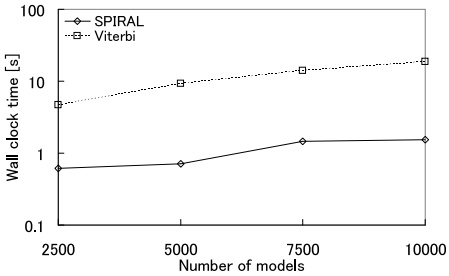
(c-1) *Traffic*, ergodic



(c-2) *Traffic*, left-right

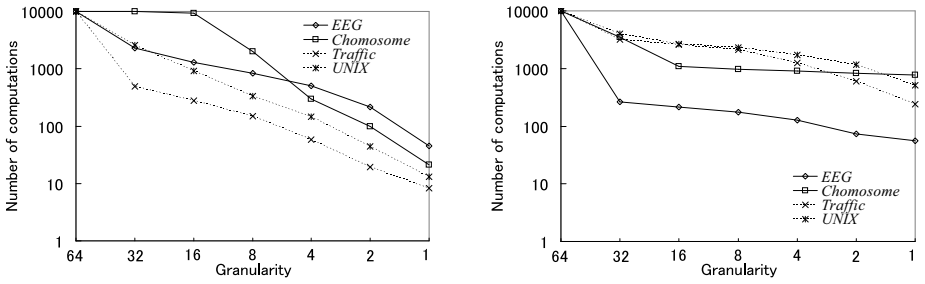


(d-1) *UNIX*, ergodic



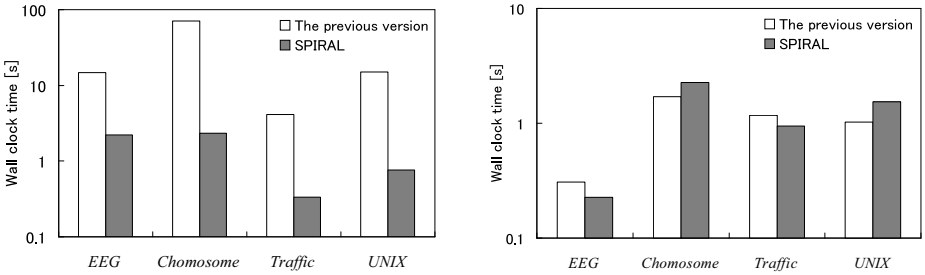
(d-2) *UNIX*, left-right

Fig. 9. Wall clock time versus number of models.



(a) Ergodic (b) Left-right

Fig. 10. Number of likelihood computations.



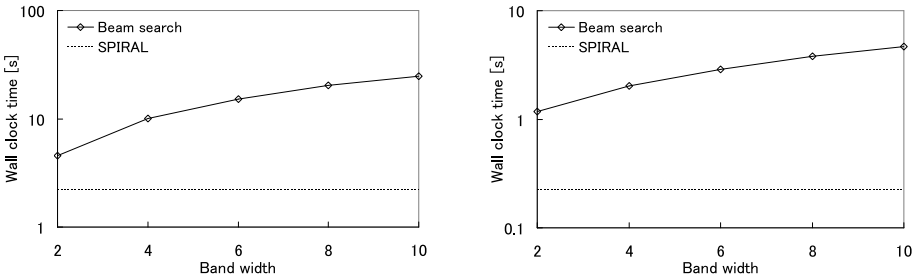
(a) Ergodic (b) Left-right

Fig. 11. Comparison of search approaches.

7.2.3 *Comparison with the previous version.* We previously investigated a search algorithm for HMM datasets [Fujiwara et al. 2008]. As described in Section 4.5, we improve our search algorithm to reduce the number of approximate likelihood computations of fine granularities. We compared the two search algorithms to show the effectiveness of the improved version. Figure 11 depicts the wall clock time for 10,000 models of 100 states. In this figure, SPIRAL represents our new search algorithm.

This figure indicates that our approach is effective, especially for ergodic HMM. For ergodic HMM, it requires $O(nm^2)$ time to compute likelihood, therefore likelihood computation cost increases as the square of model size. Our search approach, by selecting the answer candidate in each granularity, avoids computing the likelihood of finer granularities which incur high computation cost. As a result, our new approach can effectively find the best model for ergodic HMM. The improved algorithm is up to 30 times faster than the previous version.

