Finding Semantics in Time Series

Peng Wang [1,2]* Haixun Wang [2] Wei Wang [1]

[1]Fudan University, Shanghai, China
{pengwang5, weiwang1}@fudan.edu.cn
[2]Microsoft Research Asia, Beijing, China
haixunw@microsoft.com

ABSTRACT

In order to understand a complex system, we analyze its output or its log data. For example, we track a system's resource consumption (CPU, memory, message queues of different types, etc) to help avert system failures; we examine economic indicators to assess the severity of a recession; we monitor a patient's heart rate or EEG for disease diagnosis. Time series data is involved in many such applications. Much work has been devoted to pattern discovery from time series data, but not much has attempted to use the time series data to unveil a system's internal dynamics. In this paper, we go beyond learning patterns from time series data. We focus on obtaining a better understanding of its data generating mechanism, and we regard patterns and their temporal relations as organic components of the hidden mechanism. Specifically, we propose to model time series data using a novel pattern-based hidden Markov model (pHMM), which aims at revealing a global picture of the system that generates the time series data. We propose an iterative approach to refine pHMMs learned from the data. In each iteration, we use the current pHMM to guide time series segmentation and clustering, which enables us to learn a more accurate pHMM. Furthermore, we propose three pruning strategies to speed up the refinement process. Empirical results on real datasets demonstrate the feasibility and effectiveness of the proposed approach.

Categories and Subject Descriptors

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

General Terms

Algorithms, Performance

Keywords

Time Series, Hidden Markov Model, Pattern

*This work was done at Microsoft Research Asia

1. INTRODUCTION

Time series data is being generated at an unprecedented speed and volume in a wide range of applications in almost every domain. For example, daily fluctuations of the stock market, traces produced by a computer cluster, medical and biological experimental observations, readings obtained from sensor networks, position updates of moving objects in location-based services, etc, are all represented in time series. Consequently, there is an enormous interest in analyzing (including query processing and mining) time series data, which has resulted in a large number of works on new methodologies for indexing, classifying, clustering, and summarizing time series data [8, 28, 13].

In this paper, we propose to study time series from a new angle. Our goal is to understand the complex system that produces the time series. Thus, instead of finding isolated historic patterns, or predicting the next time series value based on the pattern in the most recent time window, we focus on explaining the relationships between the patterns, in particular, how they fit into a big, holistic picture that describes the underlying system.

1.1 State of the art

Much work has been done on time series analysis, including time series prediction [1, 6, 13, 9, 21], time series segmentation and symbolization [12, 14], time series representation [7, 25], and similar time series matching [8, 18]. However, not much attempt has been made to use the time series data to explain how the underlying system works.

Well known time series models such as ARIMA and Linear Regression models, have been used for time series forecasting, which is concerned with the problem of predicting time series values X_t given observations $X_1, X_2, \cdots, X_{t-1}$. One frequently used assumption in time series forecasting is that the time series has a short memory, which means current values are only related to values in a recent time window. In other words, these approaches focus on local characteristics in the time series, and do not attempt to explain observations using the internal dynamics of the system.

Approaches such as Discrete Fourier Transform [8], Discrete Wavelet Transform [2], Piecewise Linear Representation [11], and Symbolic Aggregate Approximation [14], try to model the whole time series. Their goal lies in representing the original time series in a more concise way so that we can summarize the time series or index the time series for fast pattern matching or pattern discovery. These methods cannot reveal the internal dynamics of the system, though. In DFT [8], for example, a time series is described by a set of coefficients in the frequency domain. However, the coefficients are not interpretable, that is, knowing the coefficients in the frequency domain does not necessarily enable us to understand how the system works. Similarly, in PLR [11], a time series is segmented into disjoint intervals, each of which is represented by a line segment. However, the line segments are isolated. For instance, we do not know whether the similarity between the line segments at time t_1 and t_2 means the system is in the same internal state at t_1 and t_2.

Much work has been done on discovering frequent patterns (also called motifs) in time series [17, 16]. However, frequent patterns may not necessarily be important patterns, in terms of whether they can inform us how the system works. Many mining algorithms discover a large number of patterns that are hard to interpret, which adds to the complexity of understanding the system instead of helping reduce it.

1.2 Revealing system dynamics

Our goal is to obtain a better understanding of the system that generates the time series. We assume the system that generates the time series operates under a number of latent states [23, 24, 9, 21, 5, 4]. Many systems fall into this category. Our approach is based on two important observations we made about such systems. The first observation is that once a system is in a latent state, it will stay in the state for a period of time until a certain event occurs which leads the system to another latent state. For example, when memory usage is below the physical memory capacity, the system behaves in a certain way. The system exhibits stable patterns until memory usage exceeds the physical memory capacity, when the system starts paging. The second observation is that the system goes through the same states over and over again. For example, once memory usage recedes, the system will return to its old state (without paging).

To have a better understanding of the system means to reveal the latent states of the system, and how they alternate among themselves. In this paper, we regard the time series as the output generated by a state transition machine. Time series produced in each state demonstrates a certain pattern of fluctuation, and transitions between states reflect system dynamics. That is, the information of once the system ending a certain state, what's the most likely state the system will enter.

The next question is what constitutes a state, or what constitutes a unit of observation? This is the most challenging question when using a state transition machine (e.g., an HMM) on time series data, since it is not trivial to align an observation sequence with a state sequence. A naïve and straightforward choice is to consider a single observed value as an output token of a state. However, in time series data, a single value contains very little semantics. We illustrate this by an example in Figure 1. Although A and B have the same value, the underlying system is likely to be in two different states when it outputs A and B, because A is in an upward trend and B is in a downward trend. If our goal is to predict the next time series value, then knowing the system in that value state has no predictive power. As a consequence, with single values as states, the obtained machine will be full of uncertainty, which is not we want.

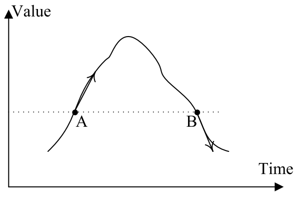


Figure 1: Single value as state

A better choice is to group neighboring values into a pattern, and regard a pattern as a unit of observation (an observation token). A good choice for a pattern is a line segment or a polynomial curve. If line segments are used, the time series in Figure 1 will be represented by two line segments with different shapes, indicating that the underlying system is in two different states. There are several benefits of using line segments: 1) lines have simple shapes and the trends they represent are easy to understand; 2) in many applications, a time series can be represented well by a sequence of line segments.

Thus, our task is to: i) define a set of observation tokens, each being a representative line segment in the time series; ii) convert the time series to an observation sequence such that each observation token aligns with a state in the unknown state sequence; iii) learn an HMM from the observation sequence. However, the task of obtaining the observation sequence, or more specifically, the task of segmenting a time series and then clustering the segments to obtain representative line segments, is not trivial. The objective of many time series segmentation and clustering approaches is to minimize the difference between the time series and the resulting line segment sequence. However, this objective is not necessarily aligned with that of finding the best state transition machine.

To see this, consider a toy example in Figure 2. Suppose we have 4 line segments $A_1, \cdots, A_4$, and the question is whether we should consider A_1 and A_2 as the same observation (i.e., representing A_1 and A_2 using the same observation token), or consider A_3 and A_4 as the same observation. It is clear that A_1 and A_2 are more similar to each other in shape. Thus, traditional clustering approaches, which aim at minimizing approximation error, will group A_1 and A_2 together.

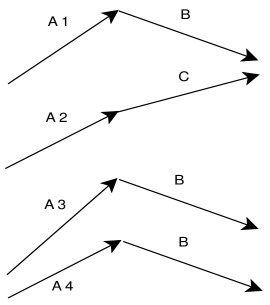


Figure 2: Segmenting

However, clustering line segments as if they are a set of disconnected elements is problematic. For instance, the line segments following A_3 and A_4 have exactly the same shape, which suggests that A_3 and A_4 may be generated by the system in the same state. On the other hand, the line segments following A_1 and A_2 are totally different, which suggests that the slight difference between A_1 and A_2 may indicate that they actually belong to two different states. To understand the internal dynamics of the system requires us to pay attention to these temporal constraints instead of just minimizing the approximation error.

1.3 A Two Phase, Iterative Approach

To achieve the goal of revealing internal system dynamics, we propose a pattern-based Hidden Markov model (pHMM) for time series data. We discover patterns in the time series, and we ensure that the discovered patterns are not disconnected or isolated, but rather, they are organic components of a state transition machine, which produces the original time series. Then, the challenges are the following: i) in order to segment the time series and discover patterns we should know the state transition machine first, because it tells us the likelihood of one pattern being followed by another pattern, otherwise we run into problems demonstrated by the example in Figure 2; and ii) to build the state transition machine we must know the patterns first, as patterns are the sole components of the state transition machine.

