SPIRAL: Efficient and Exact Model Identification for Hidden Markov Models

Yasuhiro Fujiwara
NTT Cyber Space Laboratories
1-1 Hikari-no-oka,
Yokosuka-Shi, Kanagawa,
239-0847 Japan
fujiwara.yasuhiro@lab.ntt.co.jp

Yasushi Sakurai
NTT Communication Science Laboratories
2-4 Hikaridai, Seika-Cho,
"Keihanna Science City",
Kyoto, 619-0237 Japan
yasushi.sakurai@acm.org

Masashi Yamamuro
NTT Cyber Space Laboratories
1-1 Hikari-no-oka,
Yokosuka-Shi, Kanagawa,
239-0847 Japan
yamamuro.masashi@lab.ntt.co.jp

ABSTRACT

Hidden Markov models (HMMs) have received considerable attention in various communities (e.g, speech recognition, neurology and bioinformatic) since many applications that use HMM have emerged. The goal of this work is to identify efficiently and correctly the model in a given dataset that yields the state sequence with the highest likelihood with respect to the query sequence. We propose SPIRAL, a fast search method for HMM datasets. To reduce the search cost, SPIRAL efficiently prunes a significant number of search candidates by applying successive approximations when estimating likelihood. SPIRAL is based on three ideas; (1) it clusters states of models to compute approximate likelihood, (2) it uses several granularities and approximate likelihood values in search processing, and (3) it focuses on just the promising likelihood computations by pruning out low-likelihood state sequences. We perform several experiments to verify the effectiveness of SPIRAL. The results show that SPIRAL is more than 500 times faster than the naive method.

Categories and Subject Descriptors

H.2.8 [**Database Applications**]: Data Mining

General Terms

Algorithms, Theory

Keywords

Hidden markov model, likelihood, upper bound

1. INTRODUCTION

The hidden Markov model is a ubiquitous tool for representing probability distributions over sequences of observations. Since HMMs, which assess sequence data as sequences of state transitions, are robust against noise, significant applications that use HMMs have emerged, such as mental task classification and biological analysis. The goal of this work is efficient identification of the model whose state sequence has the highest likelihood, for the given query sequence, from datasets exactly (i.e., an HMM that actually has a high-probability path for the given sequence is *not* missed by the algorithm.). To the best of our knowledge, this is the first study to address the HMM search problem that guarantees exactness.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
KDD'08, August 24–27, 2008, Las Vegas, Nevada, USA.
Copyright 2008 ACM 978-1-60558-193-4/08/08 ...$5.00.

1.1 Problem motivation

We address the following problem in this paper:

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

Increasing the speed of computing HMM likelihood remains a major goal for the speech recognition community, where the Viterbi algorithm [20] is used to compute the likelihood. This problem is a hurdle to be overcome for the following motivating applications.

1.1.1 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 [9]. 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 functioning. In contrast to biofeedback, it is undesirable for disabled people to have to adapt to computers. The basic idea behind BCI is that the computer adapts 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 surface. 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 correlates with the brain activity involved in specific mental tasks.

Mental task classification using EEG is an approach to understanding human brain functions. The HMM 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

[19]. One HMM is prepared, and its parameters are trained with the refined data and manually labeled. For classification, a query sequence is fitted to all the trained models, and is classified as belonging to the model with the highest likelihood [25].

1.1.2 Biological analysis

One of the most important contributions of biological sequences to evolutionary analysis 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, and with functions altered through the forces of natural selection [17]. Sequence searches for large datasets 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 [5]. Search techniques can also be especially useful for 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 dataset, and similar ones are identified. Alignments with the best-matching sequences are shown and scored. If a query sequence can be readily aligned with a dataset sequence of a 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 estimated, or trained, from unaligned sequences. Therefore, since their introduction to the computational biology community, HMMs have gained increasing acceptance as a means of sequence modeling, multiple alignment, and profiling [3]. HMMs can also be used to model protein families, or families of other molecular sequences such as DNA and RNA [10]. When modeling proteins, we observe the amino acid in the query sequence of the protein. For all models in the dataset, likelihoods are computed with the Viterbi algorithm. If a model has the highest likelihood in the dataset, the query sequence is assigned to the family of the model.

In addition to the applications mentioned above, HMMs have been used in many fields such as car traffic modeling in road traffic engineering [14], anomaly detection in computer science [15], scene classification for video analysis [12], isolated word recognizer in speech recognition [20], gesture recognition in motion-based image processing and recognition [6], and handwritten character recognition in optical character recognition [11]. Our proposed method is applicable to all of these areas.

