Efficient Sequence Data Analysis with Hidden Markov Models
(隠れマルコフモデルによる高速な系列データの解析手法)

指導教員　喜連川　優
東京大学生産技術研究所

藤原　靖宏
東京大学大学院
情報理工学系研究科
電子情報学専攻

2011 年 6 月 15 日提出

Abstract

Sequential data analysis, a relatively young and interdisciplinary field of computer science, is the process of extracting patterns, rules, or meaning from large sequence data sets by combining methods from statistics and artificial intelligence with database management.

With recent tremendous technical advances in processing power, storage capacity, and Internet, sequential data analysis is seen as an increasingly important approach by modern business. This is because it can give an informational advantage by transforming unprecedented quantities of sequence data into business intelligence. It is currently used in a wide range of profiling practices, such as marketing, surveillance, fraud detection, and scientific discovery. Since sequential data analysis can bring real value for real applications, demand for novel technologies in sequential data analysis is growing these days.

Hidden Markov model (HMM) is a popular tool for sequential data analysis, and is receiving considerable attention in various communities. And many applications that use HMM have emerged such as sequence labeling, speech recognition, mental task classification, biological analysis, traffic monitoring, and anomaly detection. The Viterbi algorithm is used in these applications. However, the Viterbi algorithm unfortunately requires quadratic CPU time for the number of states which are used in HMMs. And data in these applications are recently explosively increasing. Therefore efficient classification approach is needed in HMM.

To enhance the processing speed in HMM, many approximation

approaches have been proposed [Ney 92, F. Jelinek 99, Tsuruoka 05, Toutanova 03, Siddiqi 05b, Cohn 06, Jeong 09]. However, these approaches have two major problems. First, they cannot guarantee error bound. Therefore these approaches can affect the qualities of real applications. Second, approximate algorithms usually require hyperparameters, which control the tradeoff between accuracy and efficiency, and have to be manually adjusted for each task.

The proposed approaches in this thesis can efficiently process HMM without sacrificing the computational results; the proposed approaches output the results as exactly same as the Viterbi algorithm. And the proposed approaches need the same memory cost as the Viterbi algorithm; they can handle large size of dataset used in real application. This thesis mainly handles three typical problems in HMM; the first problem is exact and efficient states sequence detection for single HMM and static sequence of of arbitrary length, and the second problem is exact and efficient identification of the model whose state sequence has the highest likelihood for the given query sequence, and the third problem is exact and efficient monitoring of streaming data sequences to find the best model. I propose Staggered decoding for the first problem and SPIRAL for the second and third problem, a fast search method for HMM datasets. The proposed approaches are based on two ideas; approximation and pruning. Approximation is an idea that aggregates several state to discard unlikely states or models. And pruning is an idea that computes exact likelihood of viable states/models by pruning unlikely state transition. Experiments verify the effectiveness of Staggered decoding and SPIRAL. That is, they are much faster than the naive Viterbi algorithm based method. And, to show the generality of our approach for other graphical data structure, I applied our approach for the problem to monitoring best centrality nodes of time-evolving graphs.

Acknowledgments

I wish to express my sincere gratitude to my supervisor, Prof. Masaru Kitsuregawa at the university of Tokyo, for his continuous guidance, valuable discussion and advice.

I also would like to express grateful thank to Assoc. Prof. Masashi Toyoda, Assoc. Prof. Miyuki Nakano, Dr. Nobuhiro Kaji, and Dr. Naoki Yoshinaga for their kind advices, helpful assistances, technical supports, countless hours of useful discussions on the directions, and many issues regarding to my researches. Also, sincere thanks to my Ph. D. committee professors, Prof. Jun Adachi, Prof. Minoru Ishizuka, Prof. Kaoru Sezaki, and Prof. Tohru Asami.

Finally, I want to express my gratitude to my family; my parents Sadataka Fujiwara, Tokuko Fujiwara, my sister Mayumi Suzuki, my wife Hitomi Fujiwara, and my daughter Momoka Fujiwara. They gave me sincere support and encouragement.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Sequential data analysis, a relatively young and interdisciplinary field of computer science, is the process of extracting patterns from large sequence data sets by combining methods from statistics and artificial intelligence with database management.

With recent tremendous technical advances in processing power, storage capacity, and Internet, sequential data analysis is seen as an increasingly important approach by modern business to transform unprecedented quantities of sequence data into business intelligence giving an informational advantage. It is currently used in a wide range of profiling practices, such as marketing, surveillance, fraud detection, and scientific discovery. Since sequential data analysis can bring real value for real applications, demand for novel technologies in sequential data analysis is growing these day.

The Hidden Markov Model (HMM) is a ubiquitous tool for representing probability distributions over sequences of observations. The basic theory of HMM was developed in the late 1960s [Baum 66]. Since HMMs assess sequential data as sequences of state transitions, HMMs are robust against noise. Therefore, significant applications that use HMMs have emerged, including sequence labeling, speech recognition, mental task classification, biological analysis, traffic monitoring, and anomaly detection where data in these applications are recently explo-

sively increasing. For example:

- Google processes 20 peta bite web text pages a day to enhance their search engine quality [Dean 08] where the number of labels* (i.e., states) is typically over $2,000$ for supertagging [Brants 00, Matsuzaki 07].

- Over 790 millions genes are stored in DNA Data Bank of Japan [Sugawara 08]. These gene data are utilized in biological analysis such as predicting function, structure, or biochemical activity of unknown gene data.

- Over 2,000 GPS are attached to vehicles in Sweden for traffic monitoring†. The traffic monitoring system can reduce city traffic and, as a result, CO_2 emissions.

The Viterbi algorithm is used in these applications. But the Viterbi algorithm unfortunately requires quadratic CPU time for the number of states which are used in HMMs. Therefore, the naive Viterbi algorithm implementation is impractical for real applications due to their large size data. So efficient approach is needed in HMM.

To enhance the processing time, many approximation approaches have been proposed [Ney 92, F. Jelinek 99, Tsuruoka 05, Toutanova 03, Siddiqi 05b, Cohn 06, Jeong 09]. However, these approaches have two problems. The first problem is that they cannot guarantee error bound. Therefore these approaches can affect the qualities of real applications. And these approaches need hyperparameters which can affect accuracy and efficiency; this is the second problem. Hyperparameters have to be manually adjusted for each task.

In this thesis, I proposed the efficient approaches which guarantee the same result as the Viterbi algorithm. That is, the proposed approaches can enhance the processing speed of real applications without sacrificing the application qualities. And space complexity of the

*Labels are parts of a noun phrases, verb phrases, etc.

†http://www-03.ibm.com/press/us/en/pressrelease/28463.wss

proposed approaches are same as the Viterbi algorithm; the proposed approaches can handle large size of dataset used in real application. Furthermore, the proposed approaches do not need any parameter setting for uses. In this thesis, I handle three types of HMM problems; the first problem is exact and efficient states sequence detection for single HMM and static sequence of of arbitrary length, and the second problem 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 third problem is efficient monitoring of streaming data sequences to find the best model without exception. And, to show the generality of the proposed approaches for other data structure, I applied the proposed approaches for the problem to monitoring best centrality nodes of time-evolving graphs. Although numerous studies have been published in various research areas [Siddiqi 05a, Esposito 07], to the best of my knowledge, this is the first study to address the HMM search problem that guarantees answer exactness. While HMM has been used in many applications, it has been difficult to utilize the Viterbi algorithm due to its high computational cost. However, by providing solutions in a highly efficient manner, the proposed approaches will allow many more HMM-based applications to be developed in the future.

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 99]; the likelihood is computed using the Viterbi algorithm [Rabiner 86]. Replacing continuous density HMMs with discrete HMMs is a useful approach to reducing the computation cost [Sagayama 95]. Unfortunately, the CPU cost still remains excessive, especially for large datasets, since all possible likelihoods are computed.

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

Problem 1. *Given single HMM, and single sequence* $X = (x_1, x_2, \ldots, x_n)$*, find the state sequence which has the highest likelihood, estimated with respect to* X*.*

This problem setting is traditional one in area of sequence labeling and speech recognition.

Unlike the above traditional problem setting, recently many applications are proposed in which HMM set is used instead of single HMM such as mental task classification, biological analysis. Therefore, this thesis gives a solution by focusing on the following problem:

Problem 2. *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 03]. These applications handle continuous streams of input from external sources such as a sensor.

I address the following problem in this thesis:

Problem 3. *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 2 and 3 focus on finding the best model. However, the proposed approaches 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 thesis, it is assumed 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 thesis.

1.2 Problem Motivation

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

1.2.1 Applications of Problem 1

Sequence labeling and speech recognition are typical application for Problem 1.

Sequence labeling In the past decade, structured learning has been an active research area in the machine learning community. In particular, sequence labeling algorithms such as CRFs and structured perceptrons [Lafferty 01, Collins 02] have been extensively studied because of their importance in a wide range of application domains including natural language processing (NLP) [Manning 99] and bioinformatics

[Mount 04]. Now it is not too much to say that the sequence labeling algorithms serve as a theoretical foundation for those applications.

Most sequence labeling algorithms are probabilistic in nature, relying on statistical inference to find the best sequence. The most common statistical models in use for sequence labeling make a Markov assumption, i.e. that the choice of label for a particular word is directly dependent only on the immediately adjacent labels; hence the set of labels forms a Markov chain. This leads naturally to HMM, one of the most common statistical models used for sequence labeling. In this application, the state sequence which give likelihood is computed for target sentence. And answer is the token sequence which corresponds to the the state sequence.

Speech recognition Speech recognition research in the 1980s was characterized by a shift in methodology from the more intuitive template-based approach (a straightforward pattern recognition paradigm) toward a more rigorous statistical modeling framework. Although the basic idea of the HMM was known and understood, the methodology was not complete until the mid-1980s and it was not until after widespread publication of its theory that the HMM became the preferred approach for speech recognition. The popularity and use of the HMM as the main foundation for automatic speech recognition and understanding systems has remained constant over the past two decades, especially because of its steady stream of improvements and refinements.

An isolated phoneme recognizer can be built with HMM [Rabiner 86, Rabiner 85]. First, an HMM is built for training pronunciations. The observed acoustic features (e.g. Mel frequency cepstral coefficient) from a set of token words are used to estimate the optimum parameters of the model. For unknown pronunciation, characterized by observed features, the likelihood of the unknown pronunciation and it corresponding state sequence are computed. The answer is the phoneme sequence which corresponds to the state sequence.

1.2.2 Applications of Problem 2

I show mental task classification and biological analysis as examples of the second 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 94]. 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 04]. 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 02].

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 01]. 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 99]. 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 94]. HMMs can also be used to model protein families, or families of other molecular sequences such as DNA and RNA [Haussler 93]. When modeling proteins, the amino acids are observed 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.

1.2.3 Applications of Problem 3

Traffic monitoring and anomaly detection are key examples of the third 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 00]. 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 01, Kwon 00]. 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 01]. Therefore, anomaly detection in computer science is a key problem area because of its importance and the widespread interest in the subject [Denning 98].

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 99, Warrender 99]. 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 05], gesture recognition in motion-based image processing and recognition [Eickeler 98], and handwritten character recognition in optical character recognition [Hu 96]. The proposed method is applicable to all of these areas.

1.3 Contribution

I propose Staggered decoding for the first problem and SPIRAL for the second and third problem, a fast search method for HMM datasets. The proposed approach is based on two ideas; approximation and pruning. Approximation is an idea that aggregates several states to discard unlikely states or models. And pruning is an idea that computes exact likelihood of viable states/models by pruning unlikely state transition.

Even though the proposed approach requires precomputations for approximation to aggregate several states by using a clustering method, the proposed approach has the following attractive characteristics based on the above ideas in the search process:

- **High-speed:** Solutions based on the Viterbi algorithm are prohibitively expensive for large HMM data. The proposed approach uses carefully designed approximations to efficiently identify the most likely model.
- **Exactness:** The proposed approach does not sacrifice accuracy; it returns the highest likelihood model without any omission.
- **Applicability:** The proposed approach can be applied not only for HMM but other data structures such as time-evolving graphs.

In order to achieve high performance and to find the exact answer, The proposed approach first prunes many states/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. Experiments compared the proposed method with the method based on the Viterbi algorithm. As expected, the experiments demonstrate the superiority of the proposed approach.

1.4 Organization of the Thesis

The remainder of this thesis is organized as follows.

- In Chapter 2, I describes the detail introduction for HMM and related work on HMM in data engineering.
- Chapter 3 explains Staggered Decoding which can efficiently compute likelihood and and its corresponding state sequence.
- Chapter 4 introduces SPIRAL and shows how it identifies the best model for query sequence.

- Chapter 5 explains the proposed stream processing algorithm to support the identification of the best model.
- In Chapter 6, I extend the proposed approach for the problem to monitoring best centrality nodes of time-evolving graphs.
- Chapter 7 is a conclusion.

Chapter 2

Preliminary and Related Work

In this chapter, I first explain the detail of HMM and then describe its related work.

2.1 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 I explain the basic theory of HMMs. The main symbols in this thesis are shown in Table 2.1.

2.1.1 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 $t = 1$.

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
$\hat{P}$	Approximate likelihood

Table 2.1: Definition of main symbols.

- **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.

The following urn-and-ball example explains 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 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 2.1, 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 2.1 (a), the $m = 4$ state model has the property whereby every a_{ij} coefficient is positive. Hence, for Figure 2.1 (a),

$$a = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix}. \tag{2.1}$$

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 2.1 (b), which means possible transitions are limited to two states. Overall, the state transition probabilities for Figure 2.1 (b) are given by

$$a = \begin{bmatrix} a_{11} & a_{12} & 0 & 0 \\ 0 & a_{22} & a_{23} & 0 \\ 0 & 0 & a_{33} & a_{34} \\ 0 & 0 & 0 & a_{44} \end{bmatrix}. \tag{2.2}$$

2.1.2 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

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

Figure 2.1: HMM types.

state sequence, which gives the likelihood, is called the Viterbi path. For a given model, the likelihood P of X is computed as follows,

$$P = \max_{1 \le i \le m} (p_{in}) \tag{2.3}$$

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

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 2.2, 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.*

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

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

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

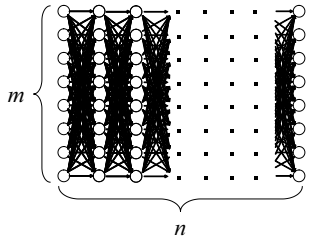


Figure 2.2: Trellis structure.

From the Viterbi algorithm,

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

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

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. But this computation cost is impractical recent application such as sequence labeling where the number of state is typically over $2,000$ [Brants 00, Matsuzaki 07] and text size are very large. For example, Google processes 20 peta bite web pages a day to enhance their search engine quality [Dean 08]. The naive solution to identifying the best model for query sequence 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 since the number of models is very large in real applications such as biological analysis, where over 790 millions genes are stored in DNA Data Bank of Japan to analyze gene data [Sugawara 08]. Furthermore, the naive approach to monitoring data stream is to perform this procedure each time a sequence value arrives. However, the naive approach is impracti-

cal considering the high frequency with which new values will arrive in recent traffic monitoring application. For example, over 2,000 GPS are attached to vehicles in Sweden*. Therefore more efficient algorithms are needed.

2.2 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 86]. Recently, HMMs have been applied in various fields such as pattern recognition and time sequence clustering. The data engineering community has published many studies on time sequence matching and query processing of uncertain data, which are slightly related to this work.

2.2.1 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 89]. 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 93]. 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

*http://www-03.ibm.com/press/us/en/pressrelease/28463.wss

quantized probability table [Sagayama 95]. These techniques can be applied to complement the proposed method. Unfortunately, it still incurs excessive CPU cost, especially for large datasets, since all possible likelihoods are computed.

Given the inefficiency of the Viterbi algorithm, several exact algorithms have already been proposed for handling large label sets. [Felzenszwalb 03] presented a fast inference algorithm for HMMs with large numbers of states, i.e., labels. Their algorithm works under the assumption that the hidden states can be embedded in a grid space, and transition probability corresponds to distance on that space. The application of their algorithm is restricted to HMMs with such a specific transition probability distribution. Simply speaking, their technique is feasible if the transition probability is a function of $|i - j|$, where i and j are state indices. CarpeDiem [Esposito 09] is an algorithm that accelerates likelihood computation without assuming any specific type of probability distribution. It can accelerate computation of any sequence labeling model under the assumption that the adjacent labels are not strongly correlated. However, this assumption can still be violated in real applications, as I will demonstrate in the experiment. In contrast, the proposed algorithm makes no such strong assumptions on data and thus has wider applicability.

The A* algorithm [Hart 68], a generalization of Dijkstra's algorithm, is also applicable to sequence labeling as an alternative to the Viterbi algorithm. However, it is known that the performance of the A* algorithm crucially depends on the heuristic function. To my knowledge, no effective heuristic functions for sequence labeling have been proposed so far, and thus it is not trivial to speed-up decoding by applying the A* algorithm. In fact, as [Esposito 09] reported, the A* algorithm can be slower than the Viterbi algorithm in sequence labeling problems.

Although I investigate decoding acceleration for the case of a large number of labels, other factors degrading decoding efficiency have also been studied by other researchers. [Roth 05] used integer linear programming to incorporate the long distance dependency between la-

bels that violates the Markov assumption and makes the Viterbi algorithm infeasible. [Qian 09] also presented an exact inference algorithm for dealing with sparse non-local features. [Lifshits 07] proposed a compression-based decoding algorithm for labeling a long repetitive sequence, such as DNA.

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 92, F. Jelinek 99]. 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.

Other approximate algorithms have also been proposed for speeding-up decoding in the context of sequence labeling. For example, [Tsuruoka 05] proposed easiest-first deterministic decoding for bidirectional dependency networks [Toutanova 03]. Although the bidirectional dependency network is more complex than linear-chains, their algorithm is equally applicable to proposed case. [Siddiqi 05b] presented the parameter tying approach for fast inference in HMMs. In their approach, several elements of the transition probability matrix are approximated as a constant value to decrease the computation cost. In following works, a similar idea was also applied to CRFs [Cohn 06, Jeong 09]. In these cases, several weights of features are set constant.

In general, approximate algorithms are much faster than exact algorithms, and hence exact algorithms are rarely used when dealing with a large number of labels. However, approximate algorithms suffer from two problems. First, it is hard for most of the approximate algorithms to not only find the optimal solution but to even bound the error rate. Second, approximate algorithms usually require hyperpa-

rameters, which control the tradeoff between accuracy and efficiency (e.g., beam width), and these have to be manually adjusted for each task.

This thesis, on the other hand, presents an exact algorithm that is free from these two problems. I also empirically demonstrate that the proposed algorithm scales well to a large number of labels; its speed is comparable to an approximate algorithm.

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 96]. 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 99, Li 00]. Lae et al. similarly studied the k-clustering problem for sequence datasets [Law 00]. They applied rival-penalized competitive learning to the problem to improve clustering accuracy.

2.2.2 Data engineering

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 93]. Faloutsos et al. and Moon et al. generalized whole sequence matching to subsequence matching (similarity searches that focus on arbitrary length sequences) [Faloutsos 94, Moon 02]. These studies use Euclidean distance as the similarity distance measure.

Even though many similarity functions have been proposed [Agrawal 95, Das 97] , 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 98]. Keogh inves-

tigated a search method based on global constraints which limits the duration along the time axis [Keogh 02]. 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 03]. Probabilistic threshold queries were also investigated by Cheng et al.; probabilities are computed to determine whether they exceed a given threshold [Cheng 04]. Tao et al. introduced U-tree [Tao 05] 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 05], scheduling [Babcock 03], load shedding [Tatbul 03], and memory requirements [Arasu 02]. Various architectures for data stream management systems, such as Aurora [Abadi 03], Stream [Motwani 03], Telegraph [Chandrasekaran 03], and Gigascope [Cranor 03], have been presented. They are slightly related to the proposed work.

Chapter 3

Efficient likelihood computation for HMM

In the past decade, sequence labeling algorithms such as HMMs, CRFs, and Collins' perceptrons have been extensively studied in the field of NLP [Rabiner 89, Lafferty 01, Collins 02]. Especially HMM is the most important tool for NLP tasks.

One important issue in sequence labeling problems is how to find the most probable label (state) sequence among all possible ones. This task, referred as decoding, is usually carried out using the Viterbi algorithm [Viterbi 67]. The Viterbi algorithm has $O(nm^2)$ time complexity,* where n is the input size and m is the number of labels, and is generally efficient. However, the Viterbi algorithm becomes prohibitively slow in dealing with a large number of labels, since its computational cost is quadratic in m.