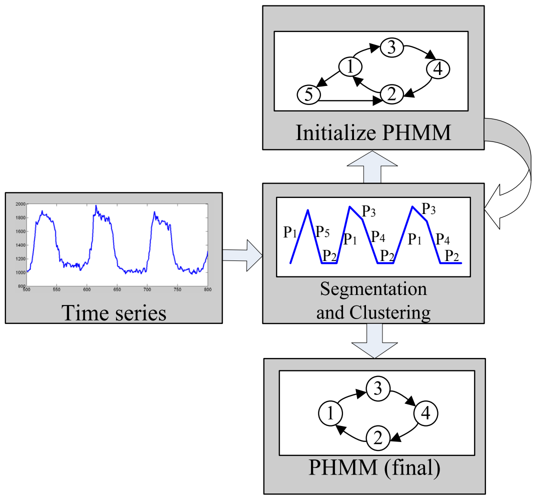


Figure 3: Overview of our approach

To solve this dilemma, we propose a two phase approach. In phase one, we first segment the time series using traditional optimization techniques. Because we do not have any knowledge about the underlying state transition machine, the best thing we can do is to use a standard approach [11] to convert the time series into a piecewise linear representation. In other words, we approximate the time series using line segments that minimize the approximation error. Then, we cluster the resulting segments using a greedy clustering method (considering both the similarity and the temporal constraints). Finally, from the segmented time series, we learn a hidden Markov model.

In phase two, we use an iterative process to refine the model. Specifically, in each round, we first segment and cluster the time series based on the learned pHMM. The pHMM provides important guidance for segmenting and clustering, resulting in higher quality patterns. Then we update the pHMM based on learned patterns. We prove that the iteration process always improves the quality of the model. The whole framework is illustrated in Figure 3.

1.4 Applications and Contributions

Our goal is to reveal the system beneath the time series data it produces. With the knowledge of the underlying system, we will be able to perform a large variety of challenging tasks. A representative list of tasks include the following:

- Trend prediction. With the knowledge of the state transition machine, we can derive the temporal relations between patterns. This enables us to answer queries such as: What the trend of the time series is in 10 minutes; or when will the time series end the current downward trend and enter an upward trend?

- Accurate multi-step value prediction. Predicting time series values long into the future is a challenging and important task. Specifically, given time series before time point t, we want to predict the values at time $t + \delta$, where δ is much bigger than 1.

- Pattern based correlation detection. In traditional approaches, in order to compute correlation between two time series, we map the time series into a vector space (e.g. using DFT or DWT), and use a distance measure (e.g. Euclidean distance or Dynamic Time Warping [7]) to calculate their similarity. Now we can compute correlation based on patterns. Furthermore, we can correlate the time series by rules such as: whenever pattern P_1 occurs in time series S_1, P_2 will occur in time series S_2.

In summary, the contributions we make in this paper are the following:

- We introduce a pattern-based hidden Markov model (pHMM) for time series data. It focuses on revealing the internal dynamics of the system that produces the time series.

- We propose an iterative approach to refine the model. Furthermore, we propose several pruning strategies to speed up the refinement process.

- We propose algorithms that use pHMM to perform multi-step value prediction, trend prediction and pattern based correlation detection.

- We conduct extensive experiments to verify the effectiveness and efficiency of the proposed approach.

1.5 Paper Organization

The rest of the paper is organized as follows. Section 2 discusses the problem and the challenges. Section 3 introduces the algorithm in the initial phase. Section 4 describes the method to refine pHMM. Section 5 shows how to utilize the learned model. Section 6 shows experimental results. In Section 7, we discuss related work, and we conclude in Section 8.

2. PRELIMINARY AND APPROACH

In this paper, we propose using the pattern-based hidden Markov model (pHMM) to reveal system dynamics from time series data.

2.1 Background of HMM

A hidden Markov model (HMM) is a statistical model in which the system being modeled is assumed to be a Markov process. It includes a finite set of states, each of which is associated with a probability distribution over all possible output tokens. Transitions among the states are governed by a set of probabilities. The states are not visible, but outputs produced by the states are. Given a sequence of observations, we learn an HMM and derive a sequence of hidden states that correspond to the sequence of observations.

Formally, an HMM, denoted by $\lambda = \{S, A, B, \pi\}$, is described with the following parameters:

- A set of states $S = \{1, 2, \cdots, K\}$.

- State transition probabilities $A = \{a_{ij}\}$, $1 \leq i, j \leq K$, i.e., a_{ij} is the probability of state i transiting to state j.

- Output probabilities $B = \{b_i(o)\}$, $1 \leq i \leq K$. o is an observation with a continuous or discrete value (or value vector). $b_i(o)$ is the probability of state i generating observation o.

- Initial probabilities $\pi = \{\pi_i\}$, $1 \leq i \leq K$. π_i is the probability of the time series beginning with state i.

When we learn an HMM, the basic assumption is that the observation sequence and the hidden state sequence is aligned. In our case, however, one big challenge is to derive the alignment.

One fundamental problem associated with HMMs is the *decoding problem*: given a model, λ, and a sequence of observations,

O, find the optimal state sequence that produces the observations. Another problem is the *learning problem*: how to estimate the parameter λ of the HMM, so that the probability of the observation sequence generated by the optimal state sequence is maximized. In both cases, an important measurement is the *production probability*. Given an HMM λ, and a sequence of observations O, the production probability is the probability of HMM λ generating O along a state sequence $\mathbf{s} = (s_1, \cdots, s_m)$, and it is computed as:

$$P(O, \mathbf{s}|\lambda) = \pi_{s_1} b_{s_1}(o_1) \prod_{j=2}^{m} a_{s_{j-1},s_j} b_{s_j}(o_j) \qquad (1)$$

The larger the production probability, the better λ and the state sequence $\mathbf{s}$ describe O.

2.2 Problem Statement

Given a time series $X = x_1, x_2, \cdots, x_n$, we aim at learning a pattern-based Hidden Markov Model (pHMM), which reveals the dynamics of the system that generates the time series.

Formally, we want to solve the following problem:

- Convert the time series X into a sequence of line segments, $\mathbf{L} = (L_1, L_2, \cdots, L_m)$;
- Learn a hidden Markov model from observation sequence $\mathbf{L}$.

so that production probability

$$P(\mathbf{L}, \mathbf{s}^*|\lambda) = \pi_{s_1} b_{s_1}(L_1) \prod_{j=2}^{m} a_{s_{j-1},s_j} b_{s_j}(L_j) \qquad (2)$$

is maximized, where $\mathbf{s}^* = (s_1, s_2, \cdots, s_m)$ is the optimal state sequence, where s_j generates line L_j with probability $b_{s_j}(L_j)$.

2.3 Challenges and Overview of Our Approach

Although much research has been done on HMMs, and HMMs have been successfully applied in many applications, it is a non-trivial challenge to learn a pattern based HMM for the time series data.

As we know, a traditional HMM is learned from an observation sequence. We can estimate the parameters of an HMM using the classic Baum-Welch algorithm. However, the premise is that we have the observation sequence. In our case, the observation sequence is essentially unknown, since it needs to be learned from the time series itself. Moreover, in the time series, the number of possible patterns is infinite. For example, in our case, we represent a pattern by a line segment, whose slope and duration are all continuous values.

Furthermore, the process of learning an observation sequence from the time series cannot be decoupled from the process of learning a hidden Markov model from the observation sequence. Intuitively, producing an observation sequence from a time series is done by time series segmentation. Existing approaches segment time series by solving an optimization problem where the objective is to minimize the difference between the time series and the line segment sequence. However, these approaches consider segments as independent and isolated, and ignore the temporal relations between them, but such temporal relations are critical in learning HMM. In our work, we learn the observation sequence and the HMM simultaneously.

We solve the above problem using a two-phase approach. In the first phase, we initialize the pHMM using a cluster-based approach. The observation sequence used to learn the initial pHMM is learned from the time series without any knowledge of the pHMM. In the second phase, we refine the model with an iterative process. In each round, we first segment the time series and cluster the line segments under the guidance of a previously learned pHMM, then the pHMM is updated based on the new segmentation.

3. THE INITIAL PHMM

We introduce our approach to build the initial pHMM in 3 steps. First, we segment the time series. Second, we cluster line segments. Third, based on the clusters, we learn the first pHMM.