1.2 Contribution

We propose a novel method called SPIRAL for a fast likelihood search. In order to reduce search cost, (1) we merge multiple states to shrink the trellis structure, (2) we compute the approximate likelihood with several granularities, 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:** The solution based on the Viterbi algorithm is 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 best model without any omissions.
- **No restriction on model type:** It achieves a high level of search performance for any type of model.

To achieve both high performance and output exactness, 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 500 times faster.

The remainder of this paper is organized as follows. Section 2 describes related work on HMMs and data engineering. Section 3 introduces SPIRAL. Section 4 reviews the results of our experiments. Section 5 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 has been well understood and used in the speech recognition field only since the early 1980's [20]. Although numerous studies have been published in various research areas [22, 7], none of the described techniques meet the conditions listed in Section 1.

Computing the likelihoods for HMMs in a 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 a high CPU cost. Hunt et al. studied a technique for reducing the number of Gaussian components by using LDA (Linear Discriminant Analysis) [13]. 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 [4]. 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 [21]. But unfortunately, it still incurs excessive CPU cost, especially for large datasets, since it computes all possible likelihoods.

The Beam search algorithm is a popular approach to reducing the computational expense of an exhaustive dynamic programming search such as the Viterbi algorithm and has been employed in many studies [18, 8]. The basic idea of the Beam search is that a path passing through states whose likelihood is much less than the highest one would not be likely to become the best path in a dynamic programming search (Viterbi path in the Viterbi algorithm). The Beam search defines a pruning beam width that sets states to be disregarded according as their likelihood. It is clear from the naivety of the pruning criterion that this reduction technique has the undesirable property of possibly causing the loss of the best path.

3. PROPOSED METHOD

In this paper, we mainly focus on a query designed to identify the model that has the highest likelihood accurately from HMM datasets. But SPIRAL can also efficiently support range queries and K-nearest neighbor queries as described in section 3.6.

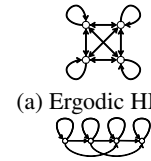
3.1 Hidden Markov Model

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 where the state is u_i $(i = 1, \cdots, m)$ at time $t = 1$.
- **State transition probability:** $a = \{a_{ij}\}$
  The probability where the state transits from state u_i to u_j.

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
P'	Approximate likelihood

Table 1: Definition of main symbols.



(a) Ergodic HMM

(b) Left-right HMM

Figure 1: HMM types.

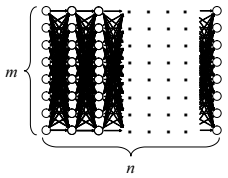


Figure 2: Trellis structure.

- **Symbol probability:** $b(v)$=$\{b_i(v)\}$
  The probability where symbol v is output in state u_i.

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

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 an urn. There can be multiple combinations of state (urn) sequence 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 probabilities a as shown in Figure 1, where the white circles represent states, and the arrows represent transitions. Ergodic HMMs, or fully connected HMMs, have a property whereby every state can be reached from every other state. As shown in Figure 1 (a), for an $m = 4$ state model, this model type has a property whereby every a_{ij} coefficient is positive. Hence, for Figure 1 (a), we have

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

The left-right HMM is another type of HMM where the state transitions have a 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 the 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 $j > i + 2$ in Figure 1 (b), which means possible transitions are limited to three states. Overall, the state transition probabilities for Figure 1 (b) are given by

$$a = \begin{bmatrix} a_{11} & a_{12} & a_{13} & 0 \\ 0 & a_{22} & a_{23} & a_{24} \\ 0 & 0 & a_{33} & a_{34} \\ 0 & 0 & 0 & a_{44} \end{bmatrix}. \quad (2)$$

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,

$$P = \max_{1 \le i \le m} (p_{in}) \quad (3)$$

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

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 a dynamic programming approach that maximizes the probabilities from previous states (i.e., each state probability is computed using all previous state probabilities, associating 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 $P = (0.5)^3 \cdot (0.25)^2$.