However, left-right HMMs have transition restrictions; in our experiment the transitions are restricted to 2 states, and it requires $O(nm)$ time to compute likelihood. For this reason there is no visible difference between the search approaches.



(a) Ergodic (b) Left-right

Fig. 12. Wall clock time versus bandwidth.

7.2.4 *SPIRAL vs Beam search.* One major advantage of SPIRAL is that it guarantees the exact answer, but this raises the following simple question: "Can SPIRAL identify models faster than another approach that does not guarantee the exact answer?" To answer the this question, we conducted comparative experiments using the well-known Beam search algorithm. We refer to the Beam search algorithm implementation as *Beam search*.

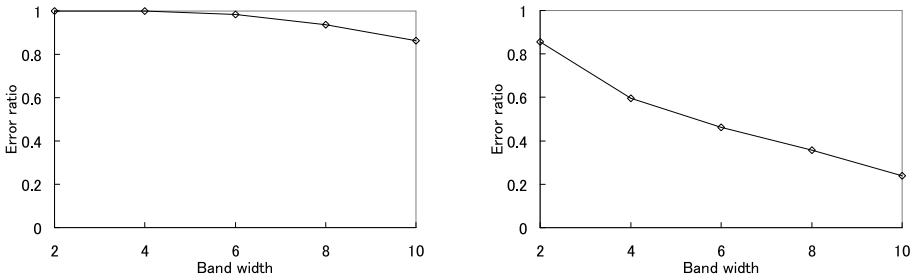
We varied the beam width, i.e. the number of states taken into account, for the Beam search algorithm. Figures 12 and 13 show the wall clock time and the likelihood error ratio, respectively. These figures show the results for 10,000 models of 100 states for *EEG*. Note, SPIRAL identifies the best model accurately, so the likelihood error ratio is 0.

The results show that the Beam search algorithm forces a trade-off between speed and accuracy. That is, as the number of states decreases, the wall clock time decreases but the computation error increases. The Beam search algorithm is an approximation technique and so can miss the best path for the original trellis structure. SPIRAL also computes approximate likelihoods, but unlike the Beam search algorithm, SPIRAL does not discard the best path in each trellis structure, so the errors are 0. Although SPIRAL guarantees the exact answer, it greatly reduces the computation time. Specifically, SPIRAL is up to 20 times faster than the Beam search algorithm in this experiment.

This result implies that SPIRAL will allow HMMs to be applied to many more applications than are currently being considered. While HMM is potentially useful in many applications as described in Section 1, it has been difficult to utilize due to the high computational costs of existing HMM-based techniques. By providing exact solutions in a highly efficient manner, SPIRAL allows HMM to enhanced band so allow larger data structures to be handled, which will improve the accuracy and effectiveness of many applications.

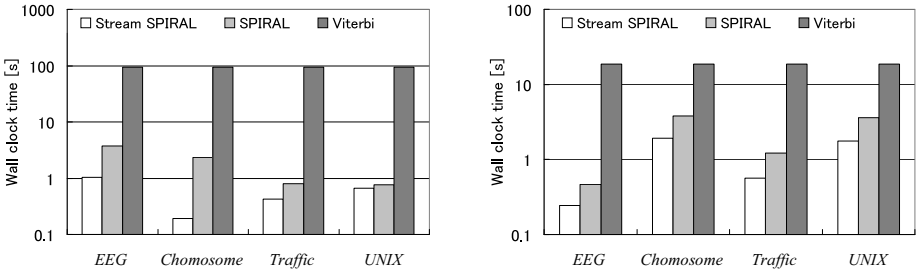
7.3 Results of monitoring data stream

We conducted several experiments to show the effectiveness of our approach for monitoring data stream.



(a) Ergodic (b) Left-right

Fig. 13. Error ratio versus bandwidth.



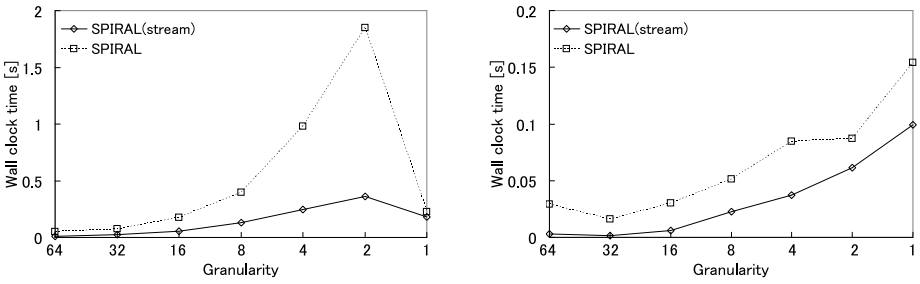
(a) Ergodic (b) Left-right

Fig. 14. Wall clock time of monitoring data stream.