3.1 From Time Series to Line Segments

In phase one, information about latent states is not available, so we perform a traditional segmentation. We use a bottom-up approach to convert the time series X into a piecewise linear representation [12]. Initially, we approximate X with $\lfloor \frac{n}{2} \rfloor$ line segments. The i-th line, L_i, connects x_{2i-1} and x_{2i}. Next, we iteratively merge the neighboring lines. In each iteration, we merge the two neighboring segments into one new line segment that has the minimal approximation error. The merging process repeats until every possible merge leads to a line whose error exceeds a user specified threshold, denoted by ε_r. Clearly, without knowledge of the latent states, it is very likely that the initial segmentation is not optimal for our goal (see Section 4.1).

3.2 From Line Segments to Clusters

After obtaining the line segments, we group them into clusters $\{C_1, C_2, \cdots, C_K\}$. A key issue is to define the similarity between line segments. Had our goal been to summarize or compress the time series, we could have used the approximation error or minimal description length as the objective function. However, as our goal is to learn a pattern-based HMM, such an approach is not always optimal.

We consider two clustering criteria:

- The similarity criterion. This is the same criterion for traditional clustering. In our case, the line segments in the same cluster should have similar shapes (slopes and lengths), and the line segments in different clusters have different shapes.
- The temporal criterion. If L_i and L_j belong to the same cluster, then L_{i+1} and L_{j+1}, which follow L_i and L_j respectively, should have the same distribution (in terms of which cluster they belong to); more often than not they belong to the same cluster.

We now formalize these two criteria. For the similarity criterion, we measure the variance of the line segments in each cluster. For cluster C_i, the relative error is computed as:

$$R(i) = \frac{1}{|C_i|} \sum_{(l_j,\theta_j) \in C_i} \{ (\frac{l_j - \bar{l}_i}{\bar{l}_i})^2 + (\frac{\theta_j - \bar{\theta}_i}{\bar{\theta}_i})^2 \}$$

where $|C_i|$ is the number of lines in cluster C_i, $\bar{l}_i$ and $\bar{\theta}_i$ are the average length and the average slope of lines in C_i. Clearly, the smaller $R(i)$ is, the more similar the line segments in C_i are.

For the second criterion, we use entropy to measure the uncertainty of the clusters following the lines of cluster C_i:

$$I(i) = \sum_{j=1}^{K} -p(j|i) \log p(j|i) \qquad (3)$$

where $p(j|i)$ denotes the probability that a line in C_i is followed by a line in C_j. Intuitively, the smaller the $I(i)$, the more certain we are about the clusters that follow lines in C_i.

A straightforward way of clustering is to construct an objective function based on these two criteria as:

$$F = \alpha \cdot R + (1 - \alpha) \cdot I \qquad (4)$$

where $R = \sum_{i=1}^{K} |C_i| R(i)$ is the overall relative error of all clusters, $I = \sum_{i=1}^{K} p(i) I(i)$ is the overall entropy of all clusters, and $\alpha \in [0,1]$ is a user-provided parameter. Then we cluster the segment lines to minimize the objective function F. However, it is hard to set a reasonable α. We illustrate it with an example. Assume we want to cluster three lines, L_1, L_2 and L_3. They have the same length, and their slopes satisfy: $\theta_1 < \theta_2 < \theta_3$. Moreover, L_1 and L_3 are followed by lines in the same cluster while L_2 is followed by a line in another cluster, and the lines in these two clusters have very different shapes. If we set a very large α, the similarity criterion is dominant. It is likely that all three lines are clustered into one cluster. On the other hand, if we set a small α, the temporal criterion will dominate. A possible result is that L_1 and L_3 are put into the same cluster C while L_2 is not. However, this is unreasonable, as L_2 is "enveloped" by L_1 and L_3.

In this paper, instead of optimizing Eq. 4 directly, we adopt a greedy approach. Initially, each line is considered as a cluster on its own. Then at each iteration, we merge two clusters by considering two criteria in turn. The approach includes three steps:

Step 1. (Similarity Criterion) For each cluster C_i, find its most similar cluster, called C_i's candidate cluster.

DEFINITION 1 (CANDIDATE CLUSTER). *For each cluster C_i, its candidate cluster, denoted as T_i, is the cluster that satisfies:*

1. *$R(T_i \cup C_i) \leq R(C \cup C_i)$ holds for any $C \neq T_i$, where $T_i \cup C_i$ is the new cluster generated by merging T_i and C_i, and $R(T_i \cup C_i)$ is its relative error.*
2. *$R(T_i \cup C_i) \leq \varepsilon_c$, where ε_c is a user-specified threshold, called the relative error threshold.*

Step 2. (Temporal Criterion) For any cluster pair (C_i, T_i), compute the entropy of new cluster $C_i \cup T_i$.

Step 3. Merge a pair with minimal entropy to a new cluster.

This process continues until every possible merge results in a relative error that exceeds the threshold ε_c.

Connection between the two measurements. In clustering, we use the similarity and the temporal criteria to measure the quality of clusters. In the problem statement, we use the production probability to measure the quality of the learned model. In fact, we can build a connection between these two measurements.

We decompose production probability into P' and P''.

$$\begin{aligned} P(\mathbf{L}, \mathbf{s}^*|\lambda) &= \pi_{s_1} b_{s_1}(L_1) \prod_{j=2}^{m} a_{s_{j-1},s_j} b_{s_j}(L_j) \\ &= \prod_{j=1}^{m} b_{s_j}(L_j) \cdot \pi_{s_1} \prod_{j=2}^{m} a_{s_{j-1},s_j} \\ &= P' \cdot P'' \end{aligned} \quad (5)$$

P' measures how well the states match the occurrences of the line segments, and it corresponds to the similarity criterion. P'' measures the certainty of transitions between states, and it corresponds to the temporal criterion. So they are consistent with each other.

3.3 From Clusters to HMM

Based on the obtained clusters, we initialize the hidden Markov model λ as follows. Assume the clusters are $\{C_1, C_2, \cdots, C_K\}$, we initialize an HMM with K states, $\{1, 2, \cdots, K\}$, in which state i corresponds to cluster C_i. In other words, we assume lines in each cluster represent the typical fluctuation of time series when the system stays in the same state.

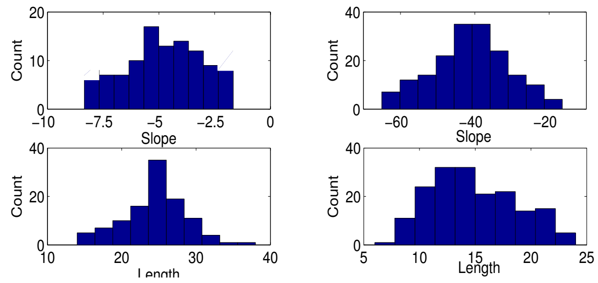
Before discussing how to initialize the output probabilities, we first define the output probability in terms of segment lines as observations. We assume slopes and lengths are independent to each other. For line $L = (l, \theta)$, the output probability is defined as a product of two probabilities:

$$b_i(L) = p_l(l|i) p_s(\theta|i)$$

in which $p_l(l|i)$ is the probability of state i generating the line with length l, and $p_s(\theta|i)$ is the probability of i generating the line with slope θ.

In many real life applications, the system operates in different states; and in each state, the system exhibits stable behavior. Each observation of one state can be regarded as the stable behavior plus some slight fluctuations, or errors. Since the observational error in an experiment is often described by Gaussian Distribution, we use it here to describe the distribution of segment lines. Formally, we assume $p_l(\cdot|i)$ and $p_s(\cdot|i)$ follow 1-dimension Gaussian Distributions, $\mathcal{N}(\mu_{il}, \sigma_{il}^2)$, and $\mathcal{N}(\mu_{is}, \sigma_{is}^2)$ respectively.

Some experiments are conducted to verify the assumption and results are shown in Figure 4. For both the Spot and Power datasets, we select one big cluster randomly, since the big cluster contains more lines that can demonstrate the distribution more clearly. It can be seen that both length and slope can be approximated by Gaussian distribution.



(a) Spot Dataset (b) Power Dataset

Figure 4: Slope and length distribution

To initialize output probabilities of state i, we need to initialize 4 parameters: $\mu_{il}, \sigma_{il}^2, \mu_{is}, \sigma_{is}^2$. We estimate μ_{il} and σ_{il}^2 as the mean and variance of lines' lengths in C_i, and estimate μ_{is} and σ_{is}^2 as the mean and variance of lines' slopes in C_i.