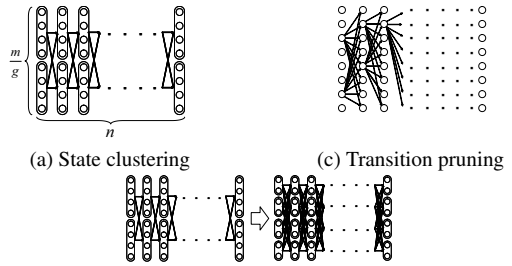
The Viterbi algorithm generally needs $O(nm^2)$ time since it compares m transitions to obtain the maximum probability for every state, that is, it requires $O(m^2)$ in each time tick. The naive solution would be to compute the likelihood for every model using the Viterbi algorithm, and then choose the most likely model (i.e., the model that shows the highest likelihood). This would incur excessive CPU time due to (1) the large size of datasets and (2) the large number of states in trellis structures.

3.2 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 every model, we approximate the likelihood, thus efficiently pruning out low likelihood models.

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



(a) State clustering (c) Transition pruning

(b) Multi-granularities

Figure 3: Basic ideas behind SPIRAL.

obtain the approximate likelihoods instead of $O(nm^2)$ time, which the Viterbi algorithm solution requires. We use a clustering approach to find groups of similar states, then create compact models. We refer to them as *degenerate* models.

This new idea has the following two major advantages. First, we can find likely models without any omissions even though we use approximations. Search omissions are avoided completely by the use of the upper bounding likelihood. This means that we can safely discard unpromising models along with their approximate likelihoods 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 operating on the degenerate model of a single granularity, we propose using multiple granularities to optimize the trade-off between accuracy and comparison speed for the datasets. As the size of the models increases, accuracy improves (i.e., the upper bounding likelihood decreases), but the likelihood computation time increases. Therefore, we generate models of multiple granularities 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 by coarse-grained approximation, whereas fine-grained approximation is needed to evaluate 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 for all the models in the datasets. This approach reinforces the first idea by adjusting the granularity of each model according to its exact likelihood, and we can identify the best model for large number of models since the exact likelihood computations are limited to the minimum number necessary with this idea.

Transition pruning. Although our approximation technique is able to discard most 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 that gives the likelihood. Even though the Viterbi algorithm does not compute the complete set of paths, the trellis structure includes an exponential number of paths. Thus, the exhaustive exploration of all paths is not computationally feasible, especially for large model sets. 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 maintains a candidate (i.e., best-so-far) likelihood before reporting the final likelihood. We use θ as the best-so-far highest likelihood. θ is updated and increases when a promising model is found during search processing. 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, we can safely discard the paths.

We can prune off all viable paths in which case likelihood computaion can be stopepd early, and while m states may need search for the next observation, the range of computation is less than the Viterbi algorithm which require $O(m^2)$ times in each time tick.

From the monotonically increasing property of θ, we can search for the most likely model efficiently over a long sequence. This is an attractive characteristic considering that the sequence could be very long given the user or application requirements.

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.

3.3 Likelihood approximation

3.3.1 State clustering

Attempts to minimize model complexity by aggregating states have been reported in the field of reinforcement learning [23]. 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 for the m original states. We first describe how to compute the probabilities of a new degenerate model, and 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 3.3.2). We obtain the probabilities of a new state u_c by merging all the states in a cluster $\mathcal{C}$ as follows:

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

We use the following example to explain state clustering.

EXAMPLE 3. *Assume that we have the same model as 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:*

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

Thus, we have

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

We use the following vector of features with clustering F_i of state u_i.

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

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. And the greater the number of difference probabilities possessed by the two models, the greater the vector becomes. We can find a good clustering arrangement 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. But we can exploit BIRCH [24] 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 beyond the scope of this paper.

3.3.2 Upper bounding likelihood

We compute approximate likelihood P' from degenerate models that have $m'(= m/g)$ states. Given a degenerate model, we compute the approximate likelihood as follows:

$$P' = \max_{1 \le c \le m'} (p'_{cn}) \tag{5}$$

$$p'_{ct} = \begin{cases} \max_{1 \le j \le m'} (p'_{j(t-1)} \cdot a'_{jc}) \cdot b'_c(x_t) & (2 \le t \le n) \\ \pi'_c \cdot b'_c(x_1) & (t = 1). \end{cases}$$

where p' is the maximum probability of states.

THEOREM 1. *For any HMM model, the following inequality holds.*

$$P \le P'. \tag{6}$$

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

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

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

$$p_{it} \le \max_{1 \le j \le m'} (p'_{j(t-1)} \cdot a'_{jc}) \cdot b'_c(x_t) = p'_{ct}.$$

Thus,

$$P \le \max_{1 \le c \le m'} (p'_{cn}) = P'.$$

□

Theorem 1 provides SPIRAL with the property of exactness. We provide the proof of this property in Section 3.7.

