An Algorithmic Approach to Event Summarization

Peng Wang[*]
Fudan University
Shanghai,China
pengwang5@fudan.edu.cn

Haixun Wang
Microsoft Research Asia
Beijing, China
haixunw@microsoft.com

Majin Liu
Fudan University
Shanghai,China
082024017@fudan.edu.cn

Wei Wang
Fudan University
Shanghai,China
weiwang1@fudan.edu.cn

ABSTRACT

Recently, much study has been directed toward summarizing event data, in the hope that the summary will lead us to a better understanding of the system that generates the events. However, instead of offering a global picture of the system, the summary obtained by most current approaches are piecewise, each describing an isolated snapshot of the system. We argue that the best summary, both in terms of its minimal description length and its interpretability, is the one obtained with the understanding of the internal dynamics of the system. Such understanding includes, for example, what are the internal states of the system, and how the system alternates among these states. In this paper, we adopt an algorithmic approach for event data summarization. More specifically, we use a hidden Markov model to describe the event generation process. We show that summarizing events based on the learned hidden Markov Model achieves short description length and high interpretability. Experiments show that our approach is both efficient and effective.

Categories and Subject Descriptors

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

General Terms

Algorithm

Keywords

Event summarization, hidden Markov model, minimal description length

[*]Work done while the author is at Microsoft Research Asia.

1. INTRODUCTION

Many complex systems employ sophisticated record keeping mechanisms that log all kinds of events occurred in the systems. Usually the log consists of a sequence of events, where each event is associated with a timestamp. Event logs contain rich information of the system. For example, in large scale cluster systems, event logs contain information about software and hardware faults and failures. More and more efforts are being devoted to event log analysis, and today, we look in system event logs for patterns of historical activities, assessments of current status, and predictions of future risks.

Event log analysis poses significant challenges. Logs are usually big and noisy; the difficulty of finding patterns of actionable insights from event logs is no less than that of finding a needle in a haystack. Some recent works employs data mining methods for event log analysis, but most approaches remain at the stage of finding frequent episodes in the data. These methods report a large amount of frequent episodes or patterns, but their meanings or significance often go untold. To our best knowledge, none of the existing work explores the relationships among discovered patterns, nor do they reveal how such patterns interact with each other to form the dynamics of the underlying system.

In this paper, we propose an *algorithmic* approach for event summarization. Unlike previous work that outputs a large amount of uncorrelated, local patterns in the data, our approach endeavors to transform the patterns into organic components of the system, so that we can have a global view and a better understanding of the system. Specifically, we wish to find the mechanism that generates the events, and use the mechanism as a summary. It bears some similarity to finding the algorithmic complexity of a string, that is, the shortest and most efficient program that can produce the string. Of course, there is no mechanic ways of doing it. What we do is to make an educated guess, which, as it turns out, not only results in a much more concise summary, but also a much more interpretable summary.

1.1 Event Sequence Summarization

Most of the existing works focus on mining frequent episodes or patterns from event logs. However, frequent episodes alone do not create a global model of the system. Instead of finding frequent patterns, some recent works focus on event sequence summarization. For example, in [10], an event sequence is split into disjoint intervals, and a pattern is generated to describe events in each interval. The result is a sequence of patterns, which constitutes a summary of the original event data. The problem of summarization is an optimization one, as its goal is to find a sequence of patterns with the minimum description length. However, a description that has the minimum length does not necessarily reveal the intricacies of the system. In particular, it does not portray the relationships among the patterns. Such relationships – for example, pattern X occurs periodically about every two hours, or pattern Y always occurs after pattern Z, etc. – are important because in many applications, they improve the understanding of the system. We illustrate its limitations with the following example.

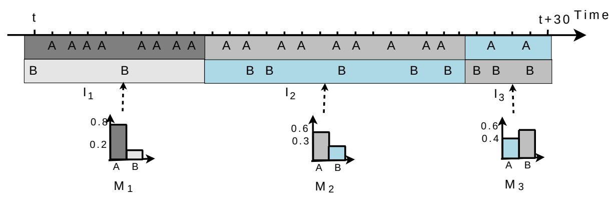


Figure 1: Motivation example

Example 1. *Figure 1 shows an event subsequence that spans time [t,t+30] and contains events of type A and B. The summarization approach presented in [10] segments the sequence into multiple intervals and learns a model of event distribution in each interval. The sequence is divided into 3 intervals I_1, I_2, I_3, which lead to 3 models M_1, M_2, M_3 (darker color represents the high frequency event). The goal of [10] is to split the event sequence in such a way that the resulting models have the minimum overall description length.*

The resulting models – the summarization of the event log – have limited capability of revealing the system dynamics hidden in the event sequence. For instance, it does not provide information such as M_1 is very different from M_3 but is similar to M_2. Nor does it tell us that M_1 is usually followed by M_2. In other words, it does not find the connection between different models. So, in this sense, the approach does not provide a *global structure* of the event sequence.