We build a state sequence to initialize transition probabilities and initial probabilities. In line sequence $(L_1, L_2, \cdots, L_m)$, we replace each line with the cluster it belongs to. Since cluster C_i corresponds to state i ($1 \leq i \leq K$), a state sequence is obtained. Then based on this, we estimate transition probabilities and initial probabilities as in the traditional HMM.

4. ITERATIVE PHMM REFINEMENT

We iteratively refine the pHMM learned in the previous round. Each iteration has two steps. In step one, based on the current pHMM, we use an extended Viterbi algorithm to segment the time series and learn the optimal state sequence. In step two, based on the new line segment sequence and the state sequence, we update the pHMM. The iteration stops until the pHMM does not change.

4.1 Motivation of Iterative Refinement

We illustrate the benefit of refining with an example in Figure 5. Assume in the initial phase, the time series is segmented into S_1, as shown in Figure 5(a). Consider intervals $[a, b]$, $[c, d]$ and $[e, f]$. It can be seen that subsequences in all of them have similar shapes.

Moreover, the lines before and after them are all similar. However, in S_1, $[a, b]$ is represented by line L_2, while $[c, d]$ is represented by two lines L_6 and L_7. Since the shape of L_2 is apparently different to L_6 or L_7 respectively, they cannot be clustered into a group. So, S_1 misses the information that L_2 is similar to the connection of L_6 and L_7. Moreover, according to the cluster approach, three clusters, as well as three states, will be generated: $C_1 = (L_6, L_{10})$, $C_2 = (L_7, L_{11})$ and $C_3 = (L_2)$. While the first two are meaningful states, C_3 is very likely to be a noise state. Note that such issues cannot be solved by frequent pattern mining, as such patterns may not necessarily be frequent in the entire time series.

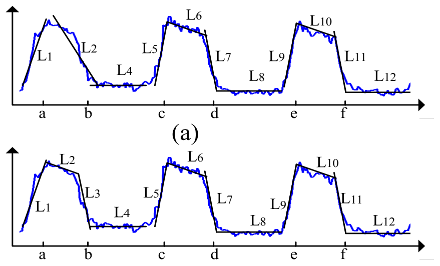


Figure 5: Segmentation with/without Refinement

In the refinement process, the time series will be re-segmented with the guidance of the current pHMM. Intuitively, the new segmentation will identify the lines that can be generated by certain states with a high probability. So $[a, b]$ will be split into L_2 and L_3, as shown in Figure 5(b). Obviously, it achieves a better pHMM.

4.2 pHMM-based Segmentation

Given an observation sequence, the Viterbi algorithm can find the optimal state sequence. However, it only works if the observation sequence is known. In our case, we only have the raw time series, instead of the observation sequence (a sequence of line segments). In this work, we extend the Viterbi algorithm to learn the line segment and the state sequence simultaneously.

4.2.1 The Traditional Viterbi Algorithm

The Viterbi algorithm is an efficient method of learning the optimal state sequence $\mathbf{s}^*$, given the HMM λ and the observation sequence O. It has a recursive procedure, and it works in parallel for all states in a strictly time synchronous manner.

The key component in the Viterbi algorithm is the *optimal probability*, denoted as $\delta_t(i)$, which is the maximal probability of HMM generating the observation segment $o_1, \cdots, o_t$, along the optimal state sequence $s_1, \cdots, s_t$, in which $s_t = i$. That is,

$$\begin{aligned}\delta_t(i) &= \max_{s_1,\cdots,s_{t-1}} P(o_1, \cdots, o_t, s_1, \cdots, s_{t-1}, s_t = i|\lambda) \\ &= \max_{s_1,\cdots,s_{t-1}} \pi_{s_1} b_{s_1}(o_1) \prod_{j=2}^{t} (a_{s_{j-1},s_j} b_{s_j}(o_j))\end{aligned}$$

The algorithm scans the observation sequence from $t = 1$, at which point the optimal probability for state i is initiated as

$$\delta_1(i) = \pi_i b_i(o_1)$$

Assume all $\delta_{t-1}(j), 1 \le j \le K$, are already obtained. The algorithm computes $\delta_t(i)$ as:

$$\delta_t(i) = \max_j (\delta_{t-1}(j) a_{ji}) b_i(o_t)$$

When the algorithm reaches the last time point n, we obtain all optimal probabilities: $\delta_n(i), 1 \le i \le K$. By comparing all of them, and backtracking the largest one, this algorithm obtains the optimal state sequence.

4.2.2 Extending the Viterbi Algorithm

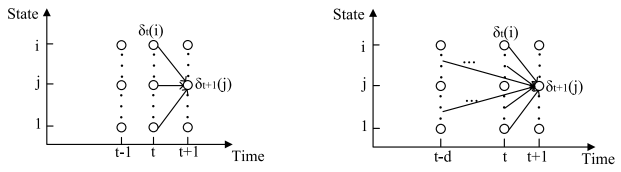
As analyzed before, we need to learn the line sequence and the state sequence simultaneously. We implement it with a modified optimal probability, $\delta_t(i)$. In other words, in the traditional Viterbi algorithm, we only learn the optimal state sequence when computing $\delta_t(i)$, while in our algorithm, we not only learn the optimal state sequence, but also find the optimal line sequence.

In our case, an observation token, or the unit of observation, is a line segment. Therefore, we define $\delta_t(i)$ as the maximal probability of the current HMM generating any line sequence up to t along the optimal state sequence ending with state i. Formally,

$$\delta_t(i) = \max_{L_1,\cdots,L_k} \max_{s_1,\cdots,s_{k-1}} P(L_1, \cdots, L_k, s_1, \cdots, s_k = i|\lambda)$$

where $\{L_1, \cdots, L_k\}$ is a line sequence, in which the last line segment L_k ends at time point t, and its corresponding state is i. state s_k corresponds to the k-th line. Note that k can be any value not exceeding $\lfloor \frac{t}{2} \rfloor$.

Intuitively, in the Viterbi algorithm, when $\delta_t(i)$ is computed from $\delta_{t-1}(j)$, it implies that the observation at time t is added to the observation sequence. Similarly, in our algorithm, when $\delta_t(i)$ is computed, it implies a "new observation" is added to the observation sequence. However, here the observation is a line segment ending at t, instead of a single value.



(a) Traditional (b) Extended

Figure 6: Computing optimal probability

To be specific, we compute a possible $\delta'_t(i)$ based on any previous optimal probability $\delta_{t-d}(j)$ as

$$\delta'_t(i) = \delta_{t-d}(j) a_{ji} b_i(L)$$

where L is the new observed line, which begins at $t-d+1$ and ends at t, and its corresponding state is i. The new line L is determined by t and d. The only limitation of d is that the approximation error of L on the interval $[t - d + 1, t]$ cannot exceed ε_r.

Since we do not know the optimal line sequence, we cannot determine the value of d beforehand. We compute $\delta_t(i)$ by checking all possible previous optimal probabilities, and choose the largest result as the final $\delta_t(i)$:

$$\delta_t(i) = \max_{d,j} (\delta_{t-d}(j) a_{ji}) b_i(L)$$

where line L is a variable for different previous optimal probabilities. Figure 6 illustrates the difference between our approach with the traditional Viterbi algorithm.

When the algorithm reaches time n, we obtain the maximal optimal probability $\max_i \delta_n(i)$. Through backtracking it, we obtain the optimal observation sequence and the corresponding state sequence. The detailed algorithm is shown in Algorithm 1.

In Algorithm 1, function $BestLine(x, y)$ (line 7) learns the best-fit line beginning from x and ending at y, which has the minimal

Algorithm 1 Detect_state_sequence

```
1: Input $\varepsilon_r$:maximal error threshold of line approximation
2: Initialize $\delta_1(i) = 0\ (1 \le i \le K)$
3: for $t \leftarrow 2, n$ do
4:     for $i \leftarrow 1, K$ do
5:         $\delta_t(i) = 0$
6:         for $d \leftarrow 2, t$ do
7:             $L = BestLine(t - d + 1, t)$
8:             if $Err(L) > \varepsilon_r$ then
9:                 Break
10:            else
11:                if $t == d$ then
12:                    $temp = \pi_i b_i(L)$
13:                else
14:                    $temp = \max_j(\delta_{t-d}(j) \cdot a_{ji}) b_i(L)$
15:                end if
16:                if $temp > \delta_t(i)$ then
17:                    $\delta_t(i) = temp$
18:                    $prev_d(t) = t - d$
19:                    $prev_s(t) = j$
20:                end if
21:            end if
22:        end for
23:    end for
24: end for
25: Obtain maximal optimal probability $\delta_n(i)$, which holds
        $$\delta_n(i) \ge \delta_n(j), j \ne i$$
26: Obtain state sequence by backtracking sequence of $prev_s$
27: Obtain line sequence by backtracking sequence of $prev_d$
```

approximation error. Function $Err(L)$ computes the approximation error of L.