3.4 Multi-granularities

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

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 a model 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 size structures, and consequently, the approximation accuracy improves.

[1] We repeated the clustering procedure until there were no more changes.

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

In search processing, 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 series until the model is pruned. Consequently, we can prune unlikely models with appropriate granularity according to likelihood. Later we describe a search algorithm based on this approach.

3.5 Transition pruning

We introduce an algorithm for computing likelihoods efficiently. We utilize two important properties.

- Likelihoods are monotone non-increasing.
  The likelihood of a state is less than or equal to the likelihoods of any transited states. This property comes from Equations (3) and (5).
- The threshold θ is monotone non-decreasing.
  In search processing, we first finds a model based on the approximate likelihood, and set the initial θ from the model. We maintain the candidate before reporting the final result. When we find a high-likelihood model whose exact likelihood is greater than θ, the candidate is updated; this makes θ larger. Therefore, we realize the second property by insisting that θ keeps increasing or remains unchanged.

We exploit the above two properties in effectively pruning paths in the trellis structure. We introduce e_{it}, which indicates a conservative estimate of the likelihood p_{it}, to prune unlikely paths as follows:

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

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

$$a_{max} = \max_{i,j}(a_{ij}) \quad (i = 1, \ldots, m; j = 1, \ldots, m) \tag{7}$$

$$b_{max}(v) = \max_i b_i(v) \tag{8}$$

The estimate is exactly the same as the maximum probability of u_i when $t = n$. The estimate e_{it}, exploiting the product of a 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_j at time t.

THEOREM 2. *For paths that pass through state $u_i (i = 1, \ldots, m)$ at time $t (1 \le t \le n)$, the following inequality holds state $u_j (j = 1, \ldots, m)$ at time n.*

$$p_{jn} \le e_{it} \tag{9}$$

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

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

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

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

```
Algorithm TransitionPruning
  add all models to S_2;
  for t := 1 to n do
    S_1 := ∅;
    for i := 1 to m do
      compute e_it from the transitions of S_2;
      if e_it ≥ θ then
        add u_i to S_1;
    end for
    // terminate the likelihood computation
    if S_1 = ∅ then
      return max_{1≤i≤m}(e_it);
    S_2 := S_1;
  end for
  return max_{1≤i≤m}(e_in)
```

Figure 4: Likelihood computation algorithm. The algorithm prunes unlikely paths against a given threshold θ.

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

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

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

This property enables SPIRAL to search the model exactly, which provides the proof in Section 3.7.

In search processing, if e_{it} gives a value smaller than θ (i.e, the best-so-far highest likelihood in the search processing), 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 the algorithm in the next section.

We use two arrays $\mathcal{S}_1$ and $\mathcal{S}_2$ for dynamic programming to keep track of transitions. We also use θ to improve the efficiency. The algorithm initializes the first array, $\mathcal{S}_1$, 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 $\mathcal{S}_1$, which means we do not need to take the state into account at $t + 1$. If $\mathcal{S}_1$ 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. *Assume that we have the same model as Example 2 and $\theta = 0.25$. The path through u_2 at $t = 1$ is not promising since $e_{21} = p_{21} \cdot (1^3) \cdot (1 \cdot 0.75 \cdot 1) = 0$ 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 $t = 1$ 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.*

3.6 Search algorithm

Our approach to achieving both high performance and output exactness is to (1) prune low-likelihood models using their approximate likelihoods, which guarantees exactness, and then (2) ensure the candidate model with the exact likelihood computations, which are limited to the minimum number necessary.

```
Algorithm Search
  θ := 0;
  // find one model as the initial candidate
  for each model M ∈ Dataset do
    compute P_h for M;
    if P_h ≥ θ then
      θ := P_h;
      M_best := M;
    end if
  end for
  // compute the initial value of θ
  compute P_0 for M_best;
  θ := P_0;
  // search for the highest likelihood model
  for each M ∈ Dataset do
    for i := h − 1 to 0 do
      compute P_i for M with TransitionPruning;
      if P_i < θ then
        break;
      else if i = 0 then
        M_best := M;
        θ := P_0;
      end if
    end for
  end for
  return M_best;
```

Figure 5: Search algorithm of SPIRAL.