In summary, the segment-by-segment approach that optimizes for the minimum description length has the following limitations:

- They do not characterize interactions among different models, and it follows that they cannot reveal the dynamics hidden in the data, or predict patterns in the future.
- They focus on best compressing the data. However, for a long event sequence with many repeating models (we will show this is a very likely scenario), the compression ratio is low because copies of same/similar models are stored.

1.2 Our approach

We want to find the mechanism that generates the events and then we encode the mechanism as the summary. Certainly, there is no algorithmic approach for finding such mechanisms. In this paper, we make the following "educated guess": a system operates in different states; in each state, the system exhibits stable behavior; state transitions have certain regularity. This educated guess often enables us to summarize the event data in a much more concise, and a much more interpretable way. In fact, the educated guess is based on the following observations:

1. It is observable that a system operates in different states. For instance, for certain period of time, some events occur frequently, and at another period of time, another set of events occur frequently. In other words, in each state, the system's behavior shows specific characteristics.
2. Once a system is in a state, it tends to remain in that state, until a certain event occurs and leads it to another state. For example, when memory usage is below the physical memory capacity, a system behaves in a certain way and exhibits certain patterns. The situation continues until the memory usage exceeds capacity, at which point the system starts paging and demonstrates another set of patterns.
3. The system alternates among a set of states over and over again. For example, once the memory usage recedes, the system will return to its old state (the one without paging).

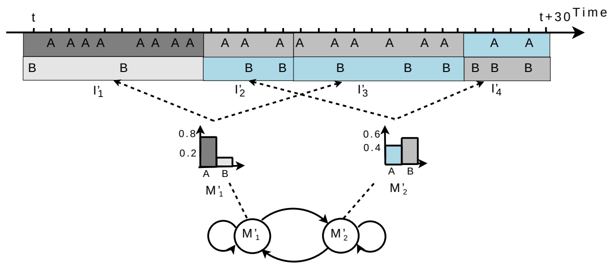


Figure 2: Our approach

These observations motivate us to summarize the event data by understanding the system first. Consider the example in Figure 1 again. Assume from the whole sequence, we are able to learn two states:

- State 1, in which event A occurs frequently and event B occurs less frequently. I_1 and I_3 are instances of this state.
- State 2, in which event A occurs less frequently and event B occurs frequently. I_2 and I_4 are instances of this state.

Given these two states, we summarize [t,t+30] as shown in Figure 2. Specifically, I_2 in Figure 1 is split into 2 intervals: I'_2 and I'_3, which correspond to M'_2 and M'_1 respectively. Furthermore, the two states are not isolated, instead they occur alternately. In other words, we hope to mine the a set of models and the temporal relations among them. Clearly, such information gives us a better understanding of the system, facilitates the prediction of its future operations.

Note that the above result is different from what we have in Figure 1. Moreover, the learned models in our approach cannot be derived by simply clustering and correlating the segment-by-segment summaries in Figure 1, since its results may include many meaningless models.

MDL and HMM. To analyze the advantage of our approach, we resort to the minimum description length (MDL) principle. Our intuition is that if we have a better understanding of the system, we should be able to improve the compression ratio of the event data.

This is indeed the case. Based on the MDL principle, to represent the event sequence, we incur two types of cost: the cost of encoding the models, and the cost of encoding the data given the model. The former depends on how many models we discover, and the cost of encoding each model; the latter depends on how well each model fits the data.