Performance analysis. In the traditional Viterbi algorithm, at each time point, it computes K probabilities: $\delta_t(i)$, $i = 1, \cdots, K$. To compute each $\delta_t(i)$, it checks K probabilities $\delta_{t-1}(j)$, $j = 1, 2, \cdots, K$. So the time complexity is $O(nK^2)$ in each round.

In our task, at each time point, we also compute K probabilities: $\delta_t(i)$,$i = 1, 2, \cdots, K$. However, to compute each $\delta_t(i)$, we need to check $(t-1) * K$ possible probabilities $\delta_{t-d}(j)$, $(j = 1, 2, \cdots, K$, $1 \le d < t)$ at most. Thus, the time complexity is $O(n^2K^2)$ in each round. Clearly, for long sequences, it is not feasible. Next, we introduce three pruning strategies to improve the efficiency.

4.2.3 Three Pruning Strategies

We propose three pruning strategies, two of which are lossless and the third is lossy (with respect to whether the final result is the same as that of the exact approach discussed before).

Strategy 1: Prune with threshold ε_r. The first one is based on the requirement that the approximation error of each line cannot exceed threshold ε_r. So we use ε_r to filter state transitions which need not be checked. Assume the current time point is t, and we compute $\delta_t(i)$. We check optimal probabilities of previous time points from $t - 1$ to 1. If we find a time point t', which satisfies

$$Err(BestLine(t' - 1, t)) > \varepsilon_r \text{ and } Err(BestLine(t', t)) < \varepsilon_r$$

it means the approximation error of the line covering interval $[t' - 1, t]$ must exceed ε_r. So later we need not check the optimal probabilities before t'. Note that while the process continues, t' will move forward gradually.

Strategy 2: Prune with optimal possibility. The second pruning strategy uses obtained candidates of optimal probabilities to filter useless optimal probabilities. In algorithm 1, to compute $\delta_t(i)$, we need to check lots of previous optimal probabilities. Each time a previous optimal probability is checked, we get a candidate of $\delta_t(i)$. Since computing optimal probability finds the maximal one, we can make pruning based on the obtained candidates.

We maintain all previous optimal probabilities in a list, denoted as LT, in which all probabilities are sorted in descending order. The optimal probability will be deleted from LT once the approximation error of the line L generated by it exceeds ε_r. We use LT_i to indicate the i-th optimal probability in LT. To compute $\delta_t(i)$, we check the entries in LT from top to bottom. Assume after we check the top-j entries in LT, the obtained maximal candidate is δ'. Then we check the $(j+1)$-th entry, LT_{j+1}, in LT. We compute whether the following inequality holds:

$$LT_{j+1} \cdot a_{max}(i) \cdot b_{max}(i) < \delta'$$

where $b_{max}(i)$ is the maximal output probability generated by state i and $a_{max}(i)$ is the highest transition probability from any state to state i.

If it holds, it indicates that from LT_{j+1}, as well as all entries after it, we cannot obtain a candidate of $\delta_t(i)$ larger than δ'. So δ' is the final $\delta_t(i)$. We add it into LT for later computing.

Strategy 3: Prune with boundary points. Although the first two strategies reduce the unit cost of computing an optimal probability, the process may still be time consuming since it needs to compute optimal probabilities for each time point. In this strategy, we reduce the number of points where we need to compute optimal probabilities, which can greatly speed up the process.

In fact, the goal of the extended Viterbi algorithm is to find the optimal line sequence, which is determined by boundary points of lines. If we only compute optimal probabilities on these points, instead of all time points, we can speed up the process greatly, while the accuracy doesn't suffer too much. An important question is: *which points are more likely to be boundaries in the optimal segmentation*? We answer this question with the following observation:

OBSERVATION 1. *If two neighboring lines before and after t have apparently different slopes, t is more likely to be a boundary.*

The reason is that if the two neighboring lines have similar slopes, it is more likely that they are merged into a line, and consequently t is a point in the middle of this line, instead of being a boundary. We illustrate the observation in Figure 7. It is obvious that points A and C should be boundaries. For D and E, the lines before and after them have similar slopes, so they are less possible to be the boundaries than A and C.

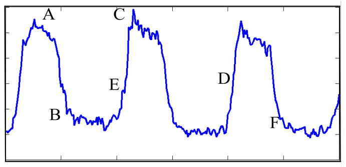


Figure 7: Boundary points

We choose points based on the above observation. Remember that in the initial phase, we segment the time series in a bottom-up way. At each step, two neighboring lines are merged and the point connecting them changes from a boundary to a middle point.

The sooner a boundary point is changed, the less likely that it is a boundary. So we sort all the time points according to the order they become middle points, and select the last N time points to form a boundary candidate list, where N is a user-specified parameter. Then, we execute Algorithm 1 only on these points. Continuing the example in Figure 7, assume we only consider these six points. They should be sorted as:

$$\cdots, D, E, B, F, A, C$$

If we just select 4 points from them to run Algorithm 1, C, A, F and B will be selected.

Strategies 1 and 2 are lossless ones and don't affect the accuracy, while the third one is a lossy strategy, since it may cause some points, which should be optimal segmentation boundaries, to be missed. However, our method of choosing points guarantees that we choose the points which are most likely to be the boundaries. Experimental results verify that this strategy can reduce the time consumption dynamically, while keeping the accuracy similar to the exact model.

4.3 Updating pHMM

Assume in round k, the current pHMM is λ^{k-1}. The obtained optimal line sequence and state sequence are denoted as $\mathbf{L}^k$ and $\mathbf{s}^k$ respectively. We update the current pHMM, so that it can generate $\mathbf{L}^k$ and $\mathbf{s}^k$ with the largest probability. Let $\mathbf{L}^k = (L_1, L_2, \cdots, L_m)$ and $\mathbf{s}^k = (s_1, s_2, \cdots, s_m)$. Note that the number of lines in $\mathbf{L}^k$, m, is possible to vary over different rounds.

Transition probabilities and initial probabilities are updated according to state sequence $\mathbf{s}^k$. To update output probabilities, we cluster the lines in $\mathbf{L}^k$ according to the corresponding states. Specifically,

$$C_i = \{L_j | s_j = i\}, i = 1, 2, \cdots, K$$

Then we update the mean and the variance of slopes and lengths with the method in the initial phase. After that, we obtain the pHMM in this round, denoted as λ^k. It happens that certain states in λ^{k-1} disappear from λ^k, if these states don't occur in state sequence $\mathbf{s}^k$.

Finally, we prove the correctness of the refinement approach, that is, the new production probability in the current round is not less than that in the last round.

THEOREM 1. *The production probability of round k is not less than that of round $k-1$, that is:*

$$P(\mathbf{L}^k, \mathbf{s}^k | \lambda^k) \geq P(\mathbf{L}^{k-1}, \mathbf{s}^{k-1} | \lambda^{k-1})$$

PROOF. Since $\mathbf{L}^k$ and $\mathbf{s}^k$ are the optimal observation sequence and state sequence which has maximal probability based on the pHMM in last round, it holds:

$$P(\mathbf{L}^k, \mathbf{s}^k | \lambda^{k-1}) \geq P(\mathbf{L}^{k-1}, \mathbf{s}^{k-1} | \lambda^{k-1}) \quad (6)$$

Next, since the parameters in λ^k are the results of maximum likelihood estimation. So it holds

$$P(\mathbf{L}^k, \mathbf{s}^k | \lambda^k) \geq P(\mathbf{L}^k, \mathbf{s}^k | \lambda^{k-1}) \quad (7)$$

Combining Eq. 6 and Eq. 7, we can obtain

$$P(\mathbf{L}^k, \mathbf{s}^k | \lambda^k) \geq P(\mathbf{L}^{k-1}, \mathbf{s}^{k-1} | \lambda^{k-1}) \quad (8)$$

$\square$

5. APPLICATIONS OF THE MODEL

The pHMM reveals the system dynamics from the time series it produces. With knowledge of the underlying system, we can deal with some advanced tasks. In this section, we introduce how to use pHMM to perform these tasks.

5.1 Multi-step Value Prediction

Different from traditional prediction models, which make predictions based on previous values, the pHMM makes predictions based on patterns. To be specific, assume we already learn a pHMM from the training time series. Now, we go through a testing time series and predict the values based on the learned pHMM. Let t be the current time point, we first detect the current state. Then based on the current state, we predict the values of $t+1, t+2, \cdots$. Multi-step value prediction is very useful in the application of system monitoring, in which earlier detection of the anomaly value is critical.