Unfortunately, several sequence labeling problems in NLP involve a large number of labels. For example, there are more than 40 and 2000 labels in POS tagging and supertagging, respectively [Brants 00, Matsuzaki 07]. These tasks incur much higher computational costs than simpler tasks like NP chunking. What is worse, the number of labels grows drastically if multiple tasks are jointly performed. As I

*In case the first-order Markov assumption is made.

shall see later, over 300 labels needed to reduce joint POS tagging and chunking into the single sequence labeling problem. Although joint learning has attracted much attention in recent years, how to perform decoding efficiently still remains an open problem.

In this chapter, I present a new decoding algorithm that overcomes this problem [†]. The distinguishing property of the proposed algorithm is threefold: (1) It is much more efficient than the Viterbi algorithm in dealing with a large number of labels. (2) It is an exact algorithm, that is, the optimality of the solution is always guaranteed unlike approximate algorithms. (3) It is automatic, requiring no task-dependent hyperparameters that have to be manually adjusted.

Experiments evaluate the proposed algorithm on three tasks: POS tagging, joint POS tagging and chunking, and supertagging. The results demonstrate that the proposed algorithm is up to orders of magnitude faster than the Viterbi and state-of-the-art algorithm [Esposito 09], making exact decoding practical even in labeling problems with a large label set.

3.1 Staggered Decoding

This section presents the new decoding algorithm. The key is to reduce the labels. The proposed algorithm locates the best label sequence by iteratively solving labeling problems with the reduced label set. This results in significant time saving in practice, because each iteration becomes much more efficient than solving the original labeling problem. More importantly, the proposed algorithm always obtains the exact solution. This is because the algorithm allows us to check the optimality of the solution using only the reduced label set.

[†]Proposed implementation is available at http://www.tkl.iis.utokyo. ac.jp/ kaji/staggered.

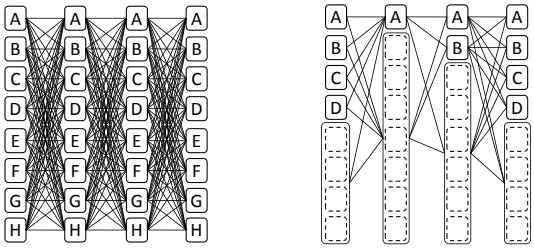


Figure 3.1: Lattice structure.

3.1.1 Degenerate lattice

I begin by introducing the *degenerate lattice*, which plays a central role in the proposed algorithm. Consider the lattice in Figure 3.1(a) where the alphabets {A, B, C, D, E, F, G, H} represent the labels associated with the nodes. Following convention, each path of the lattice is regarded as a label sequence. Note that the label set is {A, B, C, D, E, F, G, H}. By aggregating several nodes in the same column of the lattice, the original lattice can be transformed into a simpler form, which called as the degenerate lattice (Figure 3.1(b)).

Let us examine the intuition behind the degenerate lattice. Aggregating nodes can be viewed as grouping labels into a new one. Here, a label is referred to as an *active label* if it is not aggregated (e.g., {A, B, C, D} in the first column of Figure 3.1(b)), and otherwise as an *inactive label* (i.e., dotted nodes). The new label, which is made by grouping the inactive labels, is referred to as a *degenerate label* (i.e., large nodes covering the dotted ones). Two degenerate labels can be seen as equivalent if their corresponding inactive label sets are the same (e.g., degenerate labels in the first and the last column). In this approach, each path of the degenerate lattice can also be interpreted as a label sequence. In this case, however, the label to be assigned is either an active label or a degenerate label.

Then, the parameters regarding the degenerate label u_j are defined.

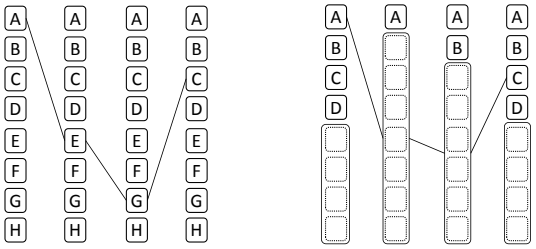


Figure 3.2: Paths in lattice structures

For reasons that will become clear later, transition probabilities are set to the maxima among the parameters of the inactive labels:

$$a_{u_i,u_j} = \max_{u'_j \in I(u_j)} (a_{u_i,u'_j}), \tag{3.1}$$

$$a_{u_j,u_i} = \max_{u'_j \in I(u_j)} (a_{u'_j,u_i}) \tag{3.2}$$

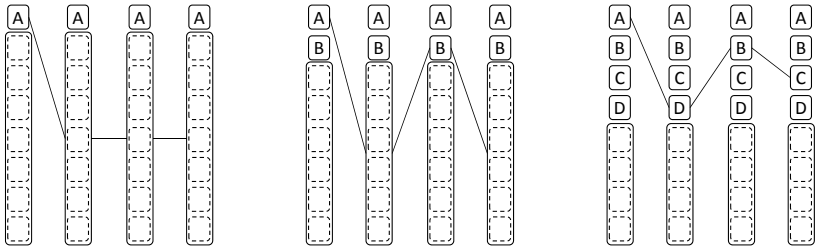
where u_i is an active label and $I(u_j)$ denotes one-to-one mapping to its corresponding inactive label set. Initial and symbol probabilities are set similarly.

The degenerate lattice has an important property on which the proposed algorithm is based:

Lemma 1. *If the best path of the degenerate lattice does not include any degenerate labels at all, it is equivalent to the best path of the original lattice.*

Proof. Let z_{max} be the best path of the degenerate lattice. The goal is to prove that if z_{max} does not include degenerate labels, then z_{max} gives the likelihood P of the model. Let y be the best path in the original lattice and Y be the set of all paths, I show this by partitioning Y into two disjoint sets: Y_0 and Y_1, where Y_0 is the subset of Y appearing in the degenerate lattice. Notice that $z_{max} \in Y_0$. Let P' be the likelihood by z_{max}, since z_{max} is the best path of the degenerate lattice,

$$\forall \mathbf{y} \in Y_0, \quad P \leq P'.$$



(a) (b) (c)

Figure 3.3: Staggered decoding.

The equation holds when $y = z_{max}$. I next examine the label sequence y such that $y \in Y_1$. For each path $y \in Y_1$, there exists a unique path z of the degenerate lattice that corresponds to y (Figure 3.2 where (a) The path $y = \{\text{A, E, G, C}\}$ of the original lattice, and (b) The path z of the degenerate lattice that corresponds to y.). Therefore,

$$P \leq P'$$

from Equation (3.1) and (3.2). These results complete the proof. □

3.1.2 Algorithm

This section describes the proposed method, which called as *Staggered Decoding* algorithm. The algorithm successively constructs degenerate lattice and checks whether the best path has degenerate labels. In building the degenerate lattice, labels with high probability, which is estimated from training data, are preferentially selected as the active label, expecting that such labels are likely to belong to the best path. The detailed algorithm is as follows:

Initialization step The algorithm starts by building the degenerate lattice in which there is only one active label in each column. The highest label is selected as the active label.

Search step The best path of the degenerate lattice is located (Figure 3.3(a) where the best path of the initial degenerate lattice, which is denoted by the line, is located.). This is done by using the Viterbi algorithm (and pruning technique, as described in Section 4.1.4). If the best path does not include any degenerate labels, The process is terminated since it is identical with the best path of the original lattice according to Lemma 1. Otherwise, proceed to the next step.

Expansion step The first process of this step doubles the number of the active labels in the degenerate lattice. The new active labels are selected from the current inactive label set in descending order of probabilities. If the inactive label set becomes empty, the original lattice is simply constructed. After expanding the active labels, the process backs to the previous step (Figure 3.3(b) where the active labels are expanded and the best path is searched again.). This procedure is repeated until the termination condition in the search step is satisfied, i.e., the best path has no degenerate labels (Figure 3.3(c) where the best path without degenerate labels is obtained.).

3.1.3 Pruning

It is crucial for the proposed algorithm to efficiently perform the search step. For this purpose, the Viterbi algorithm is basically used. In earlier iterations, the Viterbi algorithm is indeed efficient because the label set to be handled is much smaller than the original one. In later iterations, however, the proposed algorithm drastically increases the number of labels, making Viterbi decoding quite expensive.

To handle this problem, I propose a method of pruning the lattice nodes. This technique is motivated by the observation that the degenerate lattice shares many active labels with the previous iteration. In the remainder of Section4.1.4, I explain the technique by taking the

following steps:

- Section 3.1.3 examines a lower bound l such that $l \leq P$.
- Section 3.1.3 examines the maximum score $g(y_i)$ in case the token (symbol) x_i takes the label y_i:

$$g(y_i) = P_{y'_i = y_i}.$$

- Section 3.1.3 presents pruning procedure. The idea is that if $g(y_i) < l$, then the node corresponding to y_i can be removed.

Lower bound

The lower bound l can be trivially calculated in the search step. This can be done by retaining the best path among those consisting of only active labels. The score of that path is obviously the lower bound. Since the search step is repeated until the termination criteria is met, the lower bound can be updated at every search step. As the iteration proceeds, the degenerate lattice becomes closer to the original one, and therefore the lower bound becomes tighter.

Maximum score $g(y_i)$

The maximum score $g(y_i)$ can be computed from the original lattice. Let $P'_{1,i}$ be the best score of the partial label sequence ending with y_i. Presuming to traverse the lattice from left to right, $P'_{1,i}$ can be defined as

$$P'_{1,i-1} \cdot a_{y_{i-1},y_i} \cdot b_{y_i}(x_i)$$

If I traverse the lattice from right to left, an analogous score $P'_{i,m}$ can be defined as

$$P'_{i+1,m} \cdot a_{y_i,y_{i+1}} \cdot b_{y_i}(x_i)$$

Using these two scores,

$$g(y_i) = P'_{1,i} \cdot P'_{i,m} / b_{y_i}(x_i)$$

Notice that updating $P'_{1,i}$ or $P'_{i,m}$ is equivalent to the forward or backward Viterbi algorithm, respectively.

Although it is expensive to compute $P'_{1,i}$ and $P'_{i,m}$, their upper bounds can be efficiently estimated. Let $\lambda_{1,i}$ and $\lambda_{i,m}$ be analogous scores to $P'_{1,i}$ and $P'_{i,m}$ that are computed using the degenerate lattice. $P'_{1,i} \leq \lambda_{1,i}$ holds and $P'_{i,m} \leq \lambda_{i,m}$, by following similar discussions as raised in the proof of lemma 1. Therefore, it is easy to check whether $g(y_i)$ is smaller than l by using $\lambda(y_i)$ and $\bar{\lambda}(y_i)$:

$$\begin{aligned} g(y_i) &= P'_{1,i} \cdot P'_{i,m} / b_{y_i}(x_i) \\ &\leq \lambda'_{1,i} \cdot \lambda'_{i,m} / b_{y_i}(x_i) < l. \end{aligned}$$

For the sake of simplicity, y_i is assumed to be an active label. Although I do not discuss the other cases, the proposed pruning technique is also applicable to them.

$\lambda_{1,i}$ and $\lambda_{i,m}$ are computed by using the forward and backward Viterbi algorithm alternately. In the search step immediately after initialization, The forward Viterbi algorithm is performed to find the best path, that is, $\lambda_{1,i}$ is updated for all y_i. In the next search step, the backward Viterbi algorithm is carried out, and $\lambda_{i,m}$ is updated. In the succeeding search steps, they are updated alternately. As the algorithm progresses, $\lambda_{1,i}$ and $\lambda_{i,m}$ become closer to $P'_{1,i}$ and $P'_{i,m}$.

Pruning procedure

Making use of the bounds, the lattice nodes can be pruned. To do this, the values of l, $\lambda_{1,i}$ and $\lambda_{i,m}$ are kept. They are set as $l = -\infty$ and $\lambda_{1,i} = \lambda_{i,m} = \infty$ in the initialization step, and are updated in the search step. The lower bound l is updated at the end of the search step, while $\lambda_{1,i}$ and $\lambda_{i,m}$ can be updated during the running of the Viterbi algorithm. When $\lambda_{1,i}$ or $\lambda_{i,m}$ is changed, the proposed approach checks whether $g(y_i) < l$ holds and the node is pruned if the condition is met.

3.1.4 Analysis

This section provides a theoretical analysis of the proposed algorithm. In the following proofs, m, s, and n represents the number of original labels, the number of distinct tokens, and the length of input token sequence, respectively. To simplify the discussion $\log_2 m$ is assumed to be an integer (e.g., $m = 64$).

I first provide three lemmas for preparation:

Lemma 2. *The proposed algorithm requires at most* $(\log_2 m + 1)$ *iterations to terminate.*

Proof. There are 2^{i-1} active labels in the i-th search step ($i = 1, 2 \dots$), which means there are m active labels and no degenerate labels in the $(\log_2 m + 1)$-th search step. Therefore, the algorithm always terminates within $(\log_2 m + 1)$ iterations. □

Lemma 3. *The number of degenerate labels is* $\log_2 m$.

Proof. Since there is one new degenerate label in all but the last expansion step, there are $\log_2 m$ degenerate labels. □

Lemma 4. *The Viterbi algorithm requires* $O(m^2 + ms)$ *memory space and has* $O(nm^2)$ *time complexity.*

Proof. The proof is omitted since it is obvious. □

Making use of the above statements, main results can be established:

Theorem 1. *The proposed algorithm asymptotically requires* $O(m^2 + ms)$ *memory space.*

Proof. Since there are m original labels and $\log_2 m$ degenerate labels, the proposed algorithm requires $O((m+\log_2 m)^2+(m+\log_2 m)s)$ memory space to perform Viterbi decoding in the search step. In case $\log_2 m \ll m$, the algorithm requires the order of $O(m^2 + ms)$ memory space. □

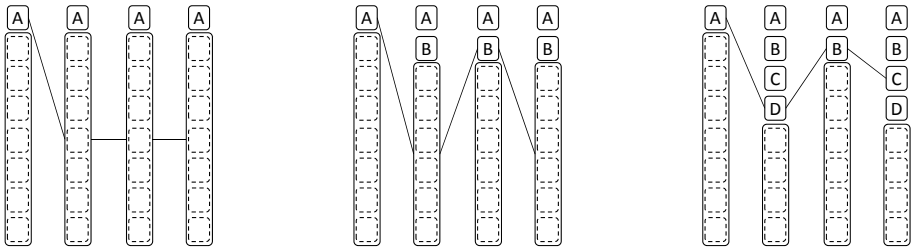


Figure 3.4: Staggered decoding with column-wise expansion.

Theorem 2. *The proposed algorithm has $O(n)$ best case time complexity and $O(nm^2)$ worst case time complexity.*

Proof. To perform the i-th search step, the proposed algorithm requires the order of $O(n4^{i-1})$ time because there are 2^{i-1} active labels. Therefore, the proposed algorithm has $O(\sum_{i=1}^{I} m4^{i-1})$ time complexity if it terminates after the I-th search step. In the best case, $I = 1$, the time complexity is $O(m)$. In the worst case, $M = \log_2 m + 1$, the time complexity is the order of $O(nm^2)$ because $\sum_{i=1}^{\log_2 m+1} m4^{i-1} < \frac{4}{3}nm^2$. □

Theorem 1 shows that the proposed algorithm asymptotically requires the same order of memory space as the Viterbi algorithm. Theorem 2 reveals that the proposed algorithm has the same order of time complexity as the Viterbi algorithm even in the worst case. In practice, the time complexity of the proposed algorithm lies somewhere between $O(m)$ and $O(nm^2)$.

3.1.5 Heuristic techniques

I finally present two heuristic techniques for further speeding up the proposed algorithm.

First, the value of lower bound l can be initialized by selecting a path from the original lattice in some ways, and then computing the score of that path. In experiments, I use the path located by the left-to-right

deterministic decoding. Although this method requires an additional cost to locate the path, it is practically effective. If l is initialized in this manner the best case time complexity becomes $O(nm)$.

The second technique is on the expansion step. Instead of the expansion technique described in Section 3.1.2, the active labels can be expanded in a heuristic manner to keep the number of active labels smaller:

Column-wise expansion step This step doubles the number of the active labels in the column, only if the best path of the degenerate lattice passes through the degenerate label of that column (Figure 3.4 where (a) the best path of the initial degenerate lattice, which does not pass through the degenerate label in the first column, (b) column-wise expansion is performed and the best path is searched again, notice that the active label in the first column is not expanded, and (c) the final result.).

A drawback of this strategy is that the algorithm requires $n(\log_2 m + 1)$ iterations in the worst case. As the result, this approach can no longer derive reasonable upper bound for the time complexity. Nevertheless, column-wise expansion is practically effective as I will demonstrate in the experiment. Note that theorem 1 still holds true even if this column-wise expansion is used.

3.2 Experiments and Discussion

3.2.1 Setting

The proposed algorithm was evaluated with three tasks: POS tagging, joint POS tagging and chunking (called joint tagging for short), and supertagging. To reduce joint tagging into a single sequence labeling problem, I produced the labels by concatenating the POS tag and the chunk tag (BIO format), e.g., NN/B-NP. In the two tasks other than

Table 3.1: Decoding speed (sent./sec).

	POS tagging	Joint tagging	Supertagging
VITERBI	5800	110	1.4
CARPEDIEM	12,000	67	0.27
SD-I	12,000	1200	170
SD-II	**19,000**	**2200**	**430**

supertagging, the input token is the word. In supertagging, the token is the pair of the word and its oracle POS tag.

The data sets I used for the three experiments are the Penn Tree-Bank (PTB) corpus, CoNLL 2000 corpus, and an HPSG treebank built from PTB corpus [Matsuzaki 07], respectively. The number of labels in the three tasks is 45, 319, and 2602, respectively.

The perceptron algorithm was used for training. The models were averaged over 10 iterations. For features, I basically followed previous studies [Tsuruoka 05, Sha 03, Ninomiya 06]. The performance of the algorithm was investigated on the test data. I used the decoding speed (sentences/sec) as the evaluation measure.

3.2.2 Results and discussions

Table 3.1 represents the performance of the proposed algorithm. SD-I represents the proposed algorithm that does not use column-wise expansion, while SD-II uses column-wise expansion. For comparison, the result of two baseline algorithms are presented as well: VITERBI and CARPEDIEM [Esposito 09]. In almost all settings, both of the proposed algorithms outperformed the two baselines. And SD-II performed consistently better than SD-I. This indicates the effectiveness of the column-wise expansion.

CARPEDIEM is the most relevant algorithm, except for VITERBI, for sequence labeling problems in NLP, as discussed in Section 2.2. However, results demonstrated that CARPEDIEM worked poorly in two

of the three tasks. This is because the transition information is crucial for the two tasks, and the assumption behind CarpeDiem is violated. In contrast, the proposed algorithms performed reasonably well across the three tasks, demonstrating the wide applicability of the proposed algorithm.

Table 3.2 represents the average iteration number of SD-I and SD-II. The two algorithms required almost the same number of iterations on average, although the iteration number is not tightly bounded if column-wise expansion is used. This indicates that SD-II could virtually avoid performing extra iterations, while heuristically restricting active labels to be expanded.

The degenerate label may be interpreted as a coarse-level label. In this sense, SD algorithm is related to coarse-to-fine PCFG parsing [Charniak 06]. While the proposed algorithm bears some resemblance, there are at least two differences. First, the coarse-to-fine parsing is essentially a heuristic pruning approach, and it is not an exact algorithm. Second, SD algorithm does not always perform decoding at the fine-grained level. It is designed to be able to stop decoding at the coarse-level.

Table 3.3 compares training time spent by Viterbi and SD-II. Although speeding up perceptron training is a by-product, it is interesting to see that the proposed algorithm is in fact effective at training time as well. The result also indicates that the proposed algorithm is more effective at test time. This is probably because the model is not predictive enough at the beginning of training, and the pruning is not effective.

3.2.3 Comparison with approximate algorithm

I finally compare two exact algorithms (Viterbi and SD-II) with beam search, which is the approximate algorithm widely adopted for sequence labeling problems in NLP. For this experiment, the beam width B was exhaustively calibrated: I tried $B = \{1, 2, 4, 8, ...\}$ until the difference

Table 3.2: The average number of iterations.

	POS tagging	Joint tagging	Supertagging
SD-I	6.03	8.11	10.0
SD-II	6.14	8.61	10.6

Table 3.3: Training time.