SPIRAL first finds a candidate for the high-likelihood model based on the upper bounding likelihood computed with the coarsest granularity. This is because the approximation quality is improved if θ is close to the final likelihood of the search result early in the search process. The first model to be found is regarded as the initial candidate for the search from which we obtain the initial θ.

Figure 5 shows the search algorithm of SPIRAL. In this figure, P_i indicates the likelihood of granularity g_i. We first compute the approximate likelihoods of g_h for all models, and then choose the best candidate in terms of approximate likelihood. We obtain the initial value of θ by computing the exact likelihood of the candidate. We continue to compute approximate likelihood values while gradually enhancing the accuracy. If the approximate likelihood is smaller than θ, we prune the model since it cannot be a qualifying model as the search result. The exact likelihood P_0 is computed only when the approximate likelihood of g_1 is greater than or equal to θ. If P_0 is larger than θ, we update the candidate and θ, which makes the search more efficient.

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

3.7 Theoretical Analysis

In this section we provide a theoretical analysis to show the accuracy and complexity of SPIRAL. Let m be the number of states, n be the sequence length, and s be the number of symbols.

3.7.1 Accuracy

LEMMA 1. *SPIRAL guarantees exactness when identifying the model whose state sequence has the highest likelihood for the given query sequence.*

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 a model M for granularity g_i and θ be the best-so-far highest likelihood in the search processing.

From Theorems 1 and 2, 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 processing, since $\theta_{max} \geq \theta$, the approximate likelihood of M_{best} is not lower than θ. The algorithm discards M if (and only if) $P_i < \theta$. Therefore, the best model M_{best} cannot be pruned in the search processing. □

3.7.2 Complexity

LEMMA 2. *Given a query 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 the m previous states for every state in each time tick. Therefore, it requires $O(nm^2)$ time. □

LEMMA 3. *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 4. *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 2, 3 and 4 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 SPIRAL uses for the likelihood approximation. In the next section, we show the effectiveness of our approach by presenting results from our extensive experiments.

4. EXPERIMENTAL EVALUATION

We performed experiments to demonstrate SPIRAL's effectiveness. We compared SPIRAL with the Viterbi algorithm and the Beam search algorithm. We refer to the Viterbi algorithm implementation as *Viterbi* and the Beam search algorithm implementation as *Beam*. The models were trained by the Baum-Welch algorithm [16].

We evaluated the search performance based mainly on wall clock time. All experiments were conducted on a 1.66GHz Intel Core 2 PC with 2GB of main memory using the same codebase. Each result reported here is the average of 250 trials. We used the following three standard datasets for 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 [1]. It contains measurements from 64 electrodes placed on subjects' scalps that were sampled at 256 Hz (3.9-msec epoch) for 1 second, that is, the length of each sequence is 256. In our experiments, we quantized EEG values every 10 microvolts, resulting in 50 elements. We computed the probabilities of models from a dataset of subjects(co2a0000365, co2a0000368, co2a0000369, co2c0000338, co2c0000339, and co2c0000340), and we extracted query from dataset(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 [2]. We collected data sequences with consecutive non-overlapping windows of length 256 from each chromosome. 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 with the rest of the dataset.

- *Traffic*:
  This dataset is loop sensor measurements of the Freeway Performance Measurement System found on the UCI website [1]. 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. We extracted sequences with lengths of 256 from sensor measurements from April 10th to September 23th with 21 hours of consecutive overlapping windows to train the models. We similarly extracted query sequences from sensor measurements from September 24th to October 1st.

We omitted the results of the left-right HMM due to space limitations, but additional experiments confirmed the effectiveness of SPIRAL for the left-right HMM.

4.1 Search cost

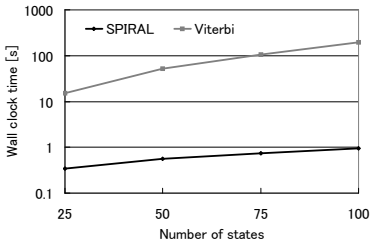
We assessed the search time needed for SPIRAL and Viterbi since none of the previously published studies considered the likelihood search problem for HMMs accurately. We conducted trials with various numbers of states and models because differences in these numbers are expected to have a strong impact on the computation time for HMM datasets with Viterbi.