Assume we already obtain a pHMM, and $Y = \{y_1, y_2, \cdots\}$ is the time series we monitor to make predictions. We first detect the current state, with the extended Viterbi algorithm in Section 4. Pruning strategies 1 and 2 can both be used here, but the third one is not, since it requires that we already have the segmentation of the whole time series, which is not available in online monitoring.

To speed up the process, we utilize another pruning strategy introduced in [18], which can be executed on the fly. It aims at reducing the number of points, at which we checked the previous optimal probabilities to compute the current optimal probability. Specifically, if we find that a past point is not likely to be a boundary, we delete all the optimal probabilities at this point from LT.

We do it as follows. Given a minimal distance D and a minimal percentage P, we dynamically remove points y_t and $y_{t'}$ if they hold

$$|t - t'| < D \text{ and } \frac{|y_t - y_{t'}|}{|y_t + y_{t'}|/2} < P$$

If these two inequalities hold, it means that these two points are near and there is no large fluctuation between their values. By this strategy, we only need to maintain a small number of optimal probabilities, by which we can compute current optimal probability efficiently.

Assume the new arrival value is y_t, we compute $\delta_t(i), 1 \leq i \leq K$, and obtain the optimal line sequence and state sequence up to t. Then we make predictions of future values based on the current state.

5.2 Trend Prediction

In many applications, users are not interested in forecasting specific values. Instead, they are interested in the evolving of trends. With pHMM, we can predict the future trends easily. For example, we can answer queries like: *what is the trend of time series after 10 minutes*; or *when will the time series end the current downward trend and enter an upward trend*.

The approach is similar with that of multi-step value prediction. When monitoring a time series, we first detect the current state online, and then make prediction based on this. For example, to estimate how long the system will stay in the current state, we compute the difference between the mean duration of the current state, and its current duration; A more useful case is to estimate the trend in a future period, such as predicting the temperature trend tomorrow between 9:00-10:00am. To answer this query, we first predict the time span of the next state based on transition probability. If it covers the period in question, we use the mean of the slope in the next state as the estimated trend; if not, we predict the state after the next state. We continue this process until the period in question is covered.

5.3 Pattern-based Correlation Detection

Correlation detection is an important operation in time series mining. Measurements, like the correlation coefficient, can tell whether similar subsequences exist between two time series. However, it is advantageous to detect a more general correlation between two time series based on patterns. Consider the example shown in Figure 8. When a burst, P_1, occurs in time series X, a more stable upward trend, P_2, will occur in time series Y with probability 80%. Note that P_1 and P_2 can be two totally different patterns. Moreover, they can have different lengths and not align. For example, occurrences of P_2 are always 5 seconds later than those of P_1. In general, we learn the correlations based on patterns, instead of values. We call this type of correlation the pattern-based correlation.

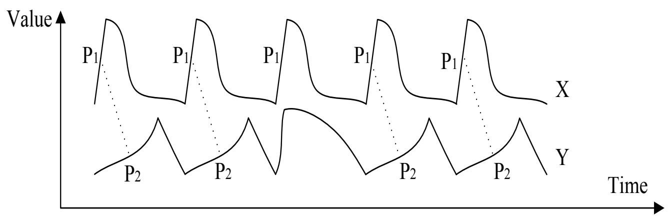


Figure 8: General correlation

pHMM can be used to find the pattern-based correlation effectively. Given two time series X and Y, we learn two pHMMs for them respectively. Then we compute the correlations between patterns in these two pHMMs. We use two criteria to measure the general correlation. The first one is frequency, which measures whether these two patterns have a similar number of occurrences. Assume we measure the correlation between pattern P_1 in X and P_2 in Y. Let P_1 occur m_1 times and P_2 occur m_2 times ($m_1 \leq m_2$). The first criterion is computed as

$$f(P_1, P_2) = \frac{m_1}{m_2}$$

The second criterion is about how well their occurrences align. It is better if most of their occurrences have similar gaps, or delays. To measure the second criterion, we compute the minimal average of their gaps' square. Since there exist many possible matchings, we choose the one with minimal gap-square-average. In the example shown in Figure 8, the best matching of P_1 and P_2 is illustrated by dotted lines.

If $m_1 \leq m_2$, we pick out m_1 occurrences of P_2, which can best match occurrences of P_1. Let $\{c_j\}$, $1 \leq j \leq m_1$, be the central time points of occurrences of P_1, and $\{c_{i_j}\}$, $1 \leq i_j \leq m_2$, be central points of m_1 occurrences of P_2. We measure the second criterion as follows:

$$g(P_1, P_2) = \min \left\{ \frac{1}{m_1} \sum_{j=1}^{m_1} (c_j - c_{i_j})^2 \right\}$$

which can be efficiently computed with a dynamic programming approach. We combine these two criteria to measure the general correlation between patterns P_1 and P_2 as:

$$GC(P_1, P_2) = \frac{g(P_1, P_2)}{f(P_1, P_2)}$$

The smaller $GC(P_1, P_2)$, the more correlated patterns P_1 and P_2.

6. EXPERIMENT

We conducted extensive performance tests for our approach. All algorithms are implemented in Matlab 7.0, and are tested on a PC with Intel Pentium IV 2.4GHz CPU and 2GB RAM.

6.1 Experiment Setup

We choose two types of real-life time series, the first is a relatively regular time series, and the second is less regular. We normalize values of both datasets into interval [0,1].

1. Power demand time series [27]. It is the 15 minutes of averaged values of power demand for research facility (ECN) in the full year 1997 and it contains 35,050 data points. The training subsequence and testing subsequence both contain 2,000 time points. Since the fluctuation of power demand in each day is similar, it is more regular.

2. Spot prices time series. It contain the spot prices for daily exchange rates of 12 kinds of currencies relative to the US dollar. For each currency, there are 2,567 (work-)daily spot prices over the period 10/9/86 to 8/9/96. In each experiment, we randomly select one currency.

We conducted three groups of experiments. In the first group of experiments, we test the efficiency of the proposed approach, especially of the three pruning strategies. In the second group of experiments, we analyze the impact of parameters ε_r and ε_c. Finally, we test the effectiveness of pHMM for answering three types of queries introduced in Section 5.

6.2 Experiment Results

Efficiency. We conduct the experiments on the Power dataset. We train the model in three scenarios: no pruning, pruning with strategy 1, pruning with both strategies 1 and 2. In each scenario, we compare the runtime under different N, the selection ratio of boundary points. For example, 0.05 means we choose $0.05*2000 = 100$ time points to train the model. The results are shown in Table 1. It can be seen that pruning reduces runtime significantly, especially pruning with both strategies 1 and 2.

N	0.05	0.1	0.5	1
No-Prune	96	381	9265	-
Prune 1	41	93	689	1921
Prune 1&2	29	70	425	934

Table 1: Runtime Comparison(s)

Since pruning loses information, we test its influence on the quality of learned pHMM. As a larger N leads to longer training time, it is desirable that a relatively small N can still build a high-quality model. We directly measure the quality of the model with the whole production probability. We test it on both the Power dataset and the Spot dataset. The production probability is replaced by the value of negative logarithm, $-\log(P(\mathbf{L}, \mathbf{s}|\lambda))$. The results are shown in Figure 9.

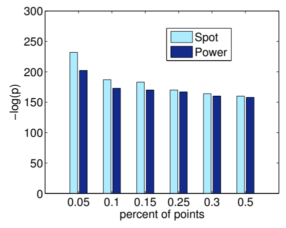


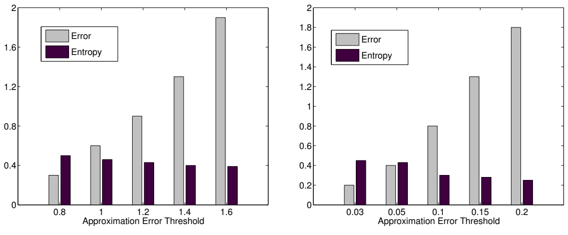
Figure 9: Influence of N

We can see that in both datasets, when the percentage is equal to or larger than 0.1, the production probability is not affected much, which verifies the effectiveness of this pruning strategy. In other words, we can still learn high quality models by looking at a small number of time points. In our experiments, we set N as 0.1.