7.3.1 *Search cost.* Figure 14 compares the two versions of SPIRAL (i.e., stream and non-stream algorithms) and Viterbi in terms of the wall clock time for various datasets where the number of states and number of models are 100 and 10,000, respectively.

As expected, the stream algorithm overwhelms the other algorithms, especially the stream algorithm can find the best model up to 490 times faster than the Viterbi algorithm. Our approach for data stream processing follows the approach used to handle static query sequences. However, it differs in setting the initial granularity and candidate, both of which provide the stream algorithm with higher search efficiency. As described in Section 5, we adopt the sliding window model which computes the latest n values of data stream, so the extracted subsequences show little difference before and after the arrival of the next data value. Our approach for data stream processing is based on the this observation, and its effectiveness is confirmed in Figure 14.

7.3.2 *Effectiveness of the data stream algorithm.* Our stream algorithm automatically changes the granularity and effectively sets the initial candidate to find the best model. To show the effectiveness of these ideas, we plot the wall clock time at each granularity for the two versions of SPIRAL. Figures 15 show the breakdown



(a) Ergodic (b) Left-right

Fig. 15. Breakdown of search cost.

in the cost of model search against 10,000 models for *EEG*, where each model has 100 states.

The stream algorithm requires less computation time at each granularity. Instead of using g_h (the coarsest) as the initial granularity for all models, this algorithm sets the initial granularity with the finest granularity at the prior time tick, thus this ensures that the algorithm reduces the number of models at each granularity. Furthermore, the stream algorithm sets the best model of the prior time tick as the initial candidate, which is expected to remain the answer. As a result, it can find the best model for data stream much more efficiently.

8. DISCUSSION

This section discusses the further extension of SPIRAL to support its implementation in real applications.

8.1 Granularity level

SPIRAL identifies the best model by gradually doubling approximate model sizes. This implies that we use the granularity of base 2. However, SPIRAL allows the user to select other base numbers, which would change the memory requirements. Therefore, experiments that examine other base numbers will be extremely useful in designing system architectures for real applications. Table II shows the wall clock time of SPIRAL for three base numbers against 10,000 models where each model has 100 states.

We can see that small base numbers raise the search speed for the ergodic HMM. As discussed in Section 7, the likelihood computation cost of ergodic HMM increases as the square of model size. Therefore, a small base number enables SPIRAL to prune models with low computation cost. However, we can not see this trend for the left-right model. As a result, we can reduce the memory space used by the left-right model by adopting base numbers above 2 while keeping the search efficiency high.

8.2 Clustering approach

As mentioned in Section 4, SPIRAL can exploit arbitrary clustering methods and distance measures. However, the combination adopted can impact the search effi-

Base number	Wall clock time [s]			
	EEG	*Chromosome*	*Traffic*	*UNIX*
2	2.22	2.35	0.33	0.76
4	3.96	3.85	1.65	1.35
8	5.97	12.77	2.25	3.54

(a) Ergodic

Base number	Wall clock time [s]			
	EEG	*Chromosome*	*Traffic*	*UNIX*
2	0.23	2.26	0.94	1.53
4	0.19	1.79	1.18	1.28
8	0.18	2.03	1.16	0.96

(b) Left-right

Table II. Wall clock time versus base number.

Clustering	Wall clock time [s]			
	EEG	*Chromosome*	*Traffic*	*UNIX*
k-means	2.22	2.35	0.33	0.76
PAM	7.66	35.82	5.39	3.58

(a) Ergodic

Clustering	Wall clock time [s]			
	EEG	*Chromosome*	*Traffic*	*UNIX*
k-means	0.23	2.26	0.94	1.53
PAM	0.27	2.13	2.22	1.97

(b) Left-right

Table III. Comparison of clustering method.

ciency since a good clustering approach yields low approximate error. We compared the k-means method used in this paper with the Euclidean distance to PAM with Jensen-Shannon divergence. PAM is the famous clustering method developed by Kaufman and Rousseeuw [Kaufman and Rousseeuw 2005], and Jensen-Shannon divergence is a popular method of measuring the similarity between two probability distributions in probability theory and statistics. Table III shows the results where the number of states is 100 and the number of models is 10,000.