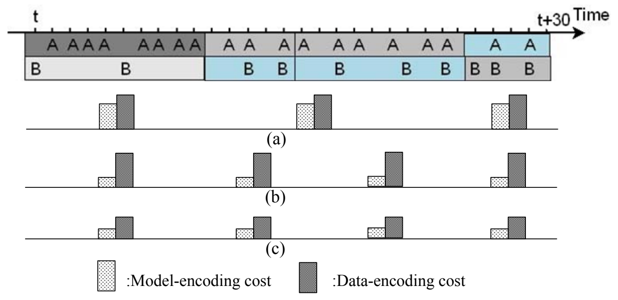


Figure 3: Cost of data summarization: (a) The segment-by-segment approach; (b) Our approach that assumes events are independent of each other; (c) Our approach that takes advantage of the correlation among events

It is not difficult to see why our approach has an edge in compressing compared with the segment-by-segment approach. Continuing with the previous example, Figure 3 illustrates the encoding cost incurred by three different approaches. For each interval, we use two bars to represent the encoding cost. The left bar indicates the cost of encoding model, and the right bar indicates the cost of encoding data given the model. Figure 3(a) shows the cost of segment-by-segment approach, and Figure 3(b) shows that of our approach. The difference comes from the fact that we only encode models once (each model is distinct). For each interval, to tell which model used to describe it, we only encode the ordinal number of the model. It is much more efficient.

Neither of the two approaches consider event correlations: certain events are likely to occur together, while others may occur exclusively to each other. Taking advantage of such properties can surely reduce the encoding cost, because the uncertainty of one event type may be reduced by information of another. In this paper, we propose another approach to exploit the correlations. Figure 3(c) shows its cost. We can see that cost of both encoding model and of encoding data are reduced compared to segment-by-segment approach.

The internal states, which determine the event occurrences that we observe, are hidden and evolving over time. Hence, it is difficult to create a MDL-based objective function and then find the best summary by minimizing it directly. In fact, the internal states are similar to hidden states in a Hidden Markov Model (HMM). Later, we will show the equivalence between the minimal description length and the maximum production probability of a HMM. This enables us to apply HMM based approaches to learn the summary.

2. RELATED WORK

Frequent episode mining. Frequent episode mining is a popular approach for obtaining temporal patterns in event sequences [1, 15, 13, 2, 16]. Agrawal et al [1] proposed several Apriori-based methods to mine sequential patterns. Mannila et al [15] proposed a framework of frequent episode discovery for mining temporal patterns from event sequences. It finds episodes that occur frequently in the time series. The frequency of an episode is defined as the number of windows on the time axis in which the episode occurs at least once. The proposed counting algorithm uses a finite state automata to obtain the frequencies of a set of candidate episodes. Some extensions to this windows-based frequency model have also been proposed [2, 16]. Overall, frequent episodes can be regarded as local models, and they are inadequate to reveal the internal dynamics of the system, as they do not model the relationship among patterns. In other words, the patterns are isolated.

Laxman et al [13, 12] proposed a new frequency measure based on the notion of non-overlapping occurrences of episodes. An efficient approach is proposed to mine the frequent episodes based on this frequency measure. Moreover, the authors build a connection between frequent episodes and HMMs. They introduce a special class of discrete HMM, called the Episode Generating HMM (EGH), and associate each episode with a unique EGH. It has the property that the more frequent an episode is, the more likely it is generated by the corresponding EGH. However, each HMM only represents one episode, and no attempt was made to summarize the entire set of patterns to provides a global view of the system that generates the event sequence. In other words, it still focuses on local patterns in event sequences.

Event summarization. Some recent works conduct event sequence summarization through sequence segmentation [14, 11, 10]. Mannila et al [14] focus on sequences of single event types. Within each interval, the occurrences of events are modeled by a constant intensity model. Koivisto et al [11] proposed a minimum description length approach to identify block structures in genetic sequences. Kiernan et al [10] extend the model proposed in [14] to deal with multiple event types. The MDL principle is used to measure the quality of summarization. But in neither works, they model the relationships between different segments, and segments are isolated. In a sense, we combine the advantages of both frequent patterns and segmentation model to construct a better global model of the event sequence. We mine the patterns that can describe the data in event sequences, as well as the relationships among the patterns.

Time series segmentation. Much work has been done on time-series segmentation [9, 8, 6, 7]. The sub time-series in each segment is represented by a line segment [9] or the mean value of the time series in the segment [8]. The goal is to segment the time series so that the total error is minimized. Thus, they aim at learning the local patterns for each segment, and do not provide connections among the patterns, which make them similar to approaches we discussed before [14, 10]. DFT (Discrete Fourier Transform) based approaches [5] and DWT (Discrete Wavelet Transform) based approaches [17] can learn certain kind of global patterns in time series, but generally, patterns derived by such approaches are not easily interpretable.