4.1.1 Wall clock time versus number of states

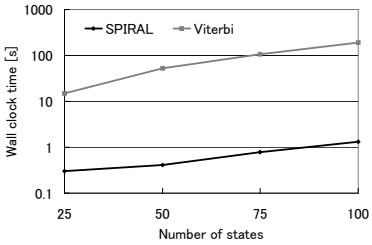
Figure 6 compares SPIRAL and Viterbi in terms of the wall clock time for various numbers of states m for 10,000 models. These figures show that SPIRAL offers greatly increased speed Viterbi requires $O(nm^2)$ time for computing likelihoods while SPIRAL requires $O(nm^2/g^2)$ for computing approximate 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. We describe this in detail in Section 4.2. Our method is several orders of magnitude faster than the Viterbi algorithm implementation under all the conditions we examined. Specifically, SPIRAL is more than 500 times faster.

4.1.2 Wall clock time versus number of models

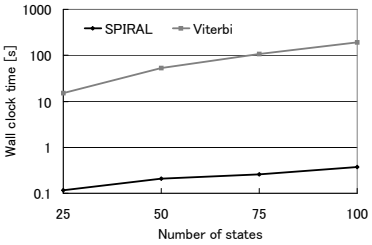
Figure 7 shows the wall clock time as a function of the number of models for *EEG*, where the number of states is $m = 100$. We omitted the results for *Chromosome* and *Traffic* due to space limitations. SPIRAL is so effective because it first computes the likelihoods of



(a) *EEG*



(b) *Chromosome*



(c) *Traffic*

Figure 6: Wall clock time versus number of states.

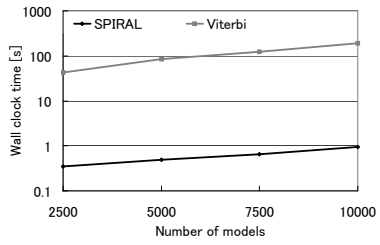


Figure 7: Wall clock time versus number of models.

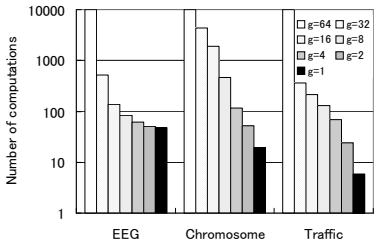


Figure 8: Number of likelihood computations versus granularity.

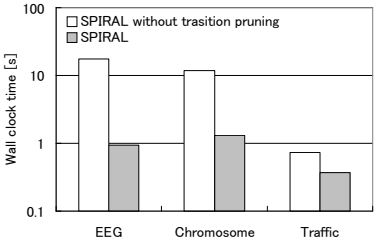


Figure 9: Wall clock time of transition pruning method.

all the models with the coarsest granularity to find the final likelihood θ, which is exploited in pruning, relatively early in the search process.

4.2 Effect of likelihood approximation

SPIRAL first prunes low-likelihood (unpromising) models using multiple granularities of approximation. The number of exact likelihood computations (and also the number of fine-grained approximations) are the dominant factors in the search cost. Accordingly, we evaluated the number of exact computations (and approximations) needed in SPIRAL. Figure 8 shows the number of approximate and exact likelihood computations for 10,000 models. The number of states is $m = 100$ in this figure.

This figure indicates that SPIRAL has strong pruning power; it excludes most of the unlikely models with approximations of $g = 64$, $g = 32$, and $g = 16$, and reduces the number of candidate models with those of the subsequent granularities. Consequently, there a very few exact computations. Owing to this high approximation quality, SPIRAL achieves excellent search performance as shown in Figures 6 and 7.

4.3 Effect of transition pruning

As mentioned in Section 3.5, we exclude unlikely paths from the trellis structure to efficiently compute exact and approximate likelihoods. To show the effectiveness of this idea, we removed the path-pruning technique from SPIRAL, and examined the wall clock time. Figure 9 shows the result for 10,000 models of 100.

The results show that the transition pruning method can provide an efficient search especially for the ergodic HMMs; SPIRAL is up to 19 times faster when we use the transition pruning method.

4.4 SPIRAL vs Beam search

One major advantage of SPIRAL is that it guarantees exactness. But this may trigger the following simple question: "Can SPIRAL identify models faster than another approach that does not guarantee exactness?" To answer the this question, we conducted comparative experiments with the well-known Beam search algorithm.

We varied the beam width for the Beam search algorithm. Figures 10 and 11 show the wall clock time and the likelihood error ratio, respectively. These figures show results for 10,000 models of 100 states for *EEG*.