The results show that SPIRAL is greatly impacted by the clustering approach, that is the k-means method basically enables SPIRAL to find the best model more efficiently than PAM. Furthermore, our additional experiments confirmed that PAM incurs high computation cost to construct degenerate structures from large data sets as described in a previous study [Kaufman and Rousseeuw 2005]. Therefore, a user should be careful in selecting the clustering approach for a real application.

8.3 Stream monitoring with dynamically changing models

We have assumed so far the use of static models for the monitoring of data streams even though each data stream is a temporally-variable sequence data. However, in real applications, SPIRAL will need to be modified to adapt to dynamically changing models. Subsequent discussions are given below according to whether the model has already been trained before starting to monitor the data stream or not.

If model has already been trained, it is not difficult to construct degenerate data structures to compute approximate likelihood before monitoring of data streams commences. Moreover, we can index these data structures by simply storing pointers to the structures, since our search algorithms (Figure 5 and 7) do not utilize any search tree structures such as B-trees, all that is needed is to maintain model sets for likelihood computation.

However, in some applications, a user may want to change the model parameters while monitoring a data stream. In this case, on-line learning [Bishop 2007] is effective since the model parameters are updated for one data set at a time. We need to update the degenerate data structures according to the trained model, but we can update these structures at low cost. That is, instead of initializing the positions of cluster centers randomly, which is common in the standard k-means method, the positions of cluster centers before the parameter update are utilized as initial center values. This approach is expected to be effective since the positions of cluster centers are almost the same before and after the update.

9. CONCLUSION

This paper addressed the problem of conducting a likelihood search on a large set of Hidden Markov Models (HMMs) with the goal of finding the best model for a given query sequence and for data streams. We proposed SPIRAL, which is based on three ideas: (1) It prunes low-likelihood models in the HMM dataset by their approximate likelihoods, which yields promising candidates in an efficient manner. (2) It varies the approximation granularity for each model to maintain a balance between computation time and approximation quality. (3) Its transition pruning discards unlikely paths in the trellis structure, which improves the efficiency.

SPIRAL achieves all of the following goals:

—High-speed search: our experiments on real data show that it clearly outperforms the naive implementation, achieving an increase in speed of several orders of magnitude.

—We prove that it guarantees the exact answer.

—It can handle any HMM model type.

Our experiments show that SPIRAL works as expected, and finds high-likelihood HMMs at high speed; Specifically, it is significantly (more than 490 times) faster than the naive implementation.

REFERENCES

Abadi, D. J., Carney, D., Çetintemel, U., Cherniack, M., Convey, C., Lee, S., Stonebraker, M., Tatbul, N., and Zdonik, S. B. 2003. Aurora: a new model and architecture for data stream management. *VLDB J. 12,* 2, 120–139.

Agrawal, R., Faloutsos, C., and Swami, A. N. 1993. Efficient similarity search in sequence databases. In *FODO*. 69–84.

Agrawal, R., Lin, K.-I., Sawhney, H. S., and Shim, K. 1995. Fast similarity search in the presence of noise, scaling, and translation in time-series databases. In *VLDB*. 490–501.

Arasu, A., Babcock, B., Babu, S., McAlister, J., and Widom, J. 2002. Characterizing memory requirements for queries over continuous data streams. In *PODS*. 221–232.

Babcock, B., Babu, S., Datar, M., and Motwani, R. 2003. Chain : Operator scheduling for memory minimization in data stream systems. In *SIGMOD Conference*. 253–264.

Baldi, P., Chauvin, Y., Hunkapiller, T., and McClure, M. A. 1994. Hidden markov models of biological primary sequence information. *Proceedings of the National Academy of Science 91*, 1059–1063.

Barbará, D., Couto, J., Jajodia, S., and Wu, N. 2001. Adam: A testbed for exploring the use of data mining in intrusion detection. *SIGMOD Record 30,* 4, 15–24.

Bickel, P., Chen, C., Kwon, J., Pravin, J. R., and Zwet, V. E. V. 2001. Traffic flow on a freeway network. In *In Workshop on Nonlinear Estimation and Classification*.