The influence of ε_r and ε_c. In this group of experiments, we test the influence of two parameters, ε_r, approximation error threshold in time series segmentation, and ε_c, relative error threshold in line clustering. The experiments are conducted on the Spot dataset and the Power dataset. The following two measurements are used to measure the quality of the learned pHMM:

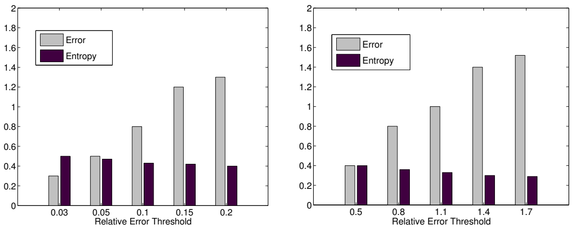
- Residual error per time point. It measures how accurately pHMM represents the original time series by segment lines. It is computed as follows. After obtaining the model and the optimal segmentation, $\mathbf{L}$, for each interval L_i, we use the 'central' line of state s_i to approximate the subsequence. The errors are summarized and then divided by the length of the whole time series. A smaller residual error means the learned model represents the time series more accurately.

- Entropy. It is the second criterion used in clustering segment lines in the initial phase, I in Eq. 4. It measures the certainty of the states about the next state. The smaller the entropy, the more certain the states are about the following state.



(a) Spot Dataset (b) Power Dataset

Figure 10: Residual Error and Entropy Vs. ε_r

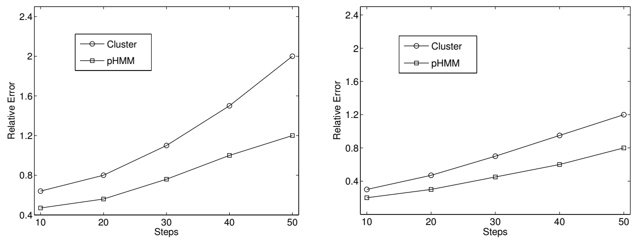


(a) Spot dataset (b) Power dataset

Figure 11: Residual Error and Entropy Vs. ε_c

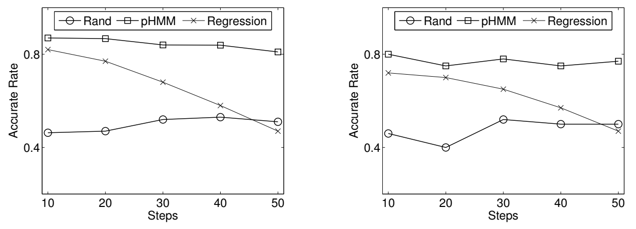
Figure 10 shows the results with varying ε_r. To make the comparison of two measurements clearer, we scale residual error by a factor of 0.002. It can be seen that in both datasets, when ε_r increases, the residual error also increases. It means larger ε_r will reduce the accuracy of the learned model to approximate the time series. In contrast, when ε_r increases, the entropy decreases. So in different applications, users can set ε_r according to the following requirement: To represent the time series more accurately, users should choose a smaller ε_r; to be more certain about states, users should choose a larger ε_r.

The results of varying ε_c are shown in Figure 11. The same scale is used as in the experiment of the residual error. It can be seen that ε_c has the same characteristics with ε_r. When it increases, since the relative error of the lines in a cluster is larger, more residual error is generated. On the contrary, entropy decreases when ε_c increases.



(a) Spot dataset (b) Power dataset

Figure 12: Accuracy of Trend Prediction



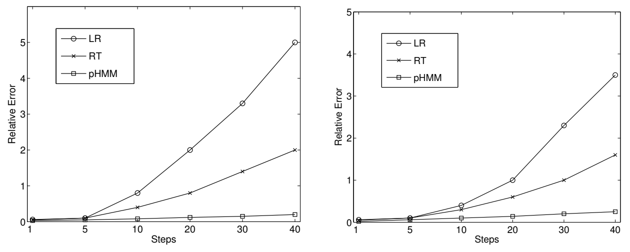
(a) Spot dataset (b) Power dataset

Figure 13: Accuracy of binary trend prediction

Trend Prediction. The major advantage of pHMM is that it can be used to perform the tasks discussed in Section 5. In this experiment, we first test the effectiveness of the proposed approach for trend prediction.

The experiments are conducted on both the Spot dataset and the Power dataset respectively. We compute the prediction accuracy as follows. In the testing time series, we randomly select 50 time points. At each time point, we make the trend predictions after 5 different gaps, 10, 20, 30, 40, 50. Assume the current time point is t. For each gap d, we predict the trend of 20-length subsequence $[t+d+1, t+d+20]$. The trend is represented by a segment line. We compare the true trend with the estimated trend by our approach. For example, at time point 100, we predict the trend of $[111, 130]$, $[121, 140] \cdots$. We compute the relative error as $\frac{e(l)-b(l)}{b(l)}$, where $e(l)$ is the slope of the estimated line, and $b(l)$ is that of the best-fit line. The accuracy of two models are compared. The first one is the pHMM model obtained by only clustering, the second by both clustering and refinement. Through this experiment, we test the effectiveness of both the cluster algorithm and the refinement process. The results are shown in Figure 12. It can be seen that in the cluster-based model, the relative error does not increase dramatically as the gap increases, which verifies the effectiveness of our clustering approach. For both datasets, especially the Spot dataset, which is less regular than the Power dataset, pHMM is more accurate. This result clearly demonstrates the effectiveness of the refinement process.

We also conduct experiments for binary trend prediction. For each testing time series, we make a binary trend prediction: up or down. Three approaches are compared: Rand, Regression and pHMM. The Rand approach makes random guesses. The Regression approach predicts by first computing the linear regression of the time series in the current time window, and then making predictions with it. For each dataset, we also pick 50 time points to predict the trend of the next 10, 20, 30, 40 and 50 steps. Figure 13 shows the average accuracy of all 50 time points. It can be seen that pHMM is more accurate than other approaches.



(a) Spot dataset (b) Power dataset

Figure 14: Accuracy of Multi-step Value Prediction

Multi-step Value Prediction. In this experiment, we test the accuracy of pHMM for multi-step value prediction. The experiments are conducted for both Spot and Power datasets. We select 50 points randomly. For each selected point, we predict the values after 1, 5, 10, 20 ,30 and 40 steps respectively. The pHMM is compared with two classic approaches: a linear regression model (denoted by LR) and a regression tree (denoted by RT) [1]. Both approaches are trained based on values of the previous 20 time points. For the linear regression model, we learn the model in the online way. That is, at each selected time point, the model is learned based on the previous 20 values. Then all 5 predictions are made based on this model. For the regression tree, we train the model before making predictions. First, we build a training dataset from the original time series, where each row contains 20 input values and 1 output value. Then a regression tree is learned from this dataset. We make predictions as follows. Assume the current time point is t. First we predict the value at $t+1$ based on values at $t-19, t-18, \cdots, t$. Then we consider the estimated value at $t+1$ as an input value, and estimate the value at $t+2$. This process continues until all 5 values at $t+1, t+10, t+20, t+30, t+40$ are estimated.

We use relative error as the measurement. The results are shown in Figure 14. It can be seen that on both datasets, pHMM is more accurate than the other two approaches. In LR, the values are predicted based on the current line, which obviously cannot accurately predict the values after a large gap. Unlike LR, RT contains multiple prediction rules, so when the step increases, it still can find the fittest model to make predictions. However, since it has no knowledge whether the estimated value is appropriate, it cannot adjust the next predictions, even if the previous estimated value has a large error. Hence when the step increases, its accuracy drops. The results show that these two models are applicable for predicting the next value, but not for multi-step prediction. To increase the accuracy for these approaches, an alternative way is to train a specific model for the prediction with a specific step, but it needs to learn a lot of models, which is infeasible in practice. In contrast, pHMM can make predictions accurately, even when the step increases. It verifies the advantage of the pattern-based model.

Pattern-based Correlation Detection. Finally, we test the effectiveness of pHMM for pattern-based correlation detection. We conduct this experiment on the Spot dataset. Since all time series are over the same time period, we hope to find correlations between price trends of different currencies. We use time series for "French Franc" as the reference time series, and compute correlations between it and 5 other currencies (including "Australian Dollar", "Belgian Franc", "Canadian Dollar" and so on). For each time series, we train the pHMM model. Then, for each state of "French Franc", we compute its pattern-based correlation(GC) with any state in the 5 other time series. Table 2 shows both minimal GC and average GC. For example, in the first row, Minimal GC means the minimal GC computed between any state pair in which one is from "French Franc" and the other is from "Australian Dollar". Average GC is the average of all GCs between all state pairs from two currencies.