	POS tagging	Joint tagging	Supertagging
VITERBI	77 sec.	800 sec.	79 hour
SD-II	31 sec.	71 sec.	4.7 hour

Table 3.4: Comparison with beam search.

	POS tagging	Joint tagging	Supertagging
VITERBI	5800	110	1.4
SD-II	19,000	2200	**430**
BEAM	**27,000**	**4000**	270

between beam search and the exact algorithm fell below 0.1.

Table 3.4 summarizes the results; there is a substantial difference in the performance between VITERBI and BEAM. On the other hand, SD-II reached very close speed to BEAM. In fact, they achieved comparable performance in the experiment. These results demonstrate that the proposed approach successfully bridge the gap in the performance between exact and approximate algorithms, while retaining the advantages of exact algorithms.

3.3 Summary

The sequence labeling algorithm is indispensable to modern statistical NLP. However, the Viterbi algorithm, which is the standard decoding algorithm in NLP, is not efficient when a large number of labels have to be dealt with. In this chapter I presented staggered decoding, which provides a principled way of resolving this problem. I consider this

algorithm serves as an alternative to the Viterbi algorithm in various NLP tasks.

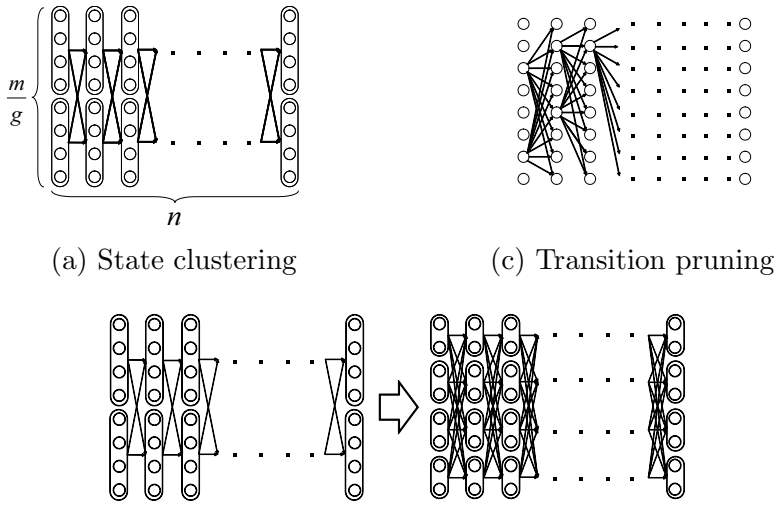
Chapter 4

Efficient search method for HMM data set

HMM is a ubiquitous tool for representing probability distributions over sequences of observations. Since HMMs, which assess time 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 chapter is efficient identification of the model whose state sequence has the highest likelihood, for the given query time sequence, from datasets with exactness. To the best of my knowledge, this is the first study to address the HMM search problem that guarantees exactness.

4.1 Proposed method

This chapter mainly focuses on a query designed to identify the model that has the highest likelihood accurately from HMM datasets. But the proposed approach can also efficiently support range queries and K-nearest neighbor queries as described in section 4.1.5.



(a) State clustering

(c) Transition pruning

(b) Multi-granularities

Figure 4.1: Basic ideas behind SPIRAL.

4.1.1 Ideas behind SPIRAL

The proposed solution, SPIRAL, is based on the three ideas, described below.

Likelihood approximation The first idea is approximations to reduce the high cost of the Viterbi algorithm solution. Instead of computing the exact likelihood of every model, the proposed approach approximates 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, m/g states are created by merging 'similar' states in the model (See Figure 4.1 (a)), which requires $O(nm^2/g^2)$ time to obtain the approximate likelihoods instead of $O(nm^2)$ time, which the Viterbi algorithm solution requires. A clustering approach is used to find groups of similar states, then create compact models. They are

referred as *degenerate* models.

This new idea has the following two major advantages. First, likely models can be found without any omissions even though approximations are used. Search omissions are avoided completely by the use of the upper bounding likelihood. This means that unpromising models can be safely discarded along with their approximate likelihoods at low CPU cost.

The second advantage is that this idea does not depend on model type; the approximate likelihoods can be estimated for any model type since this does 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, I 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. For this reason, models of multiple granularities are generated 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. This approach then gradually increases the size of the models (i.e., a model with a smaller g is used), which improves the accuracy of the approximate likelihood, as the search progresses (See Figure 4.1 (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, fine-grained approximation is applied for only to the models that remain after coarse-grained approximation. Consequently, This approach 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 the proposed approach 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 the proposed approximation technique is able to discard most unlikely models, the proposed approach still relies on exact likelihood computation to guarantee the correctness of the search results. Here I focus on reducing the cost of this computation.

The Viterbi path shows the state sequence that gives the likelihood. Mirroring the Viterbi path, 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. I therefore ask the question, which paths in the structure are not promising to explore? This can be answered by using a threshold (say ϵ).

The proposed search algorithm maintains a candidate (i.e., best-so-far) likelihood before reporting the final likelihood. ϵ is used as the best-so-far highest likelihood. ϵ is updated and increases when a promising model is found during search processing. The unlikely paths are excluded in the trellis structure by using ϵ, since ϵ never decreases during search processing. If the upper bounding likelihood of paths that pass through a state is less than ϵ, that state cannot be contained in the Viterbi path, this approach can safely discard the paths.

From the monotonically increasing property of ϵ, This approach can search for the most likely model efficiently over a long time sequence. This is an attractive characteristic considering that the time 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 this approach can compute the approximate likelihood more efficiently.

4.1.2 Likelihood approximation

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

State clustering

Attempts to minimize model complexity by aggregating states have been reported in the field of reinforcement learning [Singh 94]. The size of the trellis structure is reduced by merging similar states in order to compute likelihoods at low computation cost. To achieve this, a clustering approach is adopted. Given granularity g, this approach tries to find m/g clusters from among the m original states. I first describe how to compute the probabilities of a new degenerate model, and then show the clustering method.

This approach merge all the states in a cluster and create a new state. For the new state, the highest probability among the probabilities of the states is chosen to compute the upper bounding likelihood (described in Section 4.1.2). This approach obtain the probabilities of new state u_c by merging all the states in cluster $\mathcal{C}$ as follows:

$$
\begin{aligned}
&\pi'_c = \max_{u_i \in \mathcal{C}}(\pi_i), \qquad a'_{cc} = \max_{u_i \in \mathcal{C},\ u_k \in \mathcal{C}}(a_{ik}), \\
&a'_{cj} = \max_{u_i \in \mathcal{C}}(a_{ij}) \quad \text{for any} \quad u_j \notin \mathcal{C}, \\
&a'_{jc} = \max_{u_i \in \mathcal{C}}(a_{ji}) \quad \text{for any} \quad u_j \notin \mathcal{C}, \\
&b'_c(v) = \max_{u_i \in \mathcal{C}}(b_i(v)).
\end{aligned}
\tag{4.1}
$$

I use the following example to explain state clustering.

Example 3. *I 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 $\mathcal{C}_2$. The new state probability is obtained by taking the maximum value of the*

original probabilities:

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

Thus,

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

The following vector of features F_i is used to cluster state u_i.

$$F_i = (\pi_i; a_{i1}, \ldots, a_{im}, a_{1i}, \ldots, a_{mi}; b_i(v_1), \ldots, b_i(v_s)). \tag{4.2}$$

where s is the number of symbols. This approach chooses 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 experiments, the well-known k-means method was used to cluster states * where the Euclidean distance is used as a distance measure. However, I can exploit BIRCH [Zhang 96] 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 chapter.

Upper bounding likelihood

Approximate likelihood P' is computed from degenerate models that have $m'(= m/g)$ states. Given a degenerate model, its approximate

*I repeat the clustering procedure until there are no more changes.

likelihood is as computed follows:

$$P' = \max_{1 \le c \le m'} (p'_{cn}) \tag{4.3}$$
$$p'_{ct} = \begin{cases} \max_{1 \le j \le m'} (p'_{j(t-1)} \cdot a'_{jc}) \cdot b'_c(x_t) & (2 \le t \le n) \\ \pi'_c \cdot b'_c(x_1) & (t = 1). \end{cases}$$

where p' is the maximum probability of states.

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

$$P \le P'. \tag{4.4}$$

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

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

If $2 \le t \le n$,

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

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,

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

which completes the proof. □

Theorem 1 provides SPIRAL with the property of finding the exact answer. I provide a proof of this property in Section 4.1.6.

4.1.3 Multi-granularity

In the previous section, I presented an algorithm that computed the approximate likelihood of a degenerate model with a single level of

granularity. However, I can also exploit multiple granularity. Here, I describe the gradual refinement of the likelihood approximation with multiple granularity.

In this approach, $h+1$ distinct granularities are used that form a geometric progression $g_i = 2^i$ $(i = 0, 1, 2, \ldots, h)$. Therefore, trellis structures of models that have $\lfloor m/g_i \rfloor$ states are generated. g_h represents the smallest (coarsest) model[†] 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, This approach first computes the coarsest structure for all models. It then obtains 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, compute a finer-grained structure for that model, and check whether the approximate likelihood is smaller than θ. It iterates this check until 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, the proposed approach can prune unlikely models with appropriate granularity according to exact likelihood.

Later I describe search algorithms based on these approaches.

4.1.4 Transition pruning

This section introduces an algorithm for computing likelihoods efficiently based on the following lemma:

Lemma 1. *Likelihoods of a state sequence are monotonic non-increasing.*

Proof: In Equations (2.3) and (4.3), likelihoods are computed by dynamic programming to maximize the probabilities from previous

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

states. This procedure ensures that the likelihood of a state is less than or equal to the likelihoods of any transited states. □

The above lemma is exploited in pruning paths in the trellis structure. This approach uses e_{it}, which indicates a conservative estimate of likelihood p_{it}, to prune unlikely paths as follows:

$$e_{it} = \begin{cases} p_{it} \cdot (a_{max})^{n-t} \cdot \prod_{j=t+1}^{n} b_{max}(x_j) & (1 \leq t \leq n-1) \\ p_{in} & (t = n) \end{cases} \tag{4.5}$$

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

$$a_{max} = \max_{i,j}(a_{ij}) \qquad (i = 1, \ldots, m; j = 1, \ldots, m) \tag{4.6}$$

$$b_{max}(v) = \max_{i} b_i(v) \qquad (i = 1, \ldots, m) \tag{4.7}$$

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 2. *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.*

$$p_{jn} \leq e_{it} \tag{4.8}$$

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 1:

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

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 4.1.6.

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. Algorithm 1 shows the algorithm for the likelihood computation. The algorithm prunes unlikely paths in the trellis structure with θ. This algorithm to the approximate likelihood computation can be similarly applied to the exact computation. Threshold θ is updated when searching the model. I show this algorithm in the next section.

Two arrays $\mathcal{S}$ and $\mathcal{S}'$ are used for dynamic programming to keep track of transitions. θ is also used to improve the efficiency. The algorithm initializes the first array, $\mathcal{S}$, 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 it has not to take the state into account at $t+1$. If $\mathcal{S}$ is empty, this approach terminate the likelihood computation since the given model cannot yield a search result. This approach 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. *I 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, I do not take this path into account when I*

Input: sequence X, threshold θ.
Output: estimate e.

add all states to $\mathcal{S}'$;
for $t := 1$ **to** n **do**
  $\mathcal{S} := \emptyset$;
  for $i := 1$ **to** m **do**
    compute e_{it} for X from the transitions of $\mathcal{S}'$;
    if $e_{it} \geq \theta$ **then**
      add u_i to $\mathcal{S}$;
    end if
  end for
  $\mathcal{S}' := \mathcal{S}$;
  if $\mathcal{S} = \emptyset$ **then**
    return $\max_{1 \leq i \leq m}(e_{it})$;
  end if
end for
return $\max_{1 \leq i \leq m}(e_{in})$;

Algorithm 1: Likelihood computation algorithm.

compute the probabilities at $t = 2$. Similarly, I 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 terminate the likelihood computation and determine that this is not a likely model.

4.1.5 Search algorithm

I show the proposed approach to finding the best model for query sequence in this section. The proposed approach is to (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. I present the simplest way of finding the candi-

date in [Fujiwara 08], 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 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.

I present a sophisticated search algorithm to overcome the above problem. The proposed algorithm computes 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.

Algorithm 2 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. It first computes the approximate likelihoods of g_h for all

Input: query sequence X.
Output: the best model M_{best}.

```
1: θ := 0;
2: add all models to M
3: for i := h to 0 do
4:    θ' := 0;
5:    for each model M ∈ M do
6:       compute P_i for M;
7:       if P_i ≥ θ' then
8:          M_max := M;
9:          θ' := P_i;
10:      end if
11:   end for
12:   compute P_0 for M_max;
13:   if P_0 ≥ θ then
14:      M_best := M_max;
15:      θ = P_0;
16:   end if
17:   for each model M ∈ M do
18:      if P_i < θ then
19:         subtract M from M;
20:      end if
21:   end for
22: end for
23: return M_best;
```

Algorithm 2: Algorithm for processing query sequences.

models, and then choose the best candidate. It obtains the initial value of θ by computing the exact likelihood of the best candidate. If the approximate likelihood is smaller than θ, it prunes the model since it cannot be a qualifying model. It continues to compute approximate likelihoods while gradually enhancing the accuracy. It computes the exact likelihood of the model that gives the maximum approximate likelihood at each level of granularity, and it updates the candidate and θ to find the best model efficiently, provided P_0 is larger than θ.

Although it 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, it would utilize a search threshold as θ, instead of the best likelihood used in the above search algorithm (i.e., it does not use the candidate). The best K-th likelihood would be utilized instead of the best likelihood for K-nearest neighbor queries.

4.1.6 Theoretical Analysis

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

Accuracy

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

Lemma 2. *The threshold θ is monotonic non-decreasing in the search process of SPIRAL.*

Proof: To find the best model in the search process, it first finds a candidate model based on the coarsest approximate likelihood, and set the initial θ from the model. It maintains the candidate as the best result; when it finds a model higher likelihood, the exact likelihood of the model is greater than θ, the candidate is replaced by the new model. This makes θ larger. Therefore, θ keeps increasing. □

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

Lemma 3. *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 1 and 2, it obtains $P_0 \leq P_i$, 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 2) 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. □

Complexity

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

Lemma 4. *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 5. *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. *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 4, 5 and 6 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, I show the effectiveness of the proposed approach by presenting the results of extensive experiments.

4.2 Experimental evaluation

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

4.2.1 Experimental data and environment

I 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 [‡]. It contains measurements from 64 electrodes placed on subjects' scalps that were sampled at 256 Hz (3.9-msec epoch) for 1 second. In experiments, I quantized EEG values in 1 microvolt steps, resulting in 506 elements. I computed the probabilities of models from a dataset of 6 subjects (co2a0000365, co2a0000368, co2a0000369, co2c0000338,

[‡]http://archive.ics.uci.edu/ml/

co2c0000339, and co2c0000340), and I extracted queries from a dataset of 2 subjects (co2a0000364 and co2c0000337).

- *Chromosome*:
  I used DNA strings of human chromosomes 2, 18, 21, and 22, which were obtained from the well-known NCBI website [§]. These DNA strings are composed of the letters {A,C,G,T,N} where N is *unknown*. I treat N as a different symbol, resulting in a symbol size of 5. In 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, I extracted sequences from the sensor measurements from April 10th to September 23th. I similarly extracted query sequences from sensor measurements from September 24th to October 1st.

- *UNIX*:
  I 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. I 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 82]. In experiments, sequence length is 256 and possible

[§]http://www.ncbi.nlm.nih.gov

transitions of left-right model are restricted only to 2 states, which is typical in many applications.

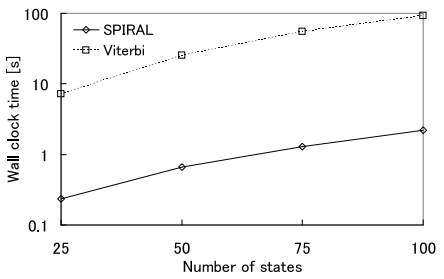
I evaluated the search performance mainly through wall clock time. All experiments were conducted on a Linux quad 3.33 GHz Intel Xeon server with 32GB of main memory. I implemented the proposed algorithms using GCC. Each result reported here is the average of 100 trials.

4.2.2 Search cost

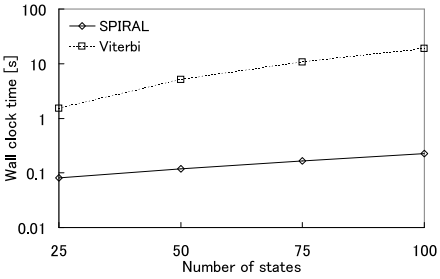
I assessed the search time needed for SPIRAL and Viterbi. I conducted trials with various numbers of states and models because differences in these numbers are expected to strongly impact the time taken by Viterbi to process HMM datasets.

Wall clock time versus number of states Figure 4.2 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. The proposed 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.

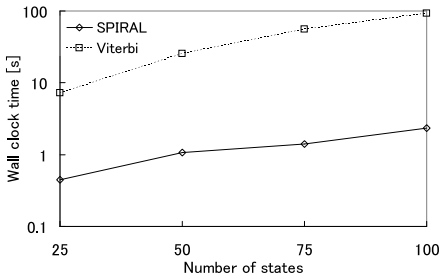
Wall clock time versus number of models Figure 4.3 shows the wall clock time as a function of the number of models, where the number of states is $m = 100$. SPIRAL is superior to the Viterbi algorithm as in the case of changing the number of states. Even if SPIRAL first



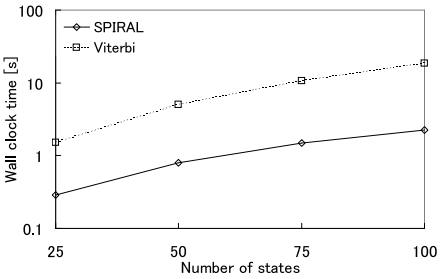
(a-1) *EEG*, ergodic



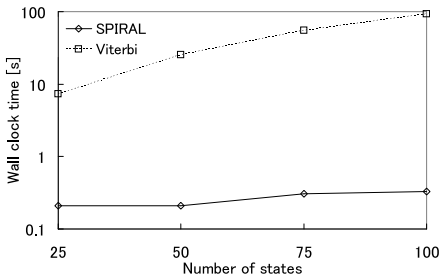
(a-2) *EEG*, left-right



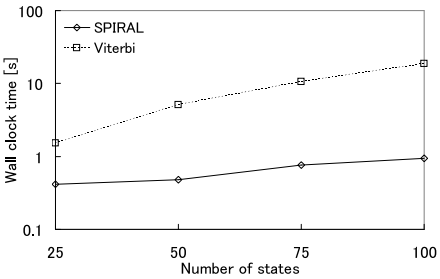
(b-1) *Chromosome*, ergodic



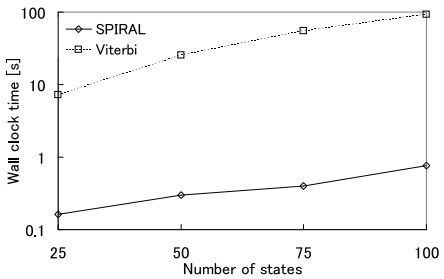
(b-2) *Chromosome*, left-right



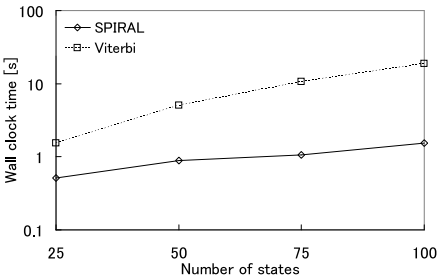
(c-1) *Traffic*, ergodic



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

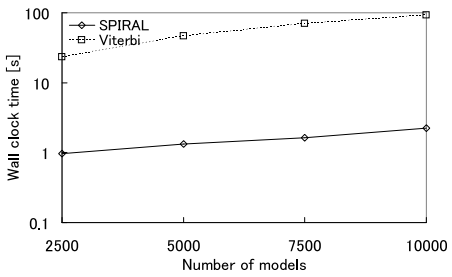


(d-1) *UNIX*, ergodic

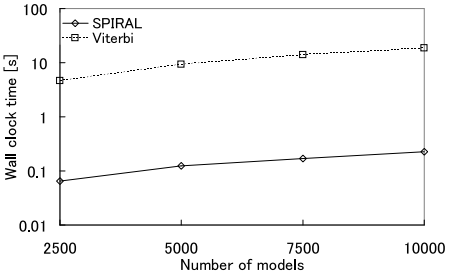


(d-2) *UNIX*, left-right

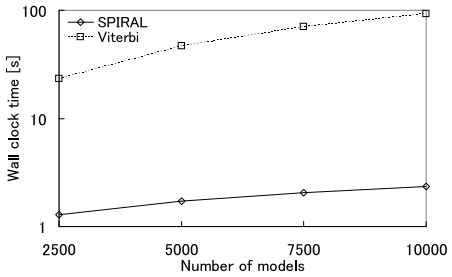
Figure 4.2: Wall clock time versus number of states.