Hidden concept learning in stream classification. The idea of using a limited number of concepts to summarize data in an endless data stream [19, 20] has occurred in other mining tasks. To classify stream data, instead of learning new classifiers from the stream, Yang el al [21] finds historical classifiers that are learned from similar data. In [4, 3], the authors assume that the set of hidden concepts that control data generation is fixed. This allows them to first find the set of hidden concepts occurring in the stream, then detect the current concepts and use the corresponding classifier to classify the current unlabeled data. Our work is based on the same assumption. However the problems and the techniques that solve the problems are different. Furthermore, instead of living with fixed number of concepts, we can balance the number of models and the quality of summarization.

3. PROBLEM STATEMENT

An event sequence consists of events that occur at discrete points in time. Specifically, events have m different types $\mathcal{E} = \{E_1, E_2, \cdots, E_m\}$, and an event occurrence is a pair (E, t), where $E \in \mathcal{E}$ and t is its occurrence time, which is represented as an integer in the interval $[1, T]$. Table 1 summarizes the notations used here and in the rest of this paper.

Symbol	Description
T	length of time line
m	number of event types
$\mathcal{M}$	set of models
$\mathcal{M}_{ind}$	models for independent events
$\mathcal{M}_{dep}$	models for correlated events
$\mathcal{I} = (I_1, \cdots, I_k)$	a segmentation of the event sequence
$\sigma(i)$	model corresponding to segment I_i
$n(E, I_i)$	number of time points at which event E occurs

Table 1: Notations

3.1 Models

We want to find the mechanism that generates the event sequence, and use it as its summary. Since it is impossible to find the exact mechanism, we assumed that the hidden mechanism can be simplified to a state machine, and that within each state, the system exhibits stable behavior, or in other words, the events it produces within each state demonstrate stable statistical properties.

Our first task is to model each state, or model the characteristics of the events produced by the system when it operates within a state. In this paper, we adopt a simple model, which only describes the intensity of each event, but ignores the relative order of the events. Specifically, each model M_i is composed of a set of m probabilities:

$$M_i = (p_i(E_1), p_i(E_2), \cdots, p_i(E_m))$$

where $p_i(E_j)$ is the occurrence probability of event type E_j.

Certainly, we can use more complex models. For example, in many applications, the occurrences of events of different types are correlated. In other words, the occurrence/absence of events of a certain type can provide information about the occurrence/absence of events of other types. A model that captures such correlations may reveal more details of the system, and produce a more concise summary of the event sequence.

We use $\mathcal{M}_{ind}$ to denote models that assume events are independent, and $\mathcal{M}_{dep}$ models that assume events are correlated. For $\mathcal{M}_{dep}$, we use conditional probability to describe the occurrences of events of different types. We describe how to learn $\mathcal{M}_{ind}$ first, and discuss the details of $\mathcal{M}_{dep}$ in Section 6.

3.2 Problem Statement

The problem of event log summarization can be stated as follows:

Problem Statement: Given an event log, divide it into a sequence of segments and describe each segment by a model in a given class, such that the overall description has the minimum length.

In this paper, we consider two classes of models, $\mathcal{M}_{ind}$ and $\mathcal{M}_{dep}$. The problem statement implicitly requires us to consider the relationships among the models, which may lead to summaries of shorter description lengths, and shed light on the internal mechanism that generates the sequence.

4. OBJECTIVE FUNCTION AND HMM

In this section, we introduce an objective function based on the MDL principle. To find the summary that has the minimum description length, we establish the equivalence between the summary and the best fitting HMM.

4.1 Objective Function

We partition an event sequence into disjoint segments, and describe each segment by a specific model. In order to measure the quality of the summarization, we propose an objective function Q that is consistent with the MDL principle. According to the MDL principle, we need to encode both the models and the data given the model. In this section, we define the objective function for model class $\mathcal{M}_{ind}$ (denoted as $\mathcal{M}$ below if no confusion should arise). In section 6, we extend it to model class $\mathcal{M}_{dep}$.