	Minimal GC	Average GC
Australian Dollar	88.33	130.45
Belgian Franc	**41.25**	**50.62**
Canadian Dollar	120.41	160.37
German Mark	60.57	120.46
Japanese Yenr	130.59	170.96

Table 2: Pattern-based Correlation

It can be seen that 5 target currencies demonstrate different correlations. The fluctuation of "Belgian Franc" is most similar to that of "French Franc", so both the minimal and average GC are minimal compared to all other currencies. "German Mark" also has a similar state with "French Franc", although its average GC is still high. For all the 3 other currencies, their states are more different with those of "French Franc". This experiment shows that with pattern-based correlation, we can compare time series in the higher level.

7. RELATED WORK

Time series representation and forecasting. A number of techniques have been proposed in the literature to represent time series with patterns. In [7], authors give an extensive performance comparison of popular time series representation approaches. They can be categorized into two groups. Those in the first group split the whole time series into disjoint segmentations and represent each segment with the mean value or a regression line, such as PLA [11] and PAA [10]. These techniques can provide an approximate shape of the time series. However, they don't exploit the relationships between segments. Those in the second category represent time series with a few dominant coefficients of certain transformations, such as DFT [8] and DWT [2]. However, the coefficients are not interpretable, that is, knowing the coefficients in the frequency domain does not necessarily enable us to understand how the system works.

Time series motifs are approximately repeated subsequences of a longer time series stream. Motifs are defined and categorized using their support, distance, cardinality, length, dimension, underlying similarity measure, etc. Many researchers have introduced techniques to find them efficiently in the case of a large database or streams [16]. But still, the set of motifs cannot provide us with the whole picture of the time series.

Recently, some works dealing with relations between patterns has been proposed. Pattern Growth Graph (PGG) [22] detects and manages variations over pseudo periodical streams. It first splits time series into segments, each of which is an occurrence of the pseudo period, and then describes time series by several connected lines. However, this work can only deal with pseudo periodical time series, and is not applied to the general time series. In [19], a multi-scale schema is proposed to compress time series. It uses techniques such as FFT and random projection to represent the original time series. However, patterns are still isolated, and hence cannot be used to make predictions.

Time series forecasting has been a topic of extensive research [1, 3]. In particular, many tools for forecasting and processing time series appear in statistics and signal processing fields. The traditional method includes ARIMA [1]. Other well-known machine learning approaches include Bayesian Network, Regression Tree, CART and Random Forests [28]. These methods try to capture relationships between the predicted value, y_t, with observed values $y_{t-1}, \cdots, y_{t-n}$. Our approach is different to them, since our goal is to build a model based on patterns instead of single values.

Markov model and the related extension models. The hidden Markov model (HMM) assumes that the states are unobservable and the observation symbols are emitted by the states according to the output probability. A well-known problem of HMM is that the first-order assumption restricts it from accurately modeling the time series data with highly varied dynamics, as it is often that the future state not only depends on the present state, but also the past states. To increase the accuracy of modeling, the n-gram model was presented [15], but the complexity and the learning cost increase exponentially when n increases.

In contrast with n-gram, variable length Markov model (VLMM) learns a minimum set of contexts with variable lengths to model the high-order Markovian process [20]. VLMM aims to extend the states to variable length contexts, which is composed of several connected states. It reduces the number and complexity of contexts by allowing the context to have variable lengths. However, VLMM is limited as an observable Markov model and not a hidden Markov model. In other words, all states are observable, and no output probability is needed.

Another Markov model related to our work is Variable length hidden Markov model (VLHMM) combines the advantages of both HMM and VLMM [26]. Instead of generating observations using single states, VLHMM extends states to contexts, which are composed of variable of states. An EM algorithm is used to learn the parameters of the model, which aims to maximize the likelihood of generating time series by the model. The difference between these models and the proposed pHMM is that they take the value of each time point as an observation, while in pHMM, the observations are patterns.

8. CONCLUSION

In this paper, we reveal the dynamics of a complex system by learning a pattern-based hidden Markov model from the time series data generated by this system. The biggest difference between a pHMM and a traditional HMM is that in pHMM, observations are not given, but learned from the data as well. We propose an approach to learn patterns (observations) and the model simultaneously. Furthermore, three pruning strategies are proposed to speed up the learning process. With pHMM, we are able to perform pattern based tasks, such as trend prediction and pattern-based correlation detection. Empirical results on real datasets demonstrate the feasibility and effectiveness of the proposed approach.

9. REFERENCES

[1] G. Box, G. Jenkins, and G. Reinsel. *Time series analysis: forecasting and control*. Prentice Hall, 1994.

[2] K. Chan and W. Fu. Efficient time series matching by wavelets. In *ICDE*, 1999.

[3] C. Chatfield. *The analysis of time series: an introduction*. Chapman & Hall/CRC, 2004.

[4] S. Chen, H. Wang, and S. Zhou. Concept clustering of evolving data. In *ICDE*, 2009.

[5] S. Chen, H. Wang, S. Zhou, and P. Yu. Stop chasing trends: Discovering high order models in evolving data. In *ICDE*, 2008.

[6] G. Dangelmayr, S. Gadaleta, D. Hundley, and M. Kirby. Time series prediction by estimating markov probabilities through topology preserving maps. In *SPIE*, 1999.

[7] H. Ding, G. Trajcevski, P. Scheuermann, X. Wang, and E. Keogh. Querying and mining of time series data: Experimental comparison of representations and distance measures. In *VLDB*, 2008.

[8] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos. Fast subsequence matching in time-series databases. In *SIGMOD*, 1994.

[9] X. Gu and H. Wang. Online anomaly prediction for robust cluster systems. In *ICDE*, 2009.

[10] E. Keogh, K. Chakrabarti, M. Pazzani, and S. Mehrotra. Dimensionality reduction for fast similarity search in large time series databases. *KAIS*, 2000.

[11] E. Keogh and M. Pazzani. An enhanced representation of time series which allows fast and accurate classification, clustering and relevance feedback. In *SIGKDD*, 1998.

[12] E. Keogh, S.Chu, D. Hart, and M. Pazzani. An online algorithm for segmenting time series. In *ICDM*, 2001.

[13] E. J. Keogh. A decade of progress in indexing and mining large time series databases. In *VLDB*, 2006.

[14] J. Lin, E. J. Keogh, L. Wei, and S. Lonardi. Experiencing sax: a novel symbolic representation of time series. In *DMKD*, 2007.

[15] J. F. Mari, D. Fohr, and J. C. Junqira. A second-order hmm for high performance word and phoneme-based continuous speech recognition. In *ICASSP*, 1996.

[16] A. Mueen and E. Keogh. Online discovery and maintenance of time series motifs. In *SIGKDD*, 2010.

[17] A. Mueen, E. Keogh, and N. Bigdely-Shamlo. Finding time series motifs in disk-resident data. In *ICDM*, 2009.

[18] C. Perng, H. Wang, S. R. Zhang, and D. Parker. Landmarks: A new model for similarity-based pattern querying in time series databases. In *ICDE*, 2000.

[19] G. Reeves, J. Liu, S. Nath, and F. Zhao. Managing massive time series streams with multiscale compressed trickles. In *VLDB*, 2009.

[20] D. Ron, Y. Singer, and N. Tishby. The power of amnesia: Learning probabilistic automata with variable memory length. *Machine Learning*, pages 117–149, 1996.

[21] Y. Tan, X. Gu, and H. Wang. Adaptive system anomaly prediction for large-scale hosting infrastructures. In *PODC*, 2010.

[22] L. Tang, B. Cui, H. Li, G. Miao, D. Yang, and X. Zhou. Effective variation management for pseudo periodical streams. In *SIGMOD*, 2007.

[23] H. Wang, W. Fan, P. S. Yu, and J. Han. Mining concept-drifting data streams using ensemble classifiers. In *SIGKDD*, 2003.

[24] H. Wang, J. Yin, J. Pei, P. S. Yu, and J. X. Yu. Suppressing model overfitting in mining concept-drifting data streams. In *SIGKDD*, 2006.

[25] P. Wang, H. Wang, M. Liu, and W. Wang. An algorithmic approach to event summarization. In *SIGMOD*, 2010.

[26] Y. Wang and L. Zhou. Mining complex time-series data by learning the temporal structure using bayesian techniques and markovian models. In *ICDM*, 2006.

[27] J. J. Wiik and E. R. Selow. Cluster and calendar based visualization of time series data. In *INFOVIS*, 1999.

[28] I. H. Witten and E. Frank. *Data mining: practical machine learning tools and techniques*. Morgan Kaufmann, 2005.