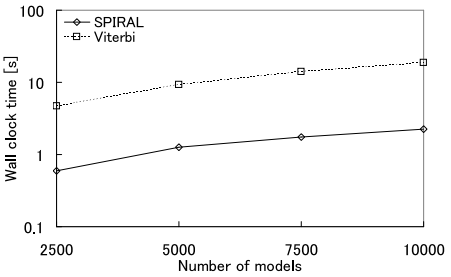
(a-1) *EEG*, ergodic



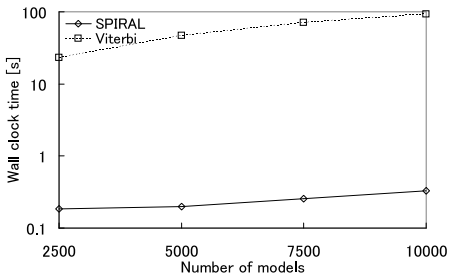
(a-2) *EEG*, left-right



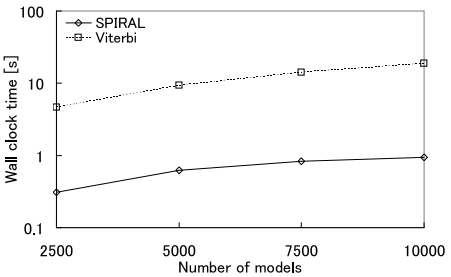
(b-1) *Chromosome*, ergodic



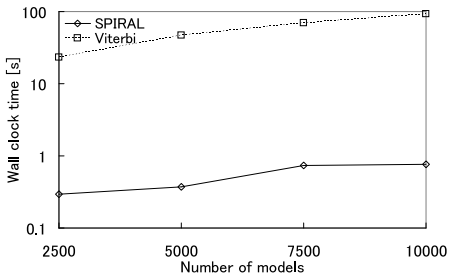
(b-2) *Chromosome*, left-right



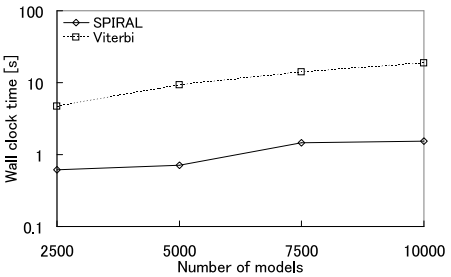
(c-1) *Traffic*, ergodic



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

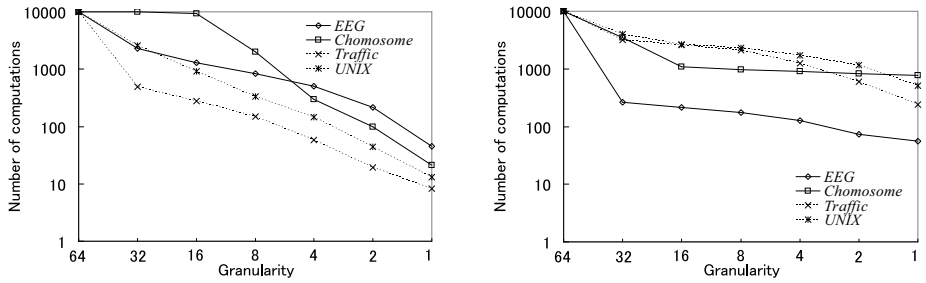


(d-1) *UNIX*, ergodic



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

Figure 4.3: Wall clock time versus number of models.



(a) Ergodic (b) Left-right

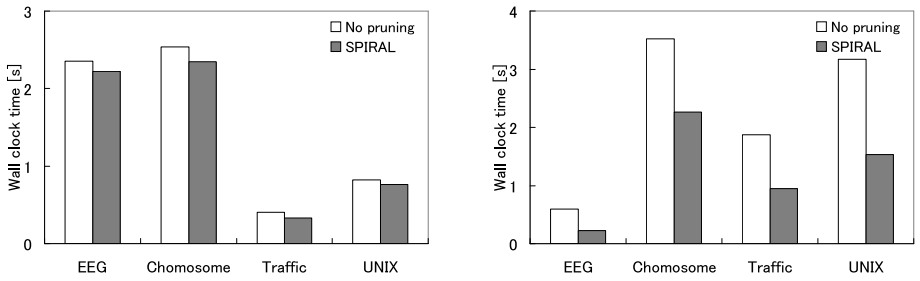
Figure 4.4: Number of likelihood computations.

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.

4.2.3 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, I evaluated the number of exact computations and approximations needed in SPIRAL. Figure 4.4 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



(a) Ergodic (b) Left-right

Figure 4.5: Wall clock time of transition pruning method.

achieves excellent search performance as shown in Figures 4.2 and 4.3.

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.

4.2.4 Effect of transition pruning

As mentioned in Section 4.1.4, SPIRAL excludes unlikely paths from the trellis structure to efficiently compute exact and approximate likelihoods. To show the effectiveness of this idea, I removed the path-pruning technique from SPIRAL, and reexamined the wall clock time. Figure 4.5 shows the result for 10,000 models of 100 states. SPIRAL without path-pruning is abbreviated as *No pruning* in this figure.

The results show that the transition pruning method can provide efficient search especially for left-right HMMs which has transition restriction. As mentioned in Section 2.1, left-right HMMs have a constraint, the initial state probability of state one is 1, unlike ergodic HMMs. Therefore if the conservative estimate of state one at the first point in left-right HMMs is smaller than ϵ, all paths of the model give the likelihoods smaller than ϵ and the model is pruned. SPIRAL is up to 3 times faster if the transition pruning method is used.

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 4.1: Wall clock time versus base number.

4.2.5 Granularity level

SPIRAL identifies the best model by gradually doubling approximate model sizes. This implies that the granularity of base 2 is used. 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 4.1 shows the wall clock time of SPIRAL for three base numbers against 10,000 models where each model has 100 states.

I can see that small base numbers raise the search speed for the ergodic HMM. 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, I can not see this trend for the left-right model. As a result, the memory space used can be reduced by the left-right model by adopting

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 4.2: Comparison of clustering method.

base numbers above 2 while keeping the search efficiency high.

4.2.6 Clustering approach

As mentioned in Section 4.1, SPIRAL can exploit arbitrary clustering methods and distance measures. However, the combination adopted can impact the search efficiency since a good clustering approach yields low approximate error. I compared the k-means method used in this chapter with the Euclidean distance to PAM with Jensen-Shannon divergence. PAM is the famous clustering method developed by Kaufman and Rousseeuw [Kaufman 05], and Jensen-Shannon divergence is a popular method of measuring the similarity between two probability distributions in probability theory and statistics. Table 4.2 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, additional experiments confirmed that PAM incurs high computation cost to construct degenerate structures from large data sets as described in a previous study [Kaufman 05]. Therefore, a user should be careful in selecting the clustering approach for a real application.

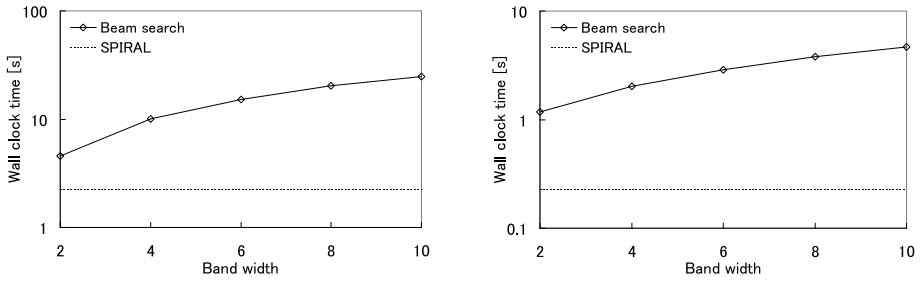
4.2.7 SPIRAL vs Beam search

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, I conducted comparative experiments using the well-known Beam search algorithm. I refer to the Beam search algorithm implementation as *Beam search*.

I varied the beam width, i.e. the number of states taken into account, for the Beam search algorithm. Figures 4.6 and 4.7 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.

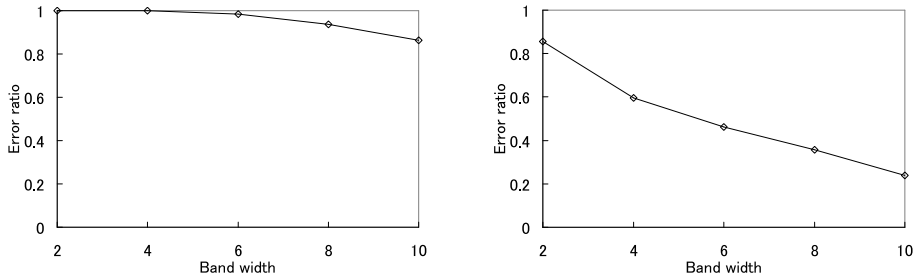
The results show that the Beam search algorithm forces a trade-off between speed and accuracy. That is, as the number of states decreases, the wall clock time decreases but the computation error increases. The Beam search algorithm is an approximation technique and so can miss the best path for the original 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, it has been difficult



(a) Ergodic (b) Left-right

Figure 4.6: Wall clock time versus bandwidth.



(a) Ergodic (b) Left-right

Figure 4.7: Error ratio versus bandwidth.

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.

4.3 Summary

This chapter addressed the problem of conducting a likelihood search on a large set of Hidden Markov Models (HMMs). I proposed SPIRAL,

which is based on three ideas to (1) prune low-likelihood models in the HMM dataset by their approximate likelihoods, which yields promising candidates in an efficient manner. (2) vary the approximation granularity for each model to maintain a balance between computation time and approximation quality. (3) discard unlikely paths in the trellis structure, which improves the efficiency.

SPIRAL achieves all of the following goals:

- High-speed search: experiments on real data show that it clearly outperforms the naive implementation, achieving an increase in speed of several orders of magnitude.
- I prove that it guarantees exactness.
- It can handle any HMM model type.

Experiments show that SPIRAL works as expected, and finds high-likelihood HMMs with high speed; Specifically, it is significantly faster than the naive implementation.

Chapter 5

Fast data stream monitoring with HMM data set

The previous chapter explained how the proposed 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, I show that SPIRAL can be used to monitor data streams. 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 environment. Researchers at several universities have embarked on projects to produce small, wireless, battery powered sensors and low level networking protocols [Kahn 99]. 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 00, Gehrke 01]:

- **Landmark window model:** In this temporal span, data streams are computed based on the values between a specific time

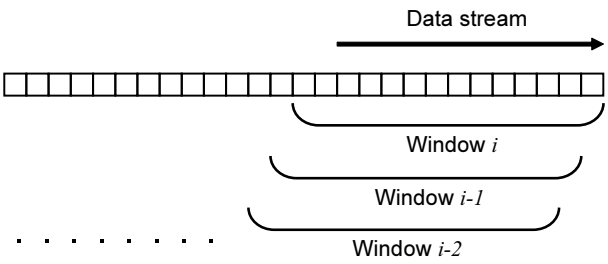


Figure 5.1: Sliding window model.

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 chapter focuses on the sliding window model, which is illustrated in Figure 5.1, because it is used most often and is the most general model [Zhu 02, Gao 05].

The data stream can be considered 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 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. It is assumed that 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 subsequences of the streaming sequence from x_1 to x_n are only interested in. 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 chapter.

In the previous section, it is assumed that there are a fixed number of HMMs and the operator-specified sequence. In this section, it is assumed that there are a fixed number of HMMs and a subsequence of the data stream, the problem 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.

5.1 Proposed solution

The proposed approach for data stream processing mainly follows the approach for static sequences mentioned in Section 4.1.5. The proposed 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.

The 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 the models can be efficiently pruned 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, this granularity can be used in the next time tick, otherwise the granularity level that is one step down (coarser) is used 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), I 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) , the 8 states model is selected (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.

The 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 4.1.5) 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, the exact likelihood of the best model of one time tick is first computed before to obtain the initial θ needed to identify the best model effectively.

Input: subsequence X, set of models $\mathcal{M}'$, the previous best model M'_{best}.
Output: the best model M_{best}.

compute P_0 for M'_{best}, $\theta := P_0$, $M_{best} := M'_{best}$,add $\mathcal{M}'_h$ to $\mathcal{M}_h$;
for $i := h$ **to** 1 **do**
  add $\mathcal{M}'_{i-1}$ to $\mathcal{M}_i$;
end for
for $i := h$ **to** 0 **do**
  $\theta' := 0$;
  for each model $M \in \mathcal{M}_i$ **do**
    compute P_i for M;
    if $P_i \geq \theta'$ **then**
      $M_{max} := M$, $\theta' := P_i$;
    end if
  end for
  compute P_0 for M_{max};
  if $P_0 \geq \theta$ **then**
    $M_{best} := M_{max}$, $\theta = P_0$;
  end if
  for each model $M \in \mathcal{M}_i$ **do**
    if $P_i \geq \theta$ **then**
      add M to $\mathcal{M}_{i-1}$, subtract M from $\mathcal{M}_i$;
    end if
  end for
  $\mathcal{M}'_i := \mathcal{M}_i$;
end for
$M'_{best} := M_{best}$;
return M_{best};

Algorithm 3: Algorithm for monitoring data streams.

5.1.1 Search algorithm

Algorithm 3 depicts the proposed approach to data stream processing. In this figure, $\mathcal{M}_i$ represents the set of models for which it computes 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, $\mathcal{M}'_h$ is added to

$\mathcal{M}_h$. 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. The proposed approach can handle range queries by applying a search threshold as θ to the above algorithm. For K-nearest neighbor queries, the proposed approach can efficiently select the initial candidate to prune models by computing the exact likelihoods of the K models at the previous time tick.

5.2 Experimental evaluation and discussion

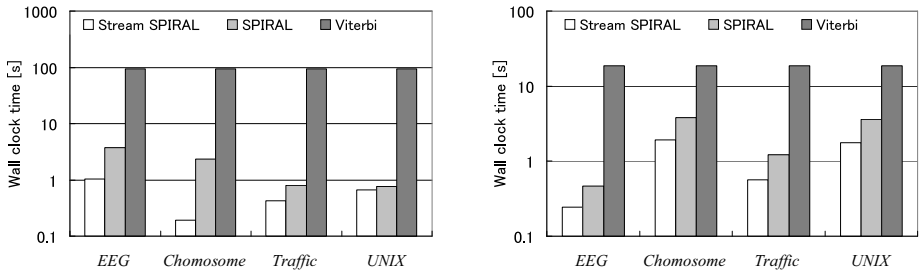
I performed experiments to demonstrate SPIRAL's effectiveness. I compared SPIRAL to the Viterbi algorithm. I refer to the Viterbi algorithm implementation as *Viterbi* hereafter.

5.2.1 Experimental data and environment

I used the following four standard datasets in the experiments; *EEG*, *Chromosome*, *Traffic*, and *UNIX* same as Section 4.2.

The models were trained by the Baum-Welch algorithm [Levinson 82]. In experiments, sequence length is 256 and possible transitions of left-right model are restricted only to 2 states, which is typical in many applications.

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



(a) Ergodic (b) Left-right

Figure 5.2: Wall clock time of monitoring data stream.

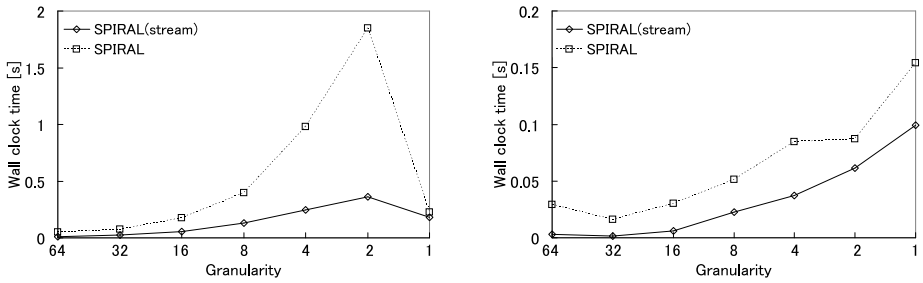
5.2.2 Results of monitoring data stream

I conducted several experiments to show the effectiveness of the proposed approach for monitoring data stream.

Search cost

Figure 5.2 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. The proposed 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. I 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. The proposed approach for data stream processing is based on the this observation, and its effectiveness is confirmed in Figure 5.2.



(a) Ergodic (b) Left-right

Figure 5.3: Breakdown of search cost.

Effectiveness of the data stream algorithm

The proposed stream algorithm automatically changes the granularity and effectively sets the initial candidate to find the best model. To show the effectiveness of these ideas, I plot the wall clock time at each granularity for the two versions of SPIRAL. Figures 5.3 show the breakdown 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.

5.2.3 Stream monitoring with dynamically changing models

It has been 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, the proposed approach can index these data structures by simply storing pointers to the structures, since the proposed search algorithms 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 07] is effective since the model parameters are updated for one data set at a time. The degenerate data structures are needed to be updated according to the trained model, but these structures can be updated 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.

5.3 Summary

This chapter 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 data streams. I presented the approach which effectively set the initial candidate model and approximation granularity. Experiments show that the proposed approach works as expected, and finds high-likelihood HMMs at high speed for data streams.

Chapter 6

Efficient Centrality Monitoring for Time-evolving Graphs

Graphs arise naturally in a wide range of disciplines and application domains, since they are an intuitive abstraction that can naturally capture data entities as well as the relationship among those entities. Data entities are represented as nodes in the graph and edges capture the relationships between data entities. Therefore, graph theory has been one of the major studies in mathematics and computer science since the publication in 1736 of the seminal paper written by Leonhard Euler.

HMM is a probabilistic graphical model which is very useful to analyze sequential data. A probabilistic graphical model comprises nodes connected to by edges. In a probabilistic graphical model, each node represents a random variable, and the edges express probabilistic relationships between these variables. And a probabilistic graphical model then captures the way in which the joint distribution over all of the random variables can be decomposed into a product of factors each depending only a subset of the variables.

In the previous chapters, I presented the several approaches to compute the likelihood for HMMs. This naturally induce the following

question "It is possible to apply the approaches for graphical data?". The aim of this chapter is to present the answer for the question; I show the solution for the problem of finding the lowest centrality node from time-evolving graphs efficiently in this chapter.

6.1 Introduction

In graph theory, the facility location problem is quite important since it involves finding good locations for one or more facilities in a given environment. In this problem, it is preferable to find the nodes whose distances to other nodes is the shortest in the graph, since the cost it takes to reach all other nodes from the nodes are expected to be low. In graph analysis, the centralities based on this concept are **closeness** and **eccentricity**. In this chapter, the closeness centrality of node u, C_u, is defined as the sum of distances from the node to other nodes. And the eccentricity centrality node u, E_u, is defined as the maximum distance from the node to other nodes.

A naive approach to the computation of centrality is based on breadth-first search (BFS). However it is not practical for large-scale graphs since it requires excessive CPU time. The previous approximate approaches, such as the annotation approach [Rattigan 06] and the embedding approach [Potamias 09, Ng 02], can estimate centralities efficiently. These approaches have the advantage of speed over BFS-based schemes at the expense of exactness. However, approximate algorithms are not adopted by many practitioners. This is because the optimality of the solution is not guaranteed; it is hard for approximate algorithms to identify the lowest centrality node. Furthermore, the focus of traditional graph theory has been limited to just 'static' graphs; the implicit assumption is that nodes and edges never change. Recent years have witnessed a dramatic increase in the availability of graph datasets that comprise many thousands and sometimes even millions of time-evolving nodes; a consequence of the widespread availability of

electronic databases and the Internet. Recent studies on large-scale graphs are discovering several important principles of time-evolving graphs [Newman 03, Leskovec 07]. Thus demands to the analysis of time-evolving graphs are increasing. I address the following problem in this chapter:

Problem 4. *Given graph $G[t] = (V, E)$ at time t where V is a set of nodes and E is a set of edges, find the nodes whose closeness centrality are the lowest, and the nodes whose eccentricity centrality are the lowest, both in graph $G[t]$.*

To the best of my knowledge, the proposed approach is the first solution to achieve both exactness and efficiency at the same time in identifying the lowest centrality node from time-evolving graphs.