For each model $M_i \in \mathcal{M}$, we need to encode m probabilities $p_i(E_1), \cdots, p_i(E_m)$. Each probability needs $\log m$ bits to encode [18]. So the cost of encoding model M_i is $m \log m$ and the cost of encoding $\mathcal{M}$, denoted by C_m, is

$$C_m = |\mathcal{M}| \cdot m \log m$$

where $|\mathcal{M}|$ denotes the number of models in $\mathcal{M}$.

Clearly, C_m only depends on m, the number of event types and $|\mathcal{M}|$, the number of models. It does not depend on the size of the event log. Since it is often the case that $T \gg m$ and $T \gg |\mathcal{M}|$ where T is the size of the event sequence, the cost of C_m is negligible when summarizing large event logs.

The cost of encoding the event sequence includes the following two parts:

- C_s: Cost for encoding the segmentation, which includes encoding the lengths of segments and the mappings from segments to models.
- C_d: Cost for encoding event occurrences.

We first discuss how to encode event occurrences in a single segment I. Assume model $M_i = (p_i(E_1), \cdots, p_i(E_m))$ is used to describe I. The probability of observing all events in I, given model M_i, is

$$p(I|M_i) = \prod_{j=1}^{m} p_i(E_j)^{n(E_j,I)}(1-p_i(E_j))^{n(\bar{E}_j,I)} \quad (1)$$

where $n(E,I)$ is the number of time points in I where E occurs and $n(\bar{E},I)$ is the number of time points in I where E does not occur.

Thus, the number of bits required to describe I, given model M_i,

$$-\log p(I|M_i)$$

Clearly, the better M_i fits I, the larger $p(I|M_i)$, and the smaller $-\log p(I|M_i)$. We use $M_{\sigma(i)}$ to denote the model that describes interval I_i. Given the entire segmentation $\mathcal{I}$ and the model set $\mathcal{M}$, the total number of bits required to describe the event occurrences within all segments is

$$C_d = -\sum_{i=1}^{|\mathcal{I}|} \log p(I_i|M_{\sigma(i)})$$

where $|\mathcal{I}|$ is the number of segments.

The second part of the cost, C_s, is the bits required to encode the segmentation. For each segment, we encode its corresponding model and its length. To encode the corresponding model, a straightforward way is to encode the id of the model. We need $\log(|\mathcal{M}|)$ bits to encode the model for a segment. However, if we know the relationships among the models, we can encode the same information with fewer bits. Intuitively, for models having higher occurrence probabilities, we should use fewer bits to encode them. Let p_i $(1 \le i \le |\mathcal{M}|)$ denote the prior probabilities of the models, and p_{ij} $(1 \le i,j \le |\mathcal{M}|,\ i \ne j)$ denote the probability of occurrence of model M_i followed by that of model M_j. For the first segment I_1, we use prior probability $p_{\sigma(1)}$ to represent its model. For I_k's $(k \ge 2)$, instead of encoding the model directly, we encode it based on the model of the previous segment, I_{k-1}. So, the cost required to encode the mapping from segments to models is:

$$C'_s = -\log p_{\sigma(1)} - \sum_{k=2}^{|\mathcal{I}|} \log p_{\sigma(k-1)\sigma(k)}$$

We encode the length of each segment by taking into consideration the probability distribution of the lengths. We can derive the distribution by finding out how long the system is likely to remain in a state. The probability that the system stays in state i is p_{ii}. Thus, for a segment I described by model M_i, the probability that its length is $|I|$ is proportional to $p_{ii}^{|I|-1}$. We perform an transformation, and use more bits to encode an infrequent length, and fewer bits to encode a frequent length. So the cost required to encode the lengths of all segments is

$$C''_s = \sum_{k=1}^{|\mathcal{I}|} (|I_k| - 1) \cdot (-\log p_{\sigma(k)\sigma(k)})$$

The total cost of encoding the segmentation is the summation of C'_s and C''_s:

$$C_s = C'_s + C''_s$$

We uses a set of probabilities, p_i and p_{ij}, $1 \le i,j \le |\mathcal{M}|$, to describe the dynamics of the system. These probabilities also need to be encoded themselves. However, since i and j are bounded by $|\mathcal{M}|$, which does not increase when the size of event log increases, we do not include it in the objective function Q, just as C_m. (in the experiments, we include both of them in the total cost.)

Finally, the objective function comes to:

$$Q(\mathcal{M}, \mathcal{I}) = C_d + C_s \quad (2)$$

Our goal is to partition the event log into a sequence of segments where each segment maps to a specific model such that Eq. 2 is minimized.