The results show that there is a trade-off between speed and accuracy in the Beam search algorithm . That is, as the number of states in a bandwidth decreases, the wall clock time decreases but the computation error increases. The Beam search algorithm is an approximation technique that 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 exactness, it greatly reduces the computation time. Specifically, SPIRAL is up to 27 times faster than the Beam search algorithm in this experiment.

5. CONCLUSION

This paper addressed the problem of conducting a likelihood search on a large set of Hidden Markov Models (HMMs). We proposed SPIRAL, which is based on three ideas: (1) we prune low-likelihood models in the HMM dataset by their approximate likelihoods, which yields promising candidates in an efficient manner. (2) We vary the approximation granularity for each model to maintain a balance between computation time and approximation quality. (3) With transition pruning, we discard 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.

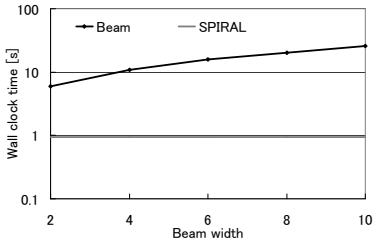


Figure 10: Wall clock time versus bandwidth.

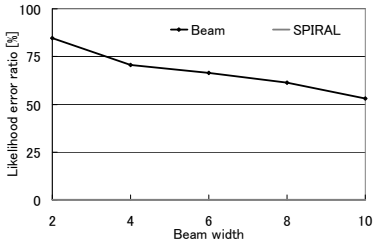


Figure 11: Likelihood error ratio versus bandwidth.

- We prove that it guarantees exactness.
- It can handle any HMM model type.

Our experiments show that SPIRAL works as expected, and finds high-likelihood HMMs with high speed; Specifically, it is significantly (more than 500 times) faster than the naive implementation.

6. REFERENCES

[1] http://archive.ics.uci.edu/ml/.

[2] http://www.ncbi.nlm.nih.gov.

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

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

[5] R. Durbin, S. R. Eddy, A. Krogh, and G. Mitchison. *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*. Cambridge University Press, 1999.

[6] S. Eickeler, A. Kosmala, and G. Rigoll. Hidden Markov Model Based Continuous Online Gesture Recognition. In *ICPR*, pages 1206–1208, 1998.

[7] R. Esposito and D. P. Radicioni. CarpeDiem: an algorithm for the fast evaluation of SSL classifiers. In *ICML*, pages 257–264, 2007.

[8] F. Jelinek. *Statistical Methods for Speech Recognition.* The MIT Press, 1999.

[9] G. Pfurtscheller, D. Flotzinger, and C. Neuper. Differentiation between finger, toe and tongue movement in man based on 40 Hz EEG. *Electroencephalography and Clinical Neurophysiology*, pages 456–460, 1994.

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

[11] J. Hu, M. K. Brown, and W. Turin. HMM Based On-Line Handwriting Recognition. *IEEE Trans. Pattern Anal. Mach. Intell.*, 18(10):1039–1045, 1996.

[12] J. Huang, Z. Liu, and Y. Wang. Joint scene classification and segmentation based on hidden Markov model. *IEEE Transactions on Multimedia*, 7(3):538–550, 2005.

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

[14] J. Kwon and K. Murphy. Modeling Freeway Traffic with Coupled HMMs. *Tech. Rep., University of California at Berkeley*, 2000.

[15] T. Lane. Hidden Markov Models for Human/Computer Interface Modeling. In *IJCAI-99 Workshop on Learning About Users*, pages 35–44, 1999.

[16] S. E. Levinson, L. R. Rabiner, and M. M. Sondhi. 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, 1982.

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

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

[19] D. Novak, Y. H. T. Al-Ani, and L. Lhotska. Electroencephalogram processing using Hidden Markov Models. In *EUROSIM*, 2004.

[20] L. R. Rabiner and B. H. Juang. An introduction to hidden Markov models. *IEEE ASSP Magazine*, 3:4–16, 1986.

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

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

[23] S. P. Singh, T. Jaakkola, and M. I. Jordan. Reinforcement Learning with Soft State Aggregation. In *NIPS*, pages 361–368, 1994.

[24] T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH: An Efficient Data Clustering Method for Very Large Databases. In *SIGMOD Conference*, pages 103–114, 1996.

[25] S. Zhong and J. Ghosh. HMMs and Coupled HMMs for multi-channel EEG classification. In *IEEE Int. Joint Conf. on Neural Networks*, pages 1154–1159, 2002.