6.1.1 Contributions

I propose a novel method called Sniper that can efficiently identify the lowest centrality node in time-evolving graphs. In order to reduce search cost, (1) the original graph size is reduced to compute approximate centrality, and (2) high-centrality nodes are pruned by terminating unnecessary distance computations early. Sniper has the following attractive characteristics based on the above ideas:

- **Exact:** Sniper does not sacrifice accuracy even though it exploits an approximate approach to prune unlikely nodes; it returns the lowest centrality node without error although the previous approximate approaches do not guarantee the exactness.

- **Efficient:** Sniper requires just $O(n^2 + nm)$ time where n and m are the number of nodes and edges, respectively. However solutions based on the existing approximate algorithms are expensive for large-scale graphs; they need $O(n^3)$ time to find the lowest centrality node.

- **Nimble:** The required memory space of Sniper is smaller than that of the previous approximate approaches; Sniper needs just $O(n + m)$ space although the previous approaches needs $O(n^2)$ space.

6.1.2 Problem motivation

The problem tackled in this chapter must be overcome to develop the following important applications.

Social networks

Networks of interaction have been studied for a long time by social science researchers, where nodes correspond to people or organizations, and edges represent some type of social interaction. The question of 'which is the most important node in a network?' is being avidly pursued by scientific researchers. An important example is the network obtained by considering scientific publications. Nodes in this case are researcher, papers, book, or entire journals, and edges correspond to co-authorship or citations. This kind of network generally grows very rapidly over time. For example, the collaboration network of scientists in the database area contains several tens of thousands of authors and its rate of growth is increasing year by year; there are several thousand new authors each year [Elmacioglu 05]. The systematic construction of such networks was introduced by Garfield, who later proposed a measure of standing for journals that is still in use. This measure, called impact factor, is defined as the number of citations per published item [Garfield 72]. Basically, the impact factor is a very simple measure, since it corresponds to the degree of the citation network.

However the degree is a local measure, because the value is only determined by the number of its adjacent nodes. That is, if a high-degree node lies in an isolated community of the network, the influence of the node is very limited.

Closeness centrality is a global centrality measure since it is computed by summing the distances to all other nodes in a graph. Therefore, it is an effective measure of influence on other nodes. The most influential node can be effectively detected as the lowest closeness centrality node by monitoring time-evolving graphs. Nascimento et al. analyzed SIGMOD's co-authorship graph [Nascimento 03] *. They successfully discovered that L. A. Rowe, M. Stonebraker, and M. J. Carey were the most influential researchers from 1986 to 1988, 1989 to 1992, and 1993 to 2002, respectively. All these three are very famous and important researchers in the database community.

P2P sensor networks

With the introduction of low-cost processors, memory, and radio technologies, it has become possible to build inexpensive wireless microsensor nodes and thus high quality, fault-tolerant P2P sensor networks. These networks can be used to collect useful information from an area of interest, especially where the physical environment is so harsh that a sensor might fail at any time.

Even though many P2P sensor network models have been proposed, their basic characteristics are common as follows: They are all composed of a large number of sensor nodes, and a small number of master nodes. All sensor nodes perform relatively limited computational operations. The master nodes collect the data from all sensors, and analyze/process the data. They are also the managers of the network. One of the main system design considerations is energy conservation, because it limits node lifetime and thus the quality of the network.

It can be observed that the nodes that have the lowest eccentricity centrality score are expected to have the short distances to all other nodes assuming that a sensor and a communication link between sensors are represented as a node and an edge, respectively. Therefore, it

*They examined the co-authorship graph for not only SIGMOD but also PODS, VLDB and ICDE. The latest results can be seen at http://db.cs.ualberta.ca/coauthorship/

is appropriate to designate the lowest eccentricity nodes as the master nodes. Based on this idea, Wang proposed an energy-efficient data collection approach for P2P sensor networks [Wang 06]. They demonstrated that their approach can considerably reduce the power needed to collect data from sensors.

While time-evolving graphs are potentially useful in many applications, they have been difficult to utilize due to their high computational costs. However, by providing exact solutions in a highly efficient manner, Sniper will allow many more data mining applications based on time-evolving graphs to be developed in the future.

6.2 Related work

The structural properties of real world networks have been investigated and shown to have several interesting properties. Researchers of data engineering have published many papers on node-to-node distance computation or time-evolving graphs.

6.2.1 Network Science

Recent studies on large graph datasets have shed light on several important network phenomena; most importantly, how the structure of the network itself evolves over time.

One phenomenon, rooted in early work in social sciences, is preferential attachment; in this phenomenon, nodes that already have many edges will tend to acquire them at a greater rate than others [Newman 03]. One active line of research has shown how preferential attachment can lead to the highly skewed distributions of edges that one sees in real network, with certain nodes acting as highly connected hubs [Reka 02].

Another phenomenon, also a key issue in sociology, is the notion of triadic closure; edges are much more likely to form between two nodes when they share a third node [Rapoport 53]. Recent work on email

logs has provided some of the first concrete measurements of the effect of this principle in a social-communication network [Kossinets 06].

Additional phenomena have begun to emerge from recent studies of social and information networks, including the densification effect. In this phenomenon, the number of edges per node increases as the network grows. Diameters, the number of maximum distances between any two nodes, decrease in a graph even as the total number of nodes increases [Leskovec 07].

6.2.2 Data Engineering

Many papers have been published on approximation for node-to-node distances. The previous distance approximation schemes are distinguished into two types: annotation approach and embedding approach. Rattigna et al. studied two annotation schemes [Rattigan 06]. They randomly select nodes in a graph and divide the graph into regions that are connected, mutually exclusive, and collectively exhaustive. They give a set of annotations to every node from the regions. Distances are computed by the annotations. They demonstrated their method can compute node distances more accurately than the embedding approaches. However, this method can require $O(n^2)$ space and $O(n^3)$ time to estimate the lowest centrality nodes as described in their paper.

The Landmark approach is an embedding approach [Goldberg 05, Potamias 09], and estimates node-to-node distance from selected nodes at $O(n)$ time. The minimum distance via a landmark node is utilized as node distance in this method. Another embedding approach is Global Network Positioning which was studied by Ng et al [Ng 02]. Node distances are estimated with L_p norm between node pairs. These embedding approaches require $O(n^2)$ space since all n nodes hold distances to $O(n)$ selected node. Moreover, they require $O(n^3)$ time to identify the lowest centrality node since $O(n^2)$ time is needed to compute estimate centrality for all n nodes.

Sun et al. applied tensor analysis to time-evolving graphs to detect graph patterns [Sun 06]. A static graph can be expressed using adjacency-matrix representation; this implies that a static graph is two dimensional. Even though SVD/PCA can represent such 2D matrices compactly, it is not effective for 3D time-evolving graphs. Their approach represents the original tensor as several small tensors and a matrix. The matrix can be dynamically updated for time-evolving graphs.

In subsequent work by Sun et al., they encoded the original time-evolving graphs by lossless compression and effectively detected communities in graphs [Sun 07].

6.3 Preliminary

In this section, I introduce the background to this chapter. Social networks and others can be described as graph $G = (V, E)$, where V is the set of nodes, and E is the set of edges. n and m are used to denote the number of nodes and edges, respectively. That is $n = |V|$ and $m = |E|$. A path from node u to v as the sequence of nodes linked by edges, beginning with node u and ending at node v. A path from node u to v is the shortest path if and only if the number of nodes in the path is the smallest possible among all paths from node u to v. Distance between node u and v, $d(u, v)$, is the number of edges in the shortest path connecting them in a graph. Therefore $d(u, u) = 0$ for every $u \in V$, and $d(u, v) = d(v, u)$ for $u, v \in V$.

The closeness centrality of node u, C_u, is the sum of the distances from the node to any other node, and computed as follows:

$$C_u = \sum_{v \in V} d(u, v) \tag{6.1}$$

The eccentricity centrality of node u, E_u, is the maximum distance from the node to any other node, and computed as follows:

$$E_u = \max\{d(u, v) : v \in V\} \tag{6.2}$$

Table 6.1: Definition of main symbols.

Symbols	Definitions
V	Set of nodes in graph G
E	Set of edges in graph G
n	Number of nodes
m	Number of edges
t	Time stamp, $t \geq 1$
$d(u, v)$	Distance from node u to v
C_u	Closeness centrality of node u
N_u	Neighbors of node u

The aims of closeness and eccentricity centrality are to identify the node that minimizes the total distance (i.e., the average distance) and the maximum distance to any other node in a graph, respectively.

The closeness and eccentricity centrality of node u can be naively computed by the approach of BFS, which is often used for searching a graph. Given graph $G = (V, E)$ and source node u, BFS systematically explores every node that is reachable from node u. It computes the distance from node u to each reachable node. To compute the closeness centrality of node u, the naive approach exploits BFS from node u at the first step to compute distances to all other nodes, and then computes the sum of the distances in the next step. Similarly, the eccentricity centrality of node u can be computed by BFS by simply exploring all other nodes to compute the maximum distance. However, 'computing shortest paths among all node pairs is computationally prohibitive' as described in [Leskovec 07].

6.4 Centrality monitoring

In this section, I explain the two main ideas and introduce Sniper. The main advantage of Sniper is to exactly and efficiently identify the lowest closeness and eccentricity centrality nodes in time-evolving graphs.

I focus on identifying the lowest closeness centrality nodes in this section, and discuss how to detect the lowest eccentricity nodes in Section 6.4.6. And I focus on undirected and unweighted graphs in this section. However, the proposed approach can be applied to weighted or directed graphs as described in Section 6.4.6. Moreover, the proposed approach can handle range queries (find the nodes whose centralities are less than a given threshold) and K-best queries (find the K lowest centrality nodes) as described in Section 6.4.6. It is assumed that no two nodes will have exactly the same centrality value and one node is added to a time-evolving graph at each time tick. These assumptions can be eliminated easily. Table 6.1 lists the main symbols and their definitions.

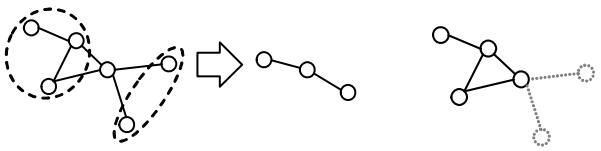
6.4.1 Ideas behind Sniper

The proposed solution is based on the two ideas described below.

Node aggregation I introduce approximations to reduce the high cost of the existing approaches. Instead of computing the exact centrality of every node, the centrality is approximated, and high-centrality nodes are efficiently pruned.

For a given graph with n nodes and m edges, an approximate graph of n' nodes and m' edges ($n' < n, m' < m$) is created by aggregating 'similar' nodes in the graph (see Figure 6.1). For the approximate graph, $O(n' + m')$ time is required for Sniper to compute the approximate centralities, while the existing approximate algorithm requires $O(n^2)$ time as described in Section 6.2. The Jaccard coefficient is exploited to find similar nodes, and then aggregate the original nodes to create node groups. I refer to such groupings as **aggregate** nodes.

This new idea has the following two major advantages. First, the answer node can be found exactly; the node that has the lowest centrality is never missed by this approach. This is because the proposed approximate graphs guarantee the lower bounding distances, i.e. ap-



(1) Node aggregation (2) Tree estimation

Figure 6.1: Basic ideas behind Sniper.

proximate centrality. This means that unpromising nodes can be safely discarded at low CPU cost. The second advantage is that this idea can reduce the number of nodes that must be processed to compute centralities, as well as reducing the computation cost for a single node. That is, the lowest centrality node can be identified among a large number of nodes efficiently.

Tree estimation Although the proposed approximation technique is able to discard most of the unlikely nodes, exact centrality computation must be used to guarantee the correctness of the search results. Here I focus on reducing the cost of this computation.

To compute the exact centrality of a node, distances to all other nodes from the node have to be computed by BFS. But clearly the exhaustive exploration of nodes in a graph is not computationally feasible, especially for large graphs. The proposal exploits the following idea: If a node cannot be the lowest centrality node, subsequent distance computations are terminated as being unnecessary.

The proposed search algorithm first holds a candidate node, which is expected to have low centrality. The distances of unexplored nodes are also estimated in the distance computation from a single BFS-tree to obtain the lower centrality bound. In the search process, if the lower centrality bound of a node gives a value larger than the exact centrality of the candidate node, the node cannot be the lowest centrality node in the original graph. Accordingly, unnecessary distance computations

can be terminated early.

With this estimation, I do not need to exhaustively explore all node pairs in a graph, so the computation cost is less than that is demanded by the BFS-based algorithm.

This technique can be applied to approximate centrality computation as well as exact computation. This means that the approximate centrality can be computed more efficiently.

6.4.2 Node aggregation

The first idea involves aggregating nodes of the original graph, which enables to compute the lower centrality bound to realize reliable node pruning.

Graph approximation

The original graph size is reduced in order to compute approximate centralities at low computation cost. To realize efficient search, given original graph G with n nodes and m edges, this approach computes n' nodes and m' edges in the approximate graph G'. That is, the original graph $G = (V, E)$ is collapsed to yield the approximate graph $G' = (V', E')$.

I first describe how to compute the edges of the approximate graph, and then show the proposed approach to aggregate original nodes.

For the aggregate nodes u' and v', there is an edge, $\{u', v'\} \in E'$, if and only if there is at least one edge between aggregated original nodes in u' and v'. Moreover, the approximate graph has no self-loop edge. These definitions are important for computing the lower centrality bound. Formally, the edges between aggregate node u' and v' are obtained as follows:

Definition 1 (Node aggregation). *In the approximate graph G', node u' and v' have an edge if and only if:*

$$(1)u' \neq v', \quad (2)\exists\{u, v\}, u \in u' \cap v \in v' \tag{6.3}$$

where $u \in u'$ indicates that aggregate node u' contains original node u.

To reduce the approximation error, similar nodes are aggregated. As described above, the aggregate nodes share an edge if and only if there is at least one edge between the original nodes that have been aggregated. Therefore, the approximation error decreases as the number of neighbors shared by the aggregated nodes increases. For this reason, the Jaccard coefficient is utilized since it is a simple and natural measure of similarity between sets [Broder 97]. The proposed approach can utilize other similarity measures such as the Dice and Simpson coefficients [†], but this choice has very little impact as I demonstrate in Section 6.5

Let N_u and N_v be neighbors (adjacent nodes) of nodes u and v, respectively; the Jaccard coefficient is defined as $|N_u \cap N_v|/|N_u \cup N_v|$, i.e. the size of the intersection of the sets divided by the size of their union. Node u and v are aggregated if the most similar node of u is node v, this yields good approximation as later demonstrated in Section 6.5. Note, nodes u and v are not aggregated if the size of their intersection is less than one half the size of their union to avoid aggregating dissimilar nodes.

If one node is added to a time-evolving graph, the most similar node is computed to update the approximate graph. The naive approach to compute the most similar node for the added node is to compute the similarities for all nodes. On the other hand, the following lemma is utilized to efficiently update the most similar node:

Lemma 7 (Update the most similar nodes). *For the added node, the most similar node is at most two hops apart.*

Proof. For two nodes having no common neighbor, the similarity coefficient is 0 by definition. Because of the presence of a two-hop path through a common neighbor, pairs with a positive similarity coefficient must be at most two hops apart. □

[†]The Dice and Simpson coefficients are defined as $2|N_u \cap N_v|/(|N_u| + |N_v|)$ and $|N_u \cap N_v|/\min(N_u, N_v)$, respectively.

Input: $G[t] = (V, E)$, a time-evolving graph at time t
u_{add}, the added node at time t
Output: $G'[t] = (V', E')$: the approximate graph.

```
1: compute distances from u_add;
2: for each node v s.t. d(u_add, v) ≤ 2 do
3:    compute the similarity coefficient;
4:    update the most similar node;
5: end for
6: for each node v ∈ V do
7:    aggregate node v to the most similar node;
8: end for
9: for each node v' ∈ V' do
10:   if v' ≠ w' ∩ (∃{v, w}, v ∈ v' ∩ w ∈ w') then
11:      Link edge for w';
12:   end if
13: end for
14: return G'[t];
```

Algorithm 4: Update

Note that this lemma holds for not only the Jaccard coefficient but also the Dice and Simpson coefficients.

The approximate graph is efficiently updated with the above lemma. Algorithm 4 shows the update algorithm for approximate graphs. This algorithm is based on Lemma 7. It computes the distances of one and two hops apart nodes from the added node by BFS (line 1). For each obtained node, it computes similarity for the added node and update the most similar node (lines 2-5). It merges similar original nodes to obtain the aggregate nodes (lines 6-8). It links the aggregate nodes with Definition 1 (lines 9-13).

Even though it is assumed that a single node is added for time-evolving graphs in each time tick, Lemma 7 can also be applied for the case of single node deletion. The above procedure is iterated for each node if several nodes are added. If one edge is added/deleted, one connected node is deleted and added the node.

Lower bounding centrality

Given an approximate graph, approximate centrality of node u' is computed as follows:

Definition 2 (Approximate closeness centrality). *For the approximate graph, the approximate closeness centrality of node u', $C_{u'}$, is computed as follows:*

$$C_{u'} = \sum_{v' \in V'} \{d(u', v') \cdot |v'|\} \tag{6.4}$$

where $|v'|$ is the number of original nodes aggregated within node v'.

I can provide the following theorem about the centrality approximation:

Lemma 8 (Approximate closeness centrality). *For any node in the approximate graph, the following inequality holds.*

$$C_{u'} \leq C_u \tag{6.5}$$

Proof. I first prove that $d(u', v') \leq d(u, v)$ holds for any node in the approximate graph. Let $u \rightsquigarrow v$ be the shortest path between nodes u and v of the original graph, and $w_i \rightsquigarrow w_j$ be the sub-path of the shortest path. If all nodes on the shortest path are aggregated into distinct nodes, the corresponding path length in the approximate graph is equal to that of the shortest path. This is because the shortest path of approximate graph $u' \rightsquigarrow w'_i \rightsquigarrow w'_j \rightsquigarrow v'$ completely matches the shortest path of the original graph, $u \rightsquigarrow w_i \rightsquigarrow w_j \rightsquigarrow v$. Otherwise, there are at least two original nodes that are aggregated into one node. That is, the nodes of the original graph, w_i and w_j, are aggregated into node w'. Therefore, the shortest path of the original graph, $u \rightsquigarrow w_i \rightsquigarrow w_j \rightsquigarrow v$, is shortened to $u' \rightsquigarrow w' \rightsquigarrow v'$. Therefore, $d(u', v') \leq d(u, v)$.

For all aggregate node u', the following inequality holds from the above property:

$$d(u', v') \cdot |v'| \leq \sum_{v \in v'} d(u, v) \tag{6.6}$$

Therefore, the following inequality holds:

$$C_{u'} = \sum_{v' \in V'} \{d(u', v') \cdot |v'|\} \leq \sum_{v \in V} d(u, v) = C_u \tag{6.7}$$

which completes the proof. □

Lemma 8 provides Sniper with the property of finding the exact answer as is described in Section 6.4.5.

6.4.3 Tree estimation

I introduce an algorithm for computing original centralities efficiently. This approach terminates subsequent distance computations from a node if the estimate centrality of the node is larger than the exact centrality of the candidate node. In this approach, lower bounding distances of unexplored nodes via BFS are computed to estimate the lower centrality bound of a node. Estimations are obtained from a single BFS-tree.

Notation

I first give some notations for the estimation. In the search process, this approach constructs the BFS-tree rooted at a selected node. As a result, the selected node forms layer 0. The direct neighbors of the node form layer 1. All nodes that are i hops apart from the selected node form layer i. The approach to selecting the node is later described.

Next, this approach checks by BFS that the exact centralities of other nodes in the tree are lower than the exact centrality of the candidate node. The set of nodes explored by BFS is defined as V_{ex}, and the set of unexplored nodes as $V_{un} (= V \setminus V_{ex})$. $d_{max}(u)$ is the maximum distance of the explored node from node u, that is $d_{max}(u) = \max\{d(u, v) : v \in V_{ex}\}$. Moreover, the explored layer of the tree is defined as L_{ex} if and only if there exists at least one explored node in the layer. Similarly the unexplored layer is defined as L_{un} if

and only if there exists no explored node in the layer. The layer number of node u is denoted as $l(u)$.