4.2 Hidden Markov Model

It is a challenging task to find the segmentation that minimizes Eq. 2. If we know the set of hidden models $\mathcal{M} = \{M_1, \cdots, M_{|\mathcal{M}|}\}$, it will not be difficult to partition the event sequence into non-overlapping occurrences of each context. If we know the partition $\mathcal{I} = \{I_1, \cdots, I_{|\mathcal{I}|}\}$, then we can cluster similar segments to derive $\mathcal{M}$. The challenge is, of course, neither $\mathcal{M}$ nor $\mathcal{I}$ is known.

Naïve solutions to this problem either fail to scale or fail to find the optimal summarization. For an event log spanning time $[1,T]$, there are $T^{|\mathcal{I}|}$ possible segmentations. Moreover, for each segmentation $\mathcal{I}$, there are $|\mathcal{I}|^{|\mathcal{M}|}$ possible mappings between segments and models. So it is infeasible to search the whole space to solve the optimization problem. A more feasible approach is given in [10], where dynamic programming is combined with a greedy approach to summarize an event log. However, the approach does not have knowledge of the internal dynamics of the system, and its segment-by-segment summarization is sub optimal as important information about the system is not fully utilized.

In this paper, we propose an HMM-based approach to solve this problem. To facilitate the discussion, we use an m-dimensional column vector $o_t = (o_t^1, \cdots, o_t^m)'$ to represent observed events at time t: if E_i occurs at time t, then we have $o_t^i = 1$, otherwise $o_t^i = 0$. Hence, the $m \times T$ array $O = (o_1, \cdots, o_T)$ represents the event log over time $[1,T]$.

Given an observed sequence of events O, we learn an HMM λ, and the optimal state sequence $\mathbf{s}$ corresponding to O, so that the *production probability* $P(O, \mathbf{s}|\lambda)$ is maximized. In Section 4.3, we prove that minimizing the objective function $Q(\mathcal{M}, \mathcal{I})$ is equivalent to maximizing the production probability. In other words, the learned λ and $\mathbf{s}$ enable us to find the model set $\mathcal{M}$ and the segmentation $\mathcal{I}$, which maximize the objective function.

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

- A set of states $S = \{1, 2, \cdots, K\}$.
- State transition probabilities $A = \{a_{ij}\}$, in which, a_{ij} is the probability of state i transiting to state j.
- Output probabilities $B = \{b_i(o)\}$, where i is a state, m-dimensional vector $o = (o^1, \cdots, o^m)$ is an observation, and $b_i(o)$ is the probability of observing o when the system is in state i.
- Initial probabilities $\pi = \{\pi_i\}$, in which π_i is the probability of system beginning in state i.

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,

$\mathbf{s} = (s_1, \cdots, s_T)$:

$$P(O, \mathbf{s}|\lambda) = \pi_{s_1} b_{s_1}(o_1) \prod_{j=2}^{T} a_{s_{j-1}, s_j} b_{s_j}(o_j) \tag{3}$$

where s_t is the state at time point t. A larger production probability means that λ and the state sequence $\mathbf{s}$ can describe O better. The optimal state sequence is the one maximizing the production probability.

4.3 Equivalence between $p(O, \mathbf{s}|\lambda)$ and Q

First, we establish connections between terminologies.

- Each HMM state i corresponds to model $M_i \in \mathcal{M}$.
- In state transition probabilities, a_{ij} is equivalent to the context switch probability p_{ij}.
- Output probabilities $b_i(o)$, $1 \leq i \leq K$, are defined based on occurrence probabilities in M_i:

  $$b_i(o) = \prod_{j=1}^{m} q_{ij}(o)$$

  where

  $$q_{ij}(o) = \begin{cases} p_i(E_j) & \text{if } o^j = 1; \\ 1 - p_i(E_j) & \text{if } o^j = 0. \end{cases}$$

- Initial probabilities are equivalent to probabilities of models, that is, $\pi_i = p_i$, $1 \leq i \leq K$.

Second, we establish the connection between the state sequence $\mathbf{s}$ and the segmentation $\mathcal{I}$. Given a state sequence $\mathbf{s} = (s_1, s_2, \cdots, s_T)$, we can obtain a segmentation by merging continuous time points which correspond to the same states into a segment. Finally each segment corresponds to a single state, which represents the model for the segment.