Bishop, C. M. 2007. *Pattern Recognition and Machine Learning*. Springer.

Bocchieri, E. 1993. Vector quantization for the efficient computation of continuous density likelihoods. In *ICASSP*. 692–695.

Chandrasekaran, S., Cooper, O., Deshpande, A., Franklin, M. J., Hellerstein, J. M., Hong, W., Krishnamurthy, S., Madden, S., Raman, V., Reiss, F., and Shah, M. A. 2003. Telegraphcq: Continuous dataflow processing for an uncertain world. In *CIDR*.

Cheng, R., Kalashnikov, D. V., and Prabhakar, S. 2003. Evaluating probabilistic queries over imprecise data. In *SIGMOD Conference*. 551–562.

Cheng, R., Xia, Y., Prabhakar, S., Shah, R., and Vitter, J. S. 2004. Efficient indexing methods for probabilistic threshold queries over uncertain data. In *VLDB*. 876–887.

Cranor, C. D., Johnson, T., Spatscheck, O., and Shkapenyuk, V. 2003. Gigascope: A stream database for network applications. In *SIGMOD Conference*. 647–651.

Das, G., Gunopulos, D., and Mannila, H. 1997. Finding similar time series. In *PKDD*. 88–100.

Denning, D. E. 1998. *Cyberspace attacks and countermeasures*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA.

Deshpande, A., Guestrin, C., Hong, W., and Madden, S. 2005. Exploiting correlated attributes in acquisitional query processing. In *ICDE*. 143–154.

Durbin, R., Eddy, S. R., Krogh, A., and Mitchison, G. 1999. *Biological sequence analysis: probabilistic models of proteins and nucleic acids*. Cambridge University Press.

Eickeler, S., Kosmala, A., and Rigoll, G. 1998. Hidden markov model based continuous online gesture recognition. In *ICPR*. 1206–1208.

Esposito, R. and Radicioni, D. P. 2007. Carpediem: an algorithm for the fast evaluation of ssl classifiers. In *ICML*. 257–264.

F. Jelinek. 1999. *Statistical methods for speech recognition.* The MIT Press.

Faloutsos, C., Ranganathan, M., and Manolopoulos, Y. 1994. Fast subsequence matching in time-series databases. In *SIGMOD Conference*. 419–429.

Fujiwara, Y., Sakurai, Y., and Yamamuro, M. 2008. Spiral: efficient and exact model identification for hidden markov models. In *KDD*. 247–255.

G. Pfurtscheller, D. F. and Neuper, C. 1994. Differentiation between finger, toe and tongue movement in man based on 40 hz eeg. *Electroencephalography and Clinical Neurophysiology*, 456–460.

Gales, M., Knill, K., and Young, S. 1999. State-based gaussian selection in large vocabulary continuous speech recognition using hmms. In *TSAP*. 152–161.

Ganti, V., Gehrke, J., and Ramakrishnan, R. 2000. Demon: Mining and monitoring evolving data. In *ICDE*. 439–448.

Gao, L. and Wang, X. S. 2005. Continuous similarity-based queries on streaming time series. *IEEE Trans. Knowl. Data Eng. 17,* 10, 1320–1332.

Gehrke, J., Korn, F., and Srivastava, D. 2001. On computing correlated aggregates over continual data streams. In *SIGMOD Conference*. 13–24.

Haussler, D., Krogh, A., Mian, I. S., and Sjolander, K. 1993. Protein modeling using hidden Markov models: Analysis of globins. In *HICSS 39*. 792–802.

Helbing, D., Herrmann, H. J., Schreckenberg, M., and Wolf, D. E. 2000. *Traffic and Granular Flow '99: Social, Traffic, and Granular Dynamics*. Springer-Verlag.

Hu, J., Brown, M. K., and Turin, W. 1996. Hmm based on-line handwriting recognition. *IEEE Trans. Pattern Anal. Mach. Intell. 18,* 10, 1039–1045.

Huang, J., Liu, Z., and Wang, Y. 2005. Joint scene classification and segmentation based on hidden markov model. *IEEE Transactions on Multimedia 7,* 3, 538–550.

Hunt, M. and Lefebvre, C. 1989. A comparison of several acoustic representations for speech recognition with degraded and undegraded speech. In *ICASSP*. 262–265.