Centrality estimation

How to estimate the centrality of a node is defined in this section. The closeness centrality of node u via BFS is estimated as follows:

Definition 3 (Estimate closeness centrality). *For the original graph, the following estimate centrality of node u, $\hat{C}_u$, is defined to terminate distance computation in BFS:*

$$\hat{C}_u = \sum_{v \in V_{ex}} d(u,v) + \sum_{v \in V_{un}} e(u,v) \tag{6.8}$$

$$e(u,v) = \begin{cases} d_{max}(u) & (v \in V_{un} \cap L_{ex}) \\ d_{max}(u) + \min\{|l(v) - l(w)|\} - 1 & (v \in L_{un}, w \in L_{ex}) \end{cases}$$

The estimation is the same as the exact centrality if all nodes are explored (i.e. $V_{ex} = V$) in Equation (6.8). To show the property of estimate centrality, I introduce the following lemma:

Lemma 9 (Estimate closeness centrality). *The following inequality holds for the original graph in BFS.*

$$\hat{C}_u \leq C_u \tag{6.9}$$

Proof. I first prove that distance of unexplored node v in an explored layer cannot shorter than d_{max}. In BFS, all distances are computed from already explored nodes, where exploration starts from a source node. This procedure ensures that distances of unexplored nodes are monotonic non-decreasing in BFS.

I next prove that the distance of unexplored node v in an unexplored layer cannot be shorter than $d_{max} + \min(|l(v) - l(w)|) - 1$ where node w is in an explored layer. If the one upper/lower layer is explored for the unexplored layer, distance of node v cannot shorter than d_{max} because distances of unexplored nodes cannot be shorter than d_{max}.

Moreover, if two upper/lower layer is explored and the one upper/lower layer is unexplored, distance of node v cannot shorter than $d_{max} + 1$. This is because the distance of the one upper/lower layer nodes cannot be shorter than d_{max} and the unexplored layer cannot be directly connected to nodes in more than two upper/lower layers. Similarly, if the i upper/lower layer is explored and the 1 to $i-1$ upper/lower layer are all unexplored, the distance of node v cannot be shorter than $d_{max} + i - 1$. This completes the proof. □

This property enables Sniper to identify the lowest centrality node exactly.

The selection of the root node of the tree is important for efficient pruning. The lowest centrality node of the previous time tick is selected as the root node. There are two reasons for this approach. The first reason is that this node and nearby nodes are expected to have lowest centrality value, and thus are likely to be the answer node after node addition. In the case of time-evolving graphs, small numbers of nodes are continually being added to the large number of already existing nodes. Therefore, there is little difference between the graphs before and after node addition. In addition, the centrality value of a node can be more accurately estimated if the node is close to the root node; this is the second reason. This is because the proposed estimation scheme is based on the distances from the root node.

Algorithm 5 shows the algorithm for the centrality estimation for a source node. It exploits the exact centrality of the candidate node as θ in Algorithm 5. It excludes the unlikely nodes in the graph by using θ. That is, if the estimated centrality of a node is larger than θ, that node cannot be the lowest centrality node, and so can be safely discarded (lines 6-8). Note that the estimation yields the exact centrality if all nodes are explored in this algorithm, therefore it returns $\hat{C}_u$ at the end of Algorithm 5 (line 10).

This algorithm to approximate graph computation can be similarly applied for exact computation.

Input: $G = (V, E)$, a graph
u, a source node
θ, exact centrality of the candidate node
Output: $\hat{C}_u$, estimate centrality

```
1: V_ex ← empty set;
2: while V_ex ≠ V do
3:    compute distance of node v, d(v, u), by BFS;
4:    append node v → V_ex;
5:    compute the estimation Ĉ_u;
6:    if Ĉ_u > θ then
7:       return Ĉ_u;
8:    end if
9: end while
10: return Ĉ_u;
```

Algorithm 5: Centrality estimation

6.4.4 Search algorithm

The main approach to finding the lowest centrality node is to prune unlikely nodes by using the proposed approximation, and then confirm by exact centrality computations whether the viable nodes are the answer. However, an important question is which node should be selected as the candidate in time-evolving graphs. The previous lowest centrality node is selected as the candidate. This node likely to have lowest centrality. After the BFS-tree construction, the exact centrality can be directly obtained with this approach.

The proposed search algorithm is outlined as follows:

1. It updates the approximate graph for node addition.
2. It constructs a BFS-Tree rooted the candidate node.
3. If the approximate centrality of a node is higher than the centrality of the candidate node, it prunes the node.
4. For viable nodes, it computes the exact centrality to confirm the answer node.

Input: $G[t] = (V, E)$, a time-evolving graph at time t
u_{add}, the added node at time t
$u_{low}[t-1]$, the previous lowest centrality node
Output: $u_{low}[t]$: the lowest centrality node.

```
1: //Update the approximate graph
2: update the approximate graph by the update algorithm;
3: //Search the lowest centrality node
4: V_exact ← empty set;
5: compute the BFS-tree of node u_low[t − 1];
6: compute θ, the exact centrality of node u_low[t − 1];
7: for each node v' ∈ V' do
8:    compute C_v' by the estimation algorithm;
9:    if C_v' ≤ θ then
10:      for each node v ∈ v' do
11:         append node v → V_exact;
12:      end for
13:   end if
14: end for
15: for each node v ∈ V_exact do
16:   compute C_v by the estimation algorithm;
17:   if C_v < θ then
18:      θ ← C_v;
19:      u_low[t] ← v;
20:   end if
21: end for
22: return u_low[t];
```

Algorithm 6: Sniper

Algorithm 6 shows the search algorithm that targets the lowest closeness centrality node. In this algorithm, $u_{low}[t]$, $u_{low}[t-1]$ and u_{add} indicate the lowest centrality node, the previous lowest centrality node, and the added node, respectively. V_{exact} represents the set of nodes for which it computes exact centralities.

The algorithm can be divided into two phases: update and search. In the update phase, Sniper computes the approximate graph by the update algorithm (line 2). In the search phase, Sniper first computes the BFS-tree of the answer node of the last time tick (line 5) and θ (line 6). If the approximate centrality of a node is larger than θ, It prunes

the node since it cannot be the lowest centrality node. Otherwise, Sniper appends aggregated original nodes to V_{exact} (lines 9-13), and then computes exact centralities to identify the lowest centrality node (lines 15-21).

Note, the search algorithm for the lowest eccentricity centrality node is only a minor variant of the algorithm used in computing approximate and estimate centralities. The proposed approximation and estimation approaches for eccentricity centrality are described in Section 6.4.6.

6.4.5 Theoretical Analysis

This section provides theoretical analyses that confirm the accuracy and complexity of Sniper. Note that the theoretical analyses cover both forms of centrality; closeness centrality and eccentricity centrality. Let n be the number of nodes and m the number of edges.

I prove that Sniper finds the lowest centrality node accurately (without fail) as follows:

Theorem 3 (Find the lowest centrality node). *Sniper guarantees the exact answer when identifying the node whose centrality is the lowest.*

Proof. Let u_{low} be the lowest centrality node in the original graph, and θ_{low} be the exact centrality of u_{low} (i.e., θ_{low} is the lowest centrality). Also let θ be the candidate centrality in the search process.

In the approximate graph, since $\theta_{low} \leq \theta$, the approximate centrality of node u_{low} is never upper than θ (Lemma 8). Similarly, in the original graph, the estimate centrality of node u_{low} is never upper than θ (Lemma 9). The algorithm discards a node if (and only if) its approximate or estimated centrality is upper than θ. Therefore, the lowest centrality node u_{low} cannot be pruned during the search process. □

I then discuss the complexity of Sniper. Note that the previous approaches need $O(n^2)$ space and $O(n^3)$ time to compute the lowest centrality node.

Theorem 4 (Complexity of Sniper). *Sniper requires $O(n+m)$ space and $O(n^2 + nm)$ time to compute the lowest centrality node.*

Proof. I first prove that Sniper requires $O(n + m)$ space. Sniper keeps the approximate graph and the original graph. In the approximate graph, since the number of nodes and edges are at most n and m, respectively, Sniper needs $O(n + m)$ space for the approximate graph; $O(n + m)$ space is required for keeping the original graph. Therefore, the space complexity of Sniper is $O(n + m)$.

Next, I prove that Sniper requires $O(n^2+nm)$ time. To identify the lowest centrality node, Sniper first updates the approximate graph and then computes approximate and exact centralities. Sniper needs $O(nm)$ time to update the approximate graph, since it requires $O(m)$ time to compute similarity for the added node against each node in the original graph. It requires $O(n^2 + nm)$ time to compute the approximate and exact centralities since the number of nodes and edges are at most n and m, respectively. Therefore, Sniper requires $O(n^2 + nm)$ time.

Theorem 4 shows, theoretically, Sniper requires less order of space and time complexities than the previous approximate approaches. In practice the search cost depends on the effectiveness of the approximation and estimation techniques used by Sniper. In the next section, I show the effectiveness of the proposed approach by presenting the results of extensive experiments.

6.4.6 Extension

In this section, I give a discussion of some extensions to Sniper.

Supporting eccentricity centrality

In Sections 6.4, I explained how to find the lowest closeness centrality node. In this section, I describe how Sniper can also identify the lowest eccentricity centrality node.

For the approximate graph, the approximate eccentricity centrality is defined as follows:

Definition 4 (Approximate eccentricity centrality). *For the approximate graph, the approximate eccentricity centrality of node u', $E_{u'}$, is computed as follows:*

$$E_{u'} = \max\{d(u', v') : v' \in V'\} \tag{6.10}$$

This approximation has the following property:

Lemma 10 (Approximate eccentricity centrality). *For any node in the approximate graph, the following inequality holds.*

$$E_{u'} \leq E_u \tag{6.11}$$

Proof. This is obvious because the approximate graph has the lower bounding distance property as described in the proof of Lemma 8. □

The eccentricity centrality of node u is estimated in BFS as follows:

Definition 5 (Estimate eccentricity centrality). *For the original graph, the following estimate centrality of node u, $\hat{E}_u$, is defined to terminate distance computation in BFS:*

$$\hat{E}_u = \max\{d(u, v), e(u, w) : u \in V_{ex}, u \in V_{un}\} \tag{6.12}$$

I show the following theorem to introduce the lower bounding property of node estimation; this property enables Sniper to identify the lowest eccentricity centrality node exactly:

Lemma 11 (estimate eccentricity centrality). *The following inequality holds for the original graph in BFS.*

$$\hat{E}_u \leq E_u \tag{6.13}$$

Proof. This is obvious from the proof of Lemma 9. □

Directed or Weighted graphs

I have focused on undirected and unweighted graphs in this chapter, but Sniper can also handle directed or weighted graphs effectively. Approximate graphs have an edge if and only if there is at least one edge between aggregated nodes in an undirected and unweighted graph. However, how approximate graphs are constructed must be modified to handle other kinds of graphs.

For directed graphs, Definition 1 is applied for each direction to handle directed edges of approximate graphs. For weighted graphs, the lowest value of the weights of the original edges is chosen as the weight of the aggregated edge to compute the lower bound of exact centralities.

To estimate centrality values for weighted graphs, Definition 2 can be directly applied. But for weighted graphs, a little modification is needed. Distance from node u to v is estimated as $d_{max}(u) + \min\{\omega(v, w) : w \in V \setminus v\}$ where $\omega(v, w)$ is the weight of edge $\{v, w\}$.

Other types of queries

Although the search algorithm described here identifies the node that has the lowest centrality, the proposed approach can be applied to range queries and K-best queries. Range queries find the nodes whose centralities are less than a given threshold, while K-best queries find the K lowest centrality nodes.

Sniper first computes the exact centrality of the candidate node, θ, by the answer node in the last time tick. It prunes unlikely nodes by approximate and exact centrality computations. The search algorithms for range and K-best queries basically follow the algorithm for the lowest centrality node. However, there is one important difference in how to obtain θ.

For range queries, I would utilize a given search threshold as θ, instead of the exact centrality of the previous time tick (i.e., I do not use the candidate). Approximate centralities of all nodes are computed and prune unlikely nodes using the given θ; the answer nodes are confirmed

by calculating exact centralities.

For K-best queries, I first compute the exact centralities of all K answer nodes in the last time tick. Next, the K-th lowest exact centrality is selected as θ. Subsequent procedures are the same as for the case of identifying the lowest centrality node.

Processing static graphs

Sniper is mainly designed to find the lowest centrality node for time-evolving graphs. However, in real applications, it can be useful in processing static graphs as well.

There are two questions that need to be resolved to efficiently handle static graphs: (1) 'How to efficiently compute the most similar nodes?' and (2) 'Which node should be selected as the candidate?'.

I do not compute the exact Jaccard coefficient here but estimate the most similar node, and this is the answer for the first question. A simple technique for estimating the Jaccard coefficient using random permutations was proposed by Broder et al. [Broder 97]. I can exploit this technique directly to efficiently compute the most similar nodes. Moreover, Lemma 7 can be utilized to reduce the number of nodes for which similarities need to be calculated.

Although the estimation technique can impact the pruning effectiveness, lower bounding centrality can be still computed. Consequently, I can find the lowest centrality node exactly by the above approach.

The answer for the second question is to select the highest degree node as the candidate. This approach is expected to be more efficient than the method of selecting the candidate at random as is shown in the experimental results (Figure 6.13).

Supporting other centralities

In this chapter, I focused on closeness and eccentricity centrality since they are important in the facility location problem. The concept behind these measures is that a node is preferred if it has short distance to other

nodes.

However many other forms of centralities have been proposed to find critical nodes from graphs, and these are based on different characteristics of graphs. For example, the betweenness centrality is based the shortest paths. This centrality follows the observation: The number of shortest paths that contain a node is a measure of the communications that the node sustains in the graph.

The proposed approaches, such as node aggregation and tree estimation, are mainly designed to find the node whose distances to other nodes in a graph are short, and I cannot straightforwardly apply these approaches to betweenness centrality. Supporting other centralities is a future work.

6.5 Experimental evaluation

I performed experiments to demonstrate Sniper's effectiveness. I compared Sniper to the annotation approaches [Rattigan 06]. In the experiments, I compared the **Zone** annotation scheme and the **Distant to zone** annotation scheme (abbreviated to **DTZ**) to Sniper since they outperform the other embedding schemes in all of the dateset; the same result is reported in [Rattigan 06]. Zone and DTZ annotation have two parameters: zones and dimensions. Zones are divided regions of the entire graph, and dimensions are sets of zones [‡]. Note that these approaches can compute the centrality quickly at the expense of exactness.

Experiments will demonstrate that:

- Efficiency and scalability: Sniper outperforms the annotation approaches by up to 110 times for the real datasets tested. Sniper is scalable to dataset size.

- Exactness: Unlike the existing approaches, which sacrifices accuracy, Sniper can find the lowest centrality node exactly and

[‡]To compute the centralities of all nodes by the annotation approaches, I sampled half pairs from all nodes, which is the same setting used in the paper.

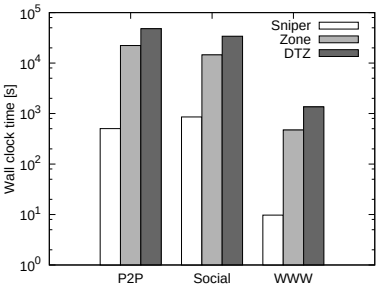


Figure 6.2: Efficiency of Sniper.

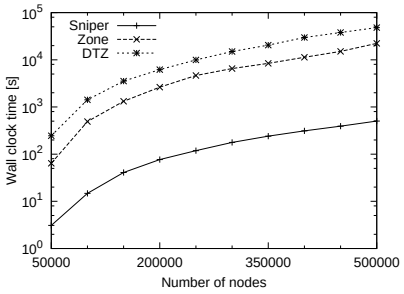


Figure 6.3: Scalability of Sniper.

efficiently.

- Effectiveness: The components of Sniper, node aggregation and tree estimation are effective in identifying the lowest centrality node.

I used the following three public datasets in the experiments: *P2P*, *Social*, and *WWW*. They are a campus P2P network for file sharing, free on-line social network, and web pages within 'nd.edu' domain, respectively. I extracted the largest connected component from the real data, and I added single nodes one by one in the experiments.

I evaluated the search performance through wall clock time. All experiments were conducted on a Linux quad 3.33 GHz Intel Xeon server with 32GB of main memory. I implemented the proposed algorithms using GCC.

6.5.1 Efficiency and scalability of Sniper

I assessed the search time needed for Sniper and the annotation approach. Figure 6.2 shows the efficiency of Sniper where the number of nodes are $500,000$ for P2P and Social, and $100,000$ for WWW. I also show the scalability of the proposed approach in Figure 6.3; this figure

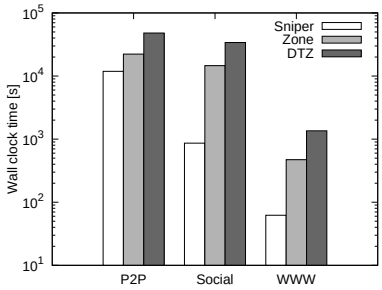


Figure 6.4: Efficiency of Sniper (eccentricity).

shows the wall clock time as a function of the number of nodes. I show the result of P2P in Figure 6.3. These figures indicate Sniper's total processing time (both update and search time are included). I set the number of zones and the dimension parameter to 2 and 1, respectively. Note that, these parameter values allow the annotation approaches to estimate the lowest centrality node most efficiently.

These figures show that the proposed method is much faster than the annotation approaches under all the conditions examined. Specifically, Sniper is more than 110 times faster.

The annotation approaches require $O(n^2)$ time for computing centralities while Sniper requires $O(n' + m')$ time for computing approximate centralities. Even if Sniper computes the approximate centralities of all aggregate nodes to prune the nodes, this cost does not alter the search cost since approximate computations are effectively terminated. Sniper requires $O(n + m)$ time to compute exact centralities for nodes that cannot be pruned through approximation. This cost, however, has no effect on the experimental results. This is because a significant number of nodes are pruned by approximation as shown in later.

Eccentricity centrality

Sniper can find not only the lowest closeness centrality node but the lowest eccentricity centrality node efficiently. I performed the experiments to demonstrate its efficiency in handling eccentricity centrality. I compared Sniper to Zone and DTZ annotation methods. The number of zones and the dimensions parameter are set to 2 and 1, respectively. Figure 6.4 shows efficiency of Sniper and the annotation approaches. In Figure 6.4, the number of nodes are $500,000$ for P2P and Social, and $100,000$ for WWW.

This figure shows Sniper can find the lowest eccentricity centrality node more efficiently than the annotation approaches; it is 40 time faster. The proposed approach needs $O(n' + m')$ to compute approximate centralities while annotation approaches require $O(n^2)$. Due to its successive approximation approach, Sniper avoids computing the exact centralities of all nodes. Furthermore, the estimation approach of Sniper effectively terminates exact and approximate centrality computations. As a result, Sniper can find the lowest eccentricity centrality node more efficiently than previous approaches.

6.5.2 Exactness of the search results

One major advantage of Sniper is that it guarantees the exact answer, but this raises the following simple question: 'How successful is the previous approaches in providing the exact answer even though it sacrifices exactness?'.

To answer this question, I conducted comparative experiments for the annotation approaches. As the metric of accuracy, I measured the error ratio, which is the error centrality value of the estimated lowest centrality node divided by the centrality value of the exact answer node. Figure 6.5 and Figure 6.6 show the error ratio and the wall clock time of the annotation approaches with various parameter settings. The number of nodes is $10,000$ and the dateset used is P2P in these figures.

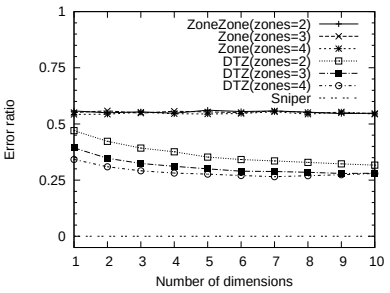


Figure 6.5: Error ratio of each approach.

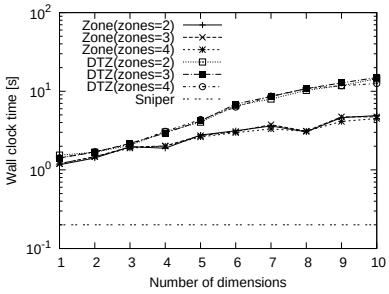


Figure 6.6: Wall clock time of the annotation approaches.