Now, we discuss the equivalence between production probability and objective function. We first decompose production probability $P(O, \mathbf{s}|\lambda)$ into two parts:

$$p(O, \mathbf{s}|\lambda) = \prod_{j=1}^{T} b_{s_j}(o_j) \cdot \pi_{s_1} \prod_{j=2}^{T} (a_{s_{j-1} s_j}) \tag{4}$$

Let $P_1 = \prod_{j=1}^{T} b_{s_j}(o_j)$ and $P_2 = \pi_{s_1} \prod_{j=2}^{T} (a_{s_{j-1} s_j})$.

Cost of encoding event occurrences. By some algebraic transformation, we have:

$$\begin{aligned} -\log P_1 &= -\log \prod_{j=1}^{T} b_{s_j}(o_j) \\ &= -\log\left(\prod_{i=1}^{|I|} \prod_{j \in I_i} b_{s_j}(o_j)\right) \\ &= -\log\left(\prod_{i=1}^{|I|} P(I_i | M_{\sigma(i)})\right) \\ &= C_d \end{aligned} \tag{5}$$

Cost of encoding segmentation. Similarly, $-\log P_2$ is equivalent to C_s, the cost of encoding the segmentation:

$$\begin{aligned} -\log P_2 &= -\log(\pi_{s_1} \prod_{j=2}^{T} a_{s_{j-1} s_j}) \\ &= -\log(\pi_{s_1} \prod_{j \geq 2, s_{j-1} \neq s_j} a_{s_{j-1} s_j}) \\ &\quad -\log(\prod_{j \geq 2, s_{j-1} = s_j} a_{s_{j-1} s_j}) \\ &= C'_s + C''_s = C_s \end{aligned} \tag{6}$$

In summary, we have

$$-\log p(O, \mathbf{s}|\lambda) = Q(\mathcal{M}, \mathcal{I})$$

Hence our problem of summarizing the event sequence converts to finding an HMM and the optimal state sequence that maximizes the production probability. Moreover, once we obtain HMM and optimal state sequence, we can derive the set of models and the segmentation.

5. THE INDEPENDENT CASE: $\mathcal{M}_{IND}$

In this section, we discuss how to efficiently learn the HMM and the optimal state sequence from the event logs. Our approach includes two phases. In the first phase, we initialize an HMM, and in the second phase, we use an iterative approach to refine the model.

5.1 Initialization

We propose two approaches: a random approach and a cluster based approach, to initiate the parameters of the HMM λ.

The random approach. In the random approach, we assume there are K models in $\mathcal{M}_{ind}$, where K is an estimated value and will change in the second phase. For each model M_i, $1 \leq i \leq K$, we generate event occurrence probabilities $p_i(E_j), j = 1, \cdots, m$ randomly. All transition probabilities and initial probabilities are also initiated randomly.

The clustering based approach. The clustering based approach takes the following steps. We first partition the event sequence into multiple segments. For each segment, we learn a model. Then, we cluster the obtained models to K groups. Finally, based on the groups of the models, we initiate the parameters of the HMM.

We segment the event sequence in a bottom-up fashion. The algorithm starts by considering each single time point as a segment, $\mathcal{I} = \{I_1, \cdots, I_T\}$. For each I_i, we learn a model based on event occurrences in this segment. Then we try to merge two neighboring segments into a bigger segment. The criterion of choosing segment pairs to merge is that the merging should incur minimal increase of C_d. Assume for segments I_i and I_{i+1}, the new segment generated by merging them is I' and the learned model from I' is M'. The increasing of cost can be computed as

$$-\log(p(I'|M')) + \log(p(I_i|M_{\sigma(i)})) + \log(p(I_{i+1}|M_{\sigma(i+1)}))$$

Once a pair is chosen, we learn a model based on the new segment. The merging continues until the number of segments reaches a user-specified threshold.

In the second step, we cluster the obtained models using the K-means clustering algorithm. Since each model is represented by the probabilities of m event types, we use K-L divergence as the distance measure between two models. Let $M_i = \{p_i(E_1), \cdots, p_i(E_m)\}$, and $M_j = \{p_j(E_1), \cdots, p_j(E_m)\}$, then

$$Dis(M_i, M_j) = \sum_{l=1}^{m} p_i(E_l) \log \frac{p_i(E_l)}{p_j(E_l)} + \sum_{l=1}^{m} p_j(E_l) \log \frac{p_j(E_l)}{p_i(E_l)}$$

In the third step, for each cluster, we learn m occurrence probabilities. Based on them, we initiate the output probabilities of each state. The transition probabilities and initial probabilities are initialized according to the segment sequence.