Kahn, J. M., Katz, R. H., and Pister, K. S. J. 1999. Next century challenges: Mobile networking for "smart dust". In *MOBICOM*. 271–278.

Kaufman, L. and Rousseeuw, P. J. 2005. *Finding Groups in Data: An Introduction to Cluster Analysis*. Wiley-Interscience.

Keogh, E. J. 2002. Exact indexing of dynamic time warping. In *VLDB*. 406–417.

Kwon, J. and Murphy, K. 2000. Modeling freeway traffic with coupled hmms. *Tech. Rep., University of California at Berkeley*.

Lane, T. 1999. Hidden markov models for human/computer interface modeling. In *IJCAI-99 Workshop on Learning About Users*. 35–44.

Law, M. H. C. and Kwok, J. T. 2000. Rival penalized competitive learning for model-based sequence clustering. In *ICPR*. 2195–2198.

Levinson, S. E., Rabiner, L. R., and Sondhi, M. M. 1982. An introduction to the application of the theory of probabilistic functions of a markov process to automatic speech recognition. *Bell Syst. Tech. J 62*, 1035–1074.

Li, C. and Biswas, G. 1999. Clustering sequence data using hidden markov model representation. In *SPIE Conference on Data Mining and Knowledge Discovery: Theory, Tools, and Technology*. 14–21.

Li, C. and Biswas, G. 2000. A bayesian approach to temporal data clustering using hidden markov models. In *ICML*. 543–550.

Moon, Y.-S., Whang, K.-Y., and Han, W.-S. 2002. General match: a subsequence matching method in time-series databases based on generalized windows. In *SIGMOD Conference*. 382–393.

Motwani, R., Widom, J., Arasu, A., Babcock, B., Babu, S., Datar, M., Manku, G. S., Olston, C., Rosenstein, J., and Varma, R. 2003. Query processing, approximation, and resource management in a data stream management system. In *CIDR*.

Mount, D. W. 2001. *Bioinformatics: sequence and genome analysis*. Cold Spring Harbor Laboratory Press.

Ney, H., Mergel, D., Noll, A., and Paesler, A. 1992. Data driven search organization for continuous speech recognition. *IEEE Trans. Signal Processing. 40,* 2, 272–281.

Novak, D., T. Al-Ani, Y. H., and Lhotska, L. 2004. Electroencephalogram processing using hidden markov models. In *EUROSIM*.

Rabiner, L. R. and Juang, B. H. 1986. An introduction to hidden markov models. *IEEE ASSP Magazine 3*, 4–16.

Sagayama, S., Knill, K., and Takahashi, S. 1995. On the use of scalar quantization for fast hmm computation. In *ICASSP*. 213–216.

Siddiqi, S. M. and Moore, A. W. 2005. Fast inference and learning in large-state-space hmms. In *ICML*. 800–807.

Singh, S. P., Jaakkola, T., and Jordan, M. I. 1994. Reinforcement learning with soft state aggregation. In *NIPS*. 361–368.

Smyth, P. 1996. Clustering sequences with hidden markov models. In *NIPS*. 648–654.

Tao, Y., Cheng, R., Xiao, X., Ngai, W. K., Kao, B., and Prabhakar, S. 2005. Indexing multi-dimensional uncertain data with arbitrary probability density functions. In *VLDB*. 922–933.

Tatbul, N., Çetintemel, U., Zdonik, S. B., Cherniack, M., and Stonebraker, M. 2003. Load shedding in a data stream manager. In *VLDB*. 309–320.

Warrender, C., Forrest, S., and Pearlmutter, B. A. 1999. Detecting intrusions using system calls: Alternative data models. In *IEEE Symposium on Security and Privacy*. 133–145.

Yi, B.-K., Jagadish, H. V., and Faloutsos, C. 1998. Efficient retrieval of similar time sequences under time warping. In *ICDE*. 201–208.

Zhang, T., Ramakrishnan, R., and Livny, M. 1996. Birch: An efficient data clustering method for very large databases. In *SIGMOD Conference*. 103–114.

Zhong, S. and Ghosh, J. 2002. Hmms and coupled hmms for multi-channel eeg classification. In *IEEE Int. Joint Conf. on Neural Networks*. 1154–1159.

Zhu, Y. and Shasha, D. 2002. Statstream: Statistical monitoring of thousands of data streams in real time. In *VLDB*. 358–369.

Received January 2009