As I can see from Figure 6.5, the error ratio of Sniper is 0 because it identifies the lowest centrality node without fail. The annotation approaches, on the other hand, have much higher error ratios. Therefore, it is not practical to use the annotation approaches in identifying the lowest centrality node. Figure 6.6 shows that Sniper greatly reduces the computation time even though it guarantees the exact answer. The efficiency of the annotation approaches depends on the parameters used.

Furthermore, the results show that the annotation approaches force a trade-off between speed and accuracy. That is, as the number of zones and dimensions parameters decreases, the wall clock time decreases but the error ratio increases. The annotation approaches are approximation techniques and so can miss the lowest centrality node. Sniper also computes approximate centralities, but unlike the annotation approaches, Sniper does not discard the lowest centrality node in the search process. As a result, Sniper is superior to the annotation approaches in not only accuracy, but also speed.

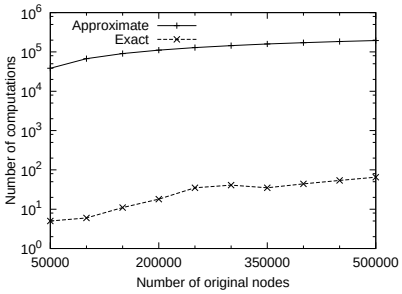


Figure 6.7: Number of centrality computations.

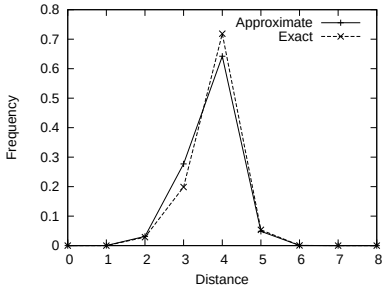


Figure 6.8: Distribution of distances.

6.5.3 Effectiveness of each approach

In the following experiments, I examine the effectiveness of the core techniques of Sniper: node aggregation and tree estimation.

Centrality approximation

Sniper prunes high-centrality (unlikely) nodes using approximations. The number of approximate and exact centrality computations is a factor that influences the search cost. Accordingly, I evaluated the number of approximate and exact computations needed in Sniper. Figure 6.7 shows the number of computations needed where the number of nodes is $100,000$.

As seen in the figure, the number of approximate computations (i.e., the size of approximate graph) are smaller than the number of original nodes. This figure also indicates that Sniper has very strong pruning power; it excludes most of the nodes. Owing to the small size of approximate graphs and the assured approximation quality, Sniper achieves excellent search performance as shown in Figures 6.2 and 6.3.

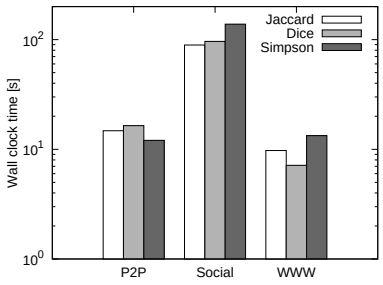


Figure 6.9: Comparison of similarity measures.

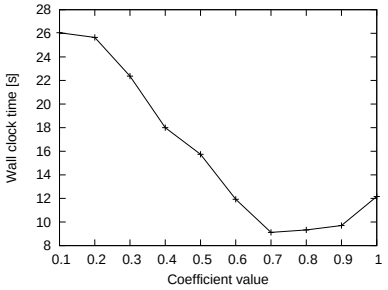


Figure 6.10: Wall clock time versus coefficient values.

Approximation accuracy

In search process, unlikely nodes are pruned by the proposed approximate graph. I show the approximation quality by distances distribution of the approximate and the original graphs. Figure 6.8 shows the result where the number of nodes is $100,000$ and dateset used is P2P.

This figure shows that the proposed approximation approach has very high accuracy; distributions are almost the same between original and approximate graphs. Even though aggregate nodes are linked if they share at least one edge between aggregated nodes to guarantee the lower bounding distances, this procedure has little impact on distances accuracy. This is because dissimilar nodes are not aggregated. Therefore the proposed approximation approach enables us to efficiently prune unlikely nodes as shown in Figure 6.7.

Similarity measures

Sniper utilizes the Jaccard coefficient as its similarity measure. However, other coefficients can be used as its similarity measure. Note that the selection can impact search efficiency. I compared the Jaccard coefficient to the Dice and Simpson coefficients with respect to the search

time. Figure 6.9 shows the results where the number of nodes was $100,000$.

The result show that Sniper can also find the lowest centrality node efficiently with the Dice and Simpson coefficients. These coefficient are commonly used for comparing the similarity of sample sets. Therefore, if two nodes have similar neighbors, these coefficients take high values. As a result, Sniper can efficiently identify the lowest centrality node with these coefficients. This result reinforces the claim that the choice of similarity measure had only virtually no impact of the performance of the proposed method.

Coefficient value

Sniper does not aggregate two nodes if the size of the intersection of adjacent node set is less than half the size of their union to avoid aggregating dissimilar nodes. This implies that two nodes are not aggregated if their Jaccard coefficient is less than 0.5. However, Sniper allows the user to select other values for node aggregation. Note that this value can affect approximate graph size and pruning power. Figure 6.10 shows the wall clock time of Sniper for several coefficient values against graphs of $100,000$ nodes where dataset was P2P.

As seen in the result, excessively small values (near 0) and excessively large values (near 1) negatively impact search efficiency. As the value increases, the size of approximate graph increases and accuracy improves (i.e., the lower bounding centrality increases). However, the approximate centrality computation time increases. Therefore, if the Jaccard coefficient is too large, it takes much time to compute the approximate centralities. However, if the value is too small, approximate nodes are not pruned effectively.

Update cost

Aggregate nodes are obtained to approximate the original graph. The most similar node is efficiently updated by computing similarities of

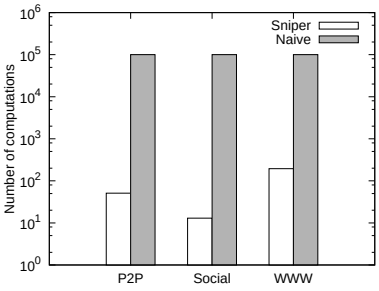


Figure 6.11: Number of computations for update.

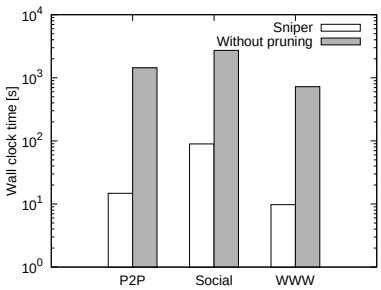


Figure 6.12: Effect of tree estimation.

only nodes close to the added node. I compared the number of similarity computations needed when the graph is updated using the proposed approach and with the naive update approach. The naive approach computes similarities between all nodes and the added node. Figure 6.11 shows the results for a set of $100,000$ nodes.

Even thought Sniper can compute the Jaccard coefficient of all nodes in the worst case, the number of similarity computations is, in practice, much lower than that of the naive approach. In the proposed update algorithm, similarities of nodes of at most two hops from the added node are computed, and the number of these nodes is very small in real graphs. Therefore, the approximate graph can be efficiently updated.

Tree estimation

Sniper terminates unnecessary distance computations early in the search process. To show the effectiveness of this idea, I removed the pruning technique from Sniper, and reexamined the wall clock time. Figure 6.12 shows the result. The number of nodes in this figure is $100,000$. Sniper without the pruning technique is abbreviated to *Without pruning* in this figure.

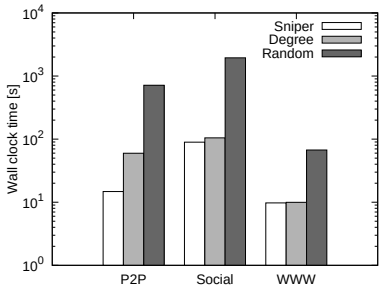


Figure 6.13: Comparison of root node selections.

The results show that the pruning technique can provide efficient search for the lowest centrality node; Sniper is up to 98 times faster if the pruning method is used. Sniper computes approximate centralities of all aggregate nodes, and the exact centralities of viable nodes to find the lowest centrality node. These computations can be effectively terminated with the technique.

Root node selection

The proposed monitoring algorithm sets the previous lowest centrality node as the root node to find the lowest centrality node efficiently. To show the effectiveness of this idea, I show the wall clock time of three root node selection methods in Figure 6.13. In this figure, *Degree* represents the results where the highest degree node is selected as the root node, and *Random* represents the results where the root node is selected at random. The number of nodes is $100,000$ in Figure 6.13.

The proposed root node selection method requires less computation time than the other two methods; it is up to 4 and 40 times faster than Degree and Random, respectively. The monitoring algorithm sets the lowest centrality node of the prior time tick as the root node, which is expected to remain the answer. As a result, it can find the lowest centrality node for time-evolving graphs efficiently.

6.6 Summary

This chapter addressed the problem of finding the lowest closeness centrality node and the lowest eccentricity centrality node from time-evolving graphs efficiently. As far as I know, this is the first study to address the lowest centrality node search problem for time-evolving graphs with the guarantee of exactness. The proposal, Sniper, is based on two ideas: (1) It approximates the original graph by aggregating original nodes to compute approximate centralities efficiently, and (2) It terminates unnecessary distance computations early in finding the answer nodes, which greatly improves the efficiency. The theoretical analyses show Sniper needs lower order of space and time complexities than the previous approximate approaches while guaranteeing exactness. Experiments show that Sniper works as expected; it can find the lowest centrality node at high speed; specifically, it is significantly (more than 110 times) faster than the existing approximate method. Centrality monitoring is fundamental for many mining applications in various domains such as social network and P2P sensor network. The proposed solution allows the lowest centrality node to be detected exactly and efficiently, and so will help to improve the effectiveness of future data mining applications.

Chapter 7

Conclusions

The theme of this thesis is to present the approaches to efficiently analyze sequence data with HMMs. The Viterbi algorithm is used in real applications. However, the Viterbi algorithm requires quadratic CPU time for the number of states. To enhance the likelihood computation speed, I proposed two approaches; approximation and pruning. Experimental results validate the effectiveness of the proposed approaches. In this chapter, I summarize the works, discuss several issues which are related to the works, and point out some potential research directions.

7.1 Summary of the thesis

HMM has been received much attentions in many research domains as an method to analyze sequence data. Sequence labeling, speech recognition, mental task classification, biological analysis, traffic monitoring, and anomaly detection are an interesting applications which are based on HMM. However the Viterbi algorithm, it has been applied to compute likelihood the modes, requires quadratic CPU time for the number of states which are used in HMMs. Therefore, I have studied the following approaches to enhance the speed likelihood computations. The proposed approach is based on two ideas; approximation and pruning. Approximation is an idea that aggregates several state to discard un-

likely states or models. And pruning is an idea that computes exact likelihood of viable states/models by pruning unlikely state transition. In this thesis, I showed that the proposed approach can significantly reduce the CPU cost for three typical HMM problems as well as analyzing time-evolving graphs as follows:

- In Chapter 3, I studied efficient likelihood computation approach for single HMM. I proposed Staggered Decoding approach as the solution. The main idea in Staggered Decoding is to gradually increase the number of original labels (states) in the iterations, and prune unnecessary label transitions by estimating upper bounding likelihood. Experimental evaluations based on standard dataset reveal that the proposed approach is much faster than the Viterbi algorithm. This results imply that the proposed approach can be an alternative to the Viterbi algorithm in many applications such as NLP tasks.

- Chapter 4 shows SPIRAL approach whose target is the problem of conducting a likelihood search on a large set of HMMs. SPIRAL is based on three ideas; (1) prune low-likelihood models in the HMM dataset by their approximate likelihoods, which yields promising candidates in an efficient manner, (2) vary the approximation granularity for each model to maintain a balance between computation time and approximation quality, and (3) discard unlikely paths in the trellis structure, which improves the efficiency. SPIRAL is also much faster than the Viterbi algorithm and expected to enhance the effectiveness of various applications such as mental task classification and biological analysis.

- In Chapter 5, I proposed the stream version of SPIRAL. That is, the problem setting in this chapter is to enhance the monitoring speed for data stream with HMMs. This problem setting is exploited in the applications such as traffic monitoring and anomaly detection. The proposed approach is carefully designed

based on the characteristics of data streams; this approach set the initial candidate model and approximation granularity by utilizing the search results in the previous time tick. Experiments show that the proposed approach works as expected, and finds high-likelihood HMMs at high speed for data streams.

- HMM is a probabilistic graphical model in which each node represents a random variable, and the edges express probabilistic relationships between these variables. In Chapter 6, to show the generality of the proposed approach for other graphical data structure, I applied the proposed approach for the problem to monitoring best centrality nodes of time-evolving graphs. I proposed Sniper as the solution which is based on two ideas; (1) approximate the original graph by aggregating original nodes to compute approximate centralities efficiently, and (2) terminate unnecessary distance computations early in finding the answer nodes, which greatly improves the efficiency. Experiments show that Sniper works as expected; it can find the lowest centrality node at high speed than the existing method.

The proposed approaches allow to compute likelihood efficiently, and I wish the proposed approaches will help to improve the effectiveness of current and future applications. I believe this is the most important contribution of this thesis.

7.2 Future work

There are several interesting research topics on the proposed approaches as follows:

- **Efficient precomputations:** The proposed approaches require precomputations for approximation to aggregate several states by using a clustering method. I utilized k-means method to aggregate states. Since state aggregation processes do not use the

query sequence, the computation cost of precomputations is independent from the query sequence. Once the aggregate states are obtained, the proposed approaches can find the best model for any data sequence. And precomputaions cost is negligible if the length of query sequence is long. Furthermore, taking into account high computation cost for learning HMM models, the computation cost of the precomputations is small. This is because model training method, such as the Baum-Welch algorithm, requires iterative computations where nm size trellis structure is explored recursively.

- **Handling long sequences:** The proposed approaches are designed to well handle models of many states since the time complexity of likelihood computation by the Viterbi algorithm is quadratic for the number of states, $O(nm^2)$. However, due to explosive data size in recent applications, more efficient approach is required. A naive approach is to aggregate several sequence symbols into single symbol. However, since this approach does not use the property of sequence data, it is not effective for likelihood computations. Exploiting the motifs (typical patterns) in a data sequence to reduce the sequence length is one promising approach. This is because the motifs exist in real sequence data. In approximate likelihood computations, using motifs instead of the sequence symbols can reduce the sequence length and can enhance the computation speed.

- **Handling the large number of models:** In the proposed approaches, the granularities of models are gradually increased to handle many models. But, if the number of models is huge, a more sophisticated approach is needed. An approach for large size of models is to construct a hierarchical data structure where several models are aggregated in a single model. By using data structure like this, answer unlikely model groups can be pruned

in high level of the hierarchical data structure. As a result, this approach can enhance the search speed.

- **More effective candidate selection:** The proposed approach sets the previous answer model as the candidate for data stream. However, by computing weighted average likelihood of each model, the candidate model can be set more effectively. This is because there is little difference between subsequences before and after the arrival of a data value. In this approach, the highest weighted average model is selected as the candidate. And weighted averages can be incrementally computed for data streams. Therefore this approach is expected to be effective to find the best model for data streams.

In Chapter 6, I confirmed the effectiveness of the proposed approaches for the centrality monitoring problem in time-evolving graphs. The proposed approach is effective since centrality in a graph and likelihood of a HMM are computed by dynamic programming for graphical data structure. Nodes in a graph are effectively aggregated same as the states in a HMM by the proposed approach. This implies that the proposed approach can be effective other graphical data structure as well as HMM when dynamic programming approach is used. I believe one convincing research direction is to apply approximation and pruning approach for other graphical data structures which are similar to HMM. I list several of them:

- Handle more complex structures than the Markov models, including semi-Markov models and factorial HMMs.
- Apply the proposed method for other graph centralities such as diameter, Random walk with restart, SimRank, or Simfusion.
- Enhance the computation speed for other graphical models such as Bayesian networks or Workflow models.

Publications

Major Publications

Journals

1. 藤原 靖宏, 櫻井 保志, 山室 雅司: ”隠れマルコフモデルデータベースの高速ゆう度検索 (Fast Likelihood Search for Hidden Markov Model Databases)”, 電子情報通信学会論文誌 D, Vol. J90-D, No. 2, pp. 325-336, 2007 年 2 月

2. Yasuhiro Fujiwara, Yasushi Sakurai, Masaru Kitsuregawa: ”Fast Likelihood Search for Hidden Markov Models”, ACM Transactions on Knowledge Discovery from Data, Vol. 11, No. 4, November, 2009

3. 藤原 靖宏, 鬼塚 真, 喜連川 優: ”時々刻々と成長するグラフのための中心性モニタリング (Centrality monitoring for time-evolving graphs)”, 情報処理学会論文誌, Vol. 52, No. 4, 2011 年 4 月

International Conference

1. Yasuhiro Fujiwara, Yasushi Sakurai, Masashi Yamamuro: ”SPIRAL: Efficient and Exact Model Identification for Hidden Markov Models”, In Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD2008), pp.247-255, Las Vegas, Nevada, USA, August 2008

2. Nobuhiro Kaji, Yasuhiro Fujiwara, Naoki Yoshinaga, Masaru Kitsuregawa: ”Efficient Staggered Decoding for Sequence Labeling”, In Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics (ACL 2010), pp.485-494, Uppsala, Sweden, July 2010

3. Yasuhiro Fujiwara, Makoto Onizuka, Masaru Kitsuregawa: ”Efficient Centrality Monitoring for Time-evolving Graphs”, In Proceedings of the The 15th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD2011), Shenzhen, China, May 2011

Others

Workshops

1. 藤原 靖宏, 櫻井 保志, 山室 雅司:”隠れマルコフデータベースの高速尤度検索 (Fast Likelihood Search for Hidden Markov Model Databases)”, 電子情報通信学会第 17 回データ工学ワークショップ (DEWS2006), 2006 年 3 月

2. 藤原 靖宏, 櫻井 保志, 山室 雅司:”SPIRAL: 隠れマルコフモデルのための高速探索手法 (SPIRAL: Efficient and Exact Model Identification for Hidden Markov Models)”, iDB 2008, 2008 年 9 月

Conventions

1. 藤原 靖宏, 櫻井 保志, 喜連川 優: ”隠れマルコフモデルによるデータストリームのモニタリング手法 (Efficient data stream monitoring for HMM)”, 情報処理学会創立 50 周年記念全国大会, 2010 年 3 月

2. 鍛治 伸裕, 藤原 靖宏, 吉永 直樹, 喜連川 優: ”自然言語処理における系列ラベリング問題のための高速で厳密な漸次的復号化アル

ゴリズム (Efficient Staggered Decoding for Sequence Labeling)”, 情報処理学会創立 50 周年記念全国大会, 2010 年 3 月

Patents

1. 藤原 靖宏, 櫻井 保志, 山室 雅司: ”隠れマルコフモデル検索方法及び装置及びプログラム及びコンピュータ読み取り可能な記録媒体”, 出願番号: 特願 2006-044428, 出願日: 2006 年 2 月 21 日, 公開番号:特開 2007-226349, 公開日: 2007 年 9 月 6 日, 登録番号: 特許 4567617, 登録日: 2010 年 8 月 13 日

2. 藤原 靖宏: ”隠れマルコフモデル探索装置及び方法及びプログラム”, 出願番号: 特願 2009-170351, 出願日: 2009 年 7 月 21 日

3. 藤原 靖宏, 櫻井 保志, 山室 雅司: ” 隠れマルコフモデル検索方法及び装置及びプログラム及びコンピュータ読み取り可能な記録媒体”, 出願番号: 特願 2006-279144, 出願日: 2006 年 10 月 12 日

Awards

1. 第 17 回データ工学ワークショップ (DEWS2006) 優秀論文賞, 2006 年 7 月, (隠れマルコフデータベースの高速尤度検索)

2. 平成 19 年度 電子情報通信学会 論文賞, 2008 年 5 月, (隠れマルコフモデルデータベースの高速ゆう度検索)

3. ACM SIGKDD 2008 Best Research Paper Awards Runner-up, August 2008, (SPIRAL: Efficient and Exact Model Identification for Hidden Markov Models)

Bibliography

[Abadi 03] Abadi, D. J., Carney, D., Çetintemel, U., Cherniack, M., Convey, C., Lee, S., Stonebraker, M., Tatbul, N., and Zdonik, S. B.: Aurora: a new model and architecture for data stream management, *VLDB J.*, Vol. 12, No. 2, pp. 120–139 (2003)

[Agrawal 93] Agrawal, R., Faloutsos, C., and Swami, A. N.: Efficient Similarity Search In Sequence Databases., in *FODO*, pp. 69–84 (1993)

[Agrawal 95] Agrawal, R., Lin, K.-I., Sawhney, H. S., and Shim, K.: Fast Similarity Search in the Presence of Noise, Scaling, and Translation in Time-Series Databases., in *VLDB*, pp. 490–501 (1995)

[Arasu 02] Arasu, A., Babcock, B., Babu, S., McAlister, J., and Widom, J.: Characterizing Memory Requirements for Queries over Continuous Data Streams, in *PODS*, pp. 221–232 (2002)

[Babcock 03] Babcock, B., Babu, S., Datar, M., and Motwani, R.: Chain : Operator Scheduling for Memory Minimization in Data Stream Systems, in *SIGMOD Conference*, pp. 253–264 (2003)

[Baldi 94] Baldi, P., Chauvin, Y., Hunkapiller, T., and McClure, M. A.: Hidden markov models of biological primary sequence information, *Proceedings of the National Academy of Science*, Vol. 91, pp. 1059–1063 (1994)

[Barbará 01] Barbará, D., Couto, J., Jajodia, S., and Wu, N.: ADAM: A Testbed for Exploring the Use of Data Mining in Intrusion Detection, *SIGMOD Record*, Vol. 30, No. 4, pp. 15–24 (2001)

[Baum 66] Baum, L. E. and Petrie, T.: Statistical inference for probabilistic functions offinite Markov chaines., *Ann. Math. Stat.*, Vol. 37, pp. 1554–1563 (1966)

[Bickel 01] Bickel, P., Chen, C., Kwon, J., Pravin, J. R., and Zwet, V. E. V.: Traffic flow on a freeway network, in *In Workshop on Nonlinear Estimation and Classification* (2001)

[Bishop 07] Bishop, C. M.: *Pattern Recognition and Machine Learning*, Springer (2007)

[Bocchieri 93] Bocchieri, E.: Vector quantization for the efficient computation of continuous density likelihoods., in *ICASSP*, pp. 692–695 (1993)

[Brants 00] Brants, T.: TnT - A Statistical Part-of-Speech Tagger, in *Proceedings of ANLP*, pp. 224–231 (2000)

[Broder 97] Broder, A. Z., Glassman, S. C., Manasse, M. S., and Zweig, G.: Syntactic Clustering of the Web, *Computer Networks*, Vol. 29, No. 8-13, pp. 1157–1166 (1997)

[Chandrasekaran 03] 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.: TelegraphCQ: Continuous Dataflow Processing for an Uncertain World, in *CIDR* (2003)

[Charniak 06] Charniak, E., Johnson, M., Elsner, M., Austerweil, J., Ellis, D., Haxton, I., Hill, C., Shrivaths, R., Moore, J., Pozar, M., and Vu, T.: Multi-level Coarse-to-Fine PCFG Parsing, in *Proceedings of NAACL*, pp. 168–175 (2006)

[Cheng 03] Cheng, R., Kalashnikov, D. V., and Prabhakar, S.: Evaluating Probabilistic Queries over Imprecise Data., in *SIGMOD Conference*, pp. 551–562 (2003)

[Cheng 04] Cheng, R., Xia, Y., Prabhakar, S., Shah, R., and Vitter, J. S.: Efficient Indexing Methods for Probabilistic Threshold Queries over Uncertain Data., in *VLDB*, pp. 876–887 (2004)

[Cohn 06] Cohn, T.: Efficient Inference in Large Conditional Random Fields, in *Proceedings of ECML*, pp. 606–613 (2006)

[Collins 02] Collins, M.: Discriminative Training Methods for Hidden Markov Models: Theory and Experiments with Perceptron Algorithms, in *Proceedings of EMNLP*, pp. 1–8 (2002)

[Cranor 03] Cranor, C. D., Johnson, T., Spatscheck, O., and Shkapenyuk, V.: Gigascope: A Stream Database for Network Applications, in *SIGMOD Conference*, pp. 647–651 (2003)

[Das 97] Das, G., Gunopulos, D., and Mannila, H.: Finding Similar Time Series., in *PKDD*, pp. 88–100 (1997)

[Dean 08] Dean, J. and Ghemawat, S.: MapReduce: simplified data processing on large clusters, *Commun. ACM*, Vol. 51, No. 1, pp. 107–113 (2008)

[Denning 98] Denning, D. E.: *Cyberspace attacks and countermeasures*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA (1998)

[Deshpande 05] Deshpande, A., Guestrin, C., Hong, W., and Madden, S.: Exploiting Correlated Attributes in Acquisitional Query Processing, in *ICDE*, pp. 143–154 (2005)

[Durbin 99] Durbin, R., Eddy, S. R., Krogh, A., and Mitchison, G.: *Biological sequence analysis: probabilistic models of proteins and nucleic acids*, Cambridge University Press (1999)

[Eickeler 98] Eickeler, S., Kosmala, A., and Rigoll, G.: Hidden markov model based continuous online gesture recognition, in *ICPR*, pp. 1206–1208 (1998)

[Elmacioglu 05] Elmacioglu, E. and Lee, D.: On six degrees of separation in DBLP-DB and more, *SIGMOD Record*, Vol. 34, No. 2, pp. 33–40 (2005)

[Esposito 07] Esposito, R. and Radicioni, D. P.: CarpeDiem: an algorithm for the fast evaluation of SSL classifiers, in *ICML*, pp. 257–264 (2007)

[Esposito 09] Esposito, R. and Radicioni, D. P.: CarpeDiem: Optimizing the Viterbi Algorithm and Applications to Supervised Sequential Learning, *Jorunal of Machine Learning Research*, Vol. 10, pp. 1851–1880 (2009)

[F. Jelinek 99] F. Jelinek, : *Statistical methods for speech recognition.*, The MIT Press (1999)

[Faloutsos 94] Faloutsos, C., Ranganathan, M., and Manolopoulos, Y.: Fast Subsequence Matching in Time-Series Databases., in *SIGMOD Conference*, pp. 419–429 (1994)

[Felzenszwalb 03] Felzenszwalb, P. F., Huttenlocher, D. P., and Kleinberg, J. M.: Fast Algorithms for Large-State-Space HMMs with Applications to Web Usage Analysis, in *Proceedings of NIPS*, pp. 409–416 (2003)

[Fujiwara 08] Fujiwara, Y., Sakurai, Y., and Yamamuro, M.: SPIRAL: efficient and exact model identification for hidden Markov models, in *KDD*, pp. 247–255 (2008)

[G. Pfurtscheller 94] G. Pfurtscheller, D. F. and Neuper, C.: Differentiation between finger, toe and tongue movement in man based on 40

Hz EEG, *Electroencephalography and Clinical Neurophysiology*, pp. 456–460 (1994)

[Gales 99] Gales, M., Knill, K., and Young, S.: State-based Gaussian selection in large vocabulary continuous speech recognition using HMMs., in *TSAP*, pp. 152–161 (1999)

[Ganti 00] Ganti, V., Gehrke, J., and Ramakrishnan, R.: DEMON: Mining and Monitoring Evolving Data, in *ICDE*, pp. 439–448 (2000)

[Gao 05] Gao, L. and Wang, X. S.: Continuous Similarity-Based Queries on Streaming Time Series, *IEEE Trans. Knowl. Data Eng.*, Vol. 17, No. 10, pp. 1320–1332 (2005)

[Garfield 72] Garfield, E.: "Citation Analysis as a Tool in Journal Evaluation", *Science*, Vol. 178, pp. 471–479 (1972)

[Gehrke 01] Gehrke, J., Korn, F., and Srivastava, D.: On Computing Correlated Aggregates Over Continual Data Streams, in *SIGMOD Conference*, pp. 13–24 (2001)

[Goldberg 05] Goldberg, A. V. and Harrelson, C.: Computing the shortest path: search meets graph theory, in *SODA*, pp. 156–165 (2005)

[Hart 68] Hart, P. E., Nilsson, N. J., and Raphael, B.: A Formal Basis for the Heuristic Determination of Minimum Cost Paths, *IEEE Transactions of Systems Science and Cybernetics*, Vol. 4, pp. 100–107 (1968)

[Haussler 93] Haussler, D., Krogh, A., Mian, I. S., and Sjolander, K.: Protein Modeling using Hidden Markov Models: Analysis of Globins, in *HICSS 39*, pp. 792–802 (1993)

[Helbing 00] Helbing, D., Herrmann, H. J., Schreckenberg, M., and Wolf, D. E.: *Traffic and Granular Flow '99: Social, Traffic, and Granular Dynamics*, Springer-Verlag (2000)

[Hu 96] Hu, J., Brown, M. K., and Turin, W.: HMM Based On-Line Handwriting Recognition., *IEEE Trans. Pattern Anal. Mach. Intell.*, Vol. 18, No. 10, pp. 1039–1045 (1996)

[Huang 05] Huang, J., Liu, Z., and Wang, Y.: Joint scene classification and segmentation based on hidden Markov model., *IEEE Transactions on Multimedia*, Vol. 7, No. 3, pp. 538–550 (2005)

[Hunt 89] Hunt, M. and Lefebvre, C.: A comparison of several acoustic representations for speech recognition with degraded and undegraded speech., in *ICASSP*, pp. 262–265 (1989)

[Jeong 09] Jeong, M., Lin, C.-Y., and Lee, G. G.: Efficient Inference of CRFs for Large-Scale Natural Language Data, in *Proceedings of ACL-IJCNLP Short Papers*, pp. 281–284 (2009)

[Kahn 99] Kahn, J. M., Katz, R. H., and Pister, K. S. J.: Next Century Challenges: Mobile Networking for "Smart Dust", in *MOBICOM*, pp. 271–278 (1999)

[Kaufman 05] Kaufman, L. and Rousseeuw, P. J.: *Finding Groups in Data: An Introduction to Cluster Analysis*, Wiley-Interscience (2005)

[Keogh 02] Keogh, E. J.: Exact Indexing of Dynamic Time Warping., in *VLDB*, pp. 406–417 (2002)

[Kossinets 06] Kossinets, G. and Watts, D. J.: Empirical Analysis of an Evolving Social Network, *Science*, Vol. 311, No. 5757, pp. 88–90 (2006)

[Kwon 00] Kwon, J. and Murphy, K.: Modeling Freeway Traffic with Coupled HMMs, *Tech. Rep., University of California at Berkeley* (2000)

[Lafferty 01] Lafferty, J., McCallum, A., and Pereira, F.: Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data, in *Proceedings of ICML*, pp. 282–289 (2001)

[Lane 99] Lane, T.: Hidden Markov Models for Human/Computer Interface Modeling, in *IJCAI-99 Workshop on Learning About Users*, pp. 35–44 (1999)

[Law 00] Law, M. H. C. and Kwok, J. T.: Rival Penalized Competitive Learning for Model-Based Sequence Clustering., in *ICPR*, pp. 2195–2198 (2000)

[Leskovec 07] Leskovec, J., Kleinberg, J. M., and Faloutsos, C.: Graph evolution: Densification and shrinking diameters, *TKDD*, Vol. 1, No. 1 (2007)

[Levinson 82] Levinson, S. E., Rabiner, L. R., and Sondhi, M. M.: An Introduction to the Application of the Theory of Probabilistic Functions of a Markov Process to Automatic Speech Recognition., *Bell Syst. Tech. J*, Vol. 62, pp. 1035–1074 (1982)

[Li 99] Li, C. and Biswas, G.: Clustering sequence data using hidden Markov model representation., in *SPIE Conference on Data Mining and Knowledge Discovery: Theory, Tools, and Technology*, pp. 14–21 (1999)

[Li 00] Li, C. and Biswas, G.: A Bayesian Approach to Temporal Data Clustering using Hidden Markov Models., in *ICML*, pp. 543–550 (2000)

[Lifshits 07] Lifshits, Y., Mozes, S., Weimann, O., and Ziv-Ukelson, M.: Speeding up HMM Decoding and Training by Exploiting Sequence Repetitions, *Computational Pattern Matching*, pp. 4–15 (2007)

[Manning 99] Manning, C. and Schütze, H.: *Foundations of Statistical Natural Language Processing*, MIT Press (1999)

[Matsuzaki 07] Matsuzaki, T., Miyao, Y., and Tsujii, J.: Efficient HPSG Parsing with Supertagging and CFG-Filtering, in *Proceedings of IJCAI*, pp. 1671–1676 (2007)

[Moon 02] Moon, Y.-S., Whang, K.-Y., and Han, W.-S.: General match: a subsequence matching method in time-series databases based on generalized windows., in *SIGMOD Conference*, pp. 382–393 (2002)

[Motwani 03] Motwani, R., Widom, J., Arasu, A., Babcock, B., Babu, S., Datar, M., Manku, G. S., Olston, C., Rosenstein, J., and Varma, R.: Query Processing, Approximation, and Resource Management in a Data Stream Management System, in *CIDR* (2003)

[Mount 01] Mount, D. W.: *Bioinformatics: sequence and genome analysis*, Cold Spring Harbor Laboratory Press (2001)

[Mount 04] Mount, D. W.: *Bioinformatics: Sequence and Genome Analysis*, Cold Spring Harbor Laboratory Press (2004)

[Nascimento 03] Nascimento, M. A., Sander, J., and Pound, J.: Analysis of SIGMOD's co-authorship graph, *SIGMOD Record*, Vol. 32, No. 3, pp. 8–10 (2003)

[Newman 03] Newman, : The Structure and Function of Complex Networks, *SIREV: SIAM Review*, Vol. 45, (2003)

[Ney 92] Ney, H., Mergel, D., Noll, A., and Paesler, A.: Data driven search organization for continuous speech recognition., *IEEE Trans. Signal Processing.*, Vol. 40, No. 2, pp. 272–281 (1992)

[Ng 02] Ng, T. S. E. and Zhang, H.: Predicting Internet Network Distance with Coordinates-Based Approaches, in *INFOCOM* (2002)

[Ninomiya 06] Ninomiya, T., Matsuzaki, T., Tsuruoka, Y., Miyao, Y., and Tsujii, J.: Extremely Lexicalized Models for Accurate and Fast HPSG Parsing, in *Proceedings of EMNLP*, pp. 155–163 (2006)

[Novak 04] Novak, D., T. Al-Ani, Y. H., and Lhotska, L.: Electroencephalogram processing using Hidden Markov Models., in *EUROSIM* (2004)

[Potamias 09] Potamias, M., Bonchi, F., Castillo, C., and Gionis, A.: Fast shortest path distance estimation in large networks, in *CIKM*, pp. 867–876 (2009)

[Qian 09] Qian, Z., Jiang, X., Zhang, Q., Huang, X., and Wu, L.: Sparse Higher Order Conditional Random Fields for Improved Sequence Labeling, in *Proceedings of ICML* (2009)

[Rabiner 85] Rabiner, L. R., Juang, B. H., Levinson, S. E., and Sondhi, M. M.: Recognition of Isolated Digits Using Hidden Markov Models With Continuous Mixture Densities., *AT&T Tech. J*, Vol. 64, pp. 1211–1234 (1985)

[Rabiner 86] Rabiner, L. R. and Juang, B. H.: An introduction to hidden Markov models., *IEEE ASSP Magazine*, Vol. 3, pp. 4–16 (1986)

[Rabiner 89] Rabiner, L. R.: A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition, in *Proceedings of The IEEE*, pp. 257–286 (1989)

[Rapoport 53] Rapoport, A.: Spread of information through a population with socio-structural bias: I. Assumption of transitivity, *Bulletin of Mathematical Biology*, Vol. 15, No. 4, pp. 523–533 (1953)

[Rattigan 06] Rattigan, M. J., Maier, M., and Jensen, D.: Using structure indices for efficient approximation of network properties, in *KDD*, pp. 357–366 (2006)

[Reka 02] Reka, A. and Barabási, : Statistical mechanics of complex networks, *Rev. Mod. Phys.*, Vol. 74, pp. 47–97 (2002)

[Roth 05] Roth, D. and Yih, W.: Integer Linear Programming Inference for Conditional Random Fields, in *Proceedings of ICML*, pp. 737–744 (2005)

[Sagayama 95] Sagayama, S., Knill, K., and Takahashi, S.: On the use of scalar quantization for fast HMM computation., in *ICASSP*, pp. 213–216 (1995)

[Sha 03] Sha, F. and Pereira, F.: Shallow Parsing with Conditional Random Fields, in *Proceedings of HLT-NAACL*, pp. 134–141 (2003)

[Siddiqi 05a] Siddiqi, S. M. and Moore, A. W.: Fast inference and learning in large-state-space HMMs, in *ICML*, pp. 800–807 (2005)

[Siddiqi 05b] Siddiqi, S. M. and Moore, A. W.: Fast Inference and Learning in Large-State-Space HMMs, in *Proceedings of ICML*, pp. 800–807 (2005)

[Singh 94] Singh, S. P., Jaakkola, T., and Jordan, M. I.: Reinforcement Learning with Soft State Aggregation., in *NIPS*, pp. 361–368 (1994)

[Smyth 96] Smyth, P.: Clustering Sequences with Hidden Markov Models., in *NIPS*, pp. 648–654 (1996)

[Sugawara 08] Sugawara, H., Yamada, H., and Yamada, H.: Introduction of super computer system for DNA data bank of Japan, *FUJITSU Sci. Tech. J.*, Vol. 44, No. 4, pp. 442–448 (2008)

[Sun 06] Sun, J., Tao, D., and Faloutsos, C.: Beyond streams and graphs: dynamic tensor analysis, in *KDD*, pp. 374–383 (2006)

[Sun 07] Sun, J., Faloutsos, C., Papadimitriou, S., and Yu, P. S.: GraphScope: parameter-free mining of large time-evolving graphs, in *KDD*, pp. 687–696 (2007)

[Tao 05] Tao, Y., Cheng, R., Xiao, X., Ngai, W. K., Kao, B., and Prabhakar, S.: Indexing Multi-Dimensional Uncertain Data with Arbitrary Probability Density Functions., in *VLDB*, pp. 922–933 (2005)

[Tatbul 03] Tatbul, N., Çetintemel, U., Zdonik, S. B., Cherniack, M., and Stonebraker, M.: Load Shedding in a Data Stream Manager, in *VLDB*, pp. 309–320 (2003)

[Toutanova 03] Toutanova, K., Klein, D., Manning, C., and Singer, Y.: Feature-rich Part-of-Speech Tagging with a Cyclic Dependency Network, in *Proceedings of HLT-NAACL*, pp. 252–259 (2003)

[Tsuruoka 05] Tsuruoka, Y. and Tsujii, J.: Bidirectional Inference with the Easiest-First Strategy for Tagging Sequence Data, in *Proceedings of HLT/EMNLP*, pp. 467–474 (2005)

[Viterbi 67] Viterbi, A. J.: Error Bounds for Convolutional Codes and an Asymeptotically Optimum Decoding Algorithm, *IEEE Transactios on Information Theory*, Vol. 13, No. 2, pp. 260–267 (1967)

[Wang 06] Wang, D.: A Graph-Center-Based Scheme for Energy-Efficient Data Collection in Wireless Sensor Networks, in *MSN*, pp. 579–587 (2006)

[Warrender 99] Warrender, C., Forrest, S., and Pearlmutter, B. A.: Detecting Intrusions using System Calls: Alternative Data Models, in *IEEE Symposium on Security and Privacy*, pp. 133–145 (1999)

[Yi 98] Yi, B.-K., Jagadish, H. V., and Faloutsos, C.: Efficient Retrieval of Similar Time Sequences Under Time Warping., in *ICDE*, pp. 201–208 (1998)

[Zhang 96] Zhang, T., Ramakrishnan, R., and Livny, M.: BIRCH: An Efficient Data Clustering Method for Very Large Databases., in *SIGMOD Conference*, pp. 103–114 (1996)

[Zhong 02] Zhong, S. and Ghosh, J.: HMMs and coupled HMMs for multi-channel EEG classification, in *IEEE Int. Joint Conf. on Neural Networks*, pp. 1154–1159 (2002)

[Zhu 02] Zhu, Y. and Shasha, D.: StatStream: Statistical Monitoring of Thousands of Data Streams in Real Time., in *VLDB*, pp. 358–369 (2002)