5.2 Iterative HMM Refining

We denote the initial HMM as λ^0. We then use an iterative process to refine the initial model. Let λ^{k-1} denote the HMM obtained in round $k-1$. In round k, the iteration takes two steps. In the first step, based on λ^{k-1}, we learn the optimal state sequence $\mathbf{s}^k$. In the second step, based on $\mathbf{s}^k$, we learn a new model, λ^k. This process continues until the HMM does not change any more. We will show that in each iterative round, the production probability is no less than that in the previous round.

Learning the Optimal State Sequence. In the first step of round k, we use the Viterbi algorithm to learn the most likely sequence of hidden states. The Viterbi algorithm computes the forward probability, $\delta_t(i)$, which is the probability of the most likely sequence of hidden states up to t and $s_t = i$. It holds that

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

Assuming at time $t-1$, we already have all the forward probabilities $\delta_{t-1}(j)$ where $j = 1, \cdots, K$. Then, we compute $\delta_t(i)$, $i = 1, \cdots, K$, based on $\delta_{t-1}(j)$, as:

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

Finally, at time T, $\delta_T(i)$ is the probability of the most likely sequence of hidden states and $s_T = i$. By backtracking the largest one in $\delta_T(i), i = 1, 2, \cdots, K$, we obtain the optimal state sequence, and denote it as $\mathbf{s}^k$.

Updating HMM. In the second step, based on $\mathbf{s}^k$, we learn a new HMM, λ^k, so that $p(O, \mathbf{s}^k|\lambda^k)$ is maximized.

Recall that $p(O, \mathbf{s}^k|\lambda^{k-1})$ can be decomposed into two parts: P_1 and P_2, and P_1 is determined by the output probabilities only.

$$P_1 = \prod_{t=1}^{T} b_{s_t}(o_t)$$

This allows us to update the output probabilities to maximize P_1. It can be transformed as

$$P_1 = \prod_{i=1}^{K} p(D_i|M_i)$$

where $D_i = \{t|s_t = i\}$ and $p(D_i|M_i)$ is similar to $p(I|M_i)$ in Eq. 1. Thus, we decompose P_1 into K independent parts: $p(D_i|M_i)$, $1 \le i \le K$, and the i-th part is only related to model M_i, which means that we can update occurrence probabilities for different models independently.

Furthermore, we decompose $p(D_i|M_i)$ into m parts, each corresponding to an event type:

$$p(D_i|M_i) = \prod_{j=1}^{m} p(E_j|M_i)$$

where

$$p(E_j|M_i) = p_i(E_j)^{n(E_j, D_i)} (1 - p_i(E_j))^{n(\bar{E_j}, D_i)}$$

It means that we can update occurrence probability for each event type independently. Hence, for event E_j, its occurrence probability in model M_i is updated as:

$$p_i(E_j) = \frac{n(E_j, D_i)}{|D_i|}$$

P_2 is determined by initial probabilities and transition probabilities. To maximize P_2, we update initial probability π_i as

$$\pi_i = \frac{n(s_t = i)}{T}$$

where $n(s_t = i)$ is the number of times state i occurs in $\mathbf{s}^k$, and update transition probability a_{ij} as

$$a_{ij} = \frac{n(s_t = i, s_{t+1} = j)}{n(s_t = i)}$$

In some cases, certain state will not occur in the state sequence. At that time, we delete it from HMM. So, the number of states, K, may be variable during the refinement process.

Thus, we obtain the new model λ^k. It can be proved easily that for any HMM λ, it holds

$$p(O, \mathbf{s}^k|\lambda^k) \ge p(O, \mathbf{s}^k|\lambda)$$

Correctness of the approach.
In the first step of round k, the learned state sequence, $\mathbf{s}^k$, is the optimal one to generate event sequence based on λ^{k-1}, it holds

$$p(O, \mathbf{s}^k|\lambda^{k-1}) \ge p(O, \mathbf{s}^{k-1}|\lambda^{k-1})$$

In the second step, λ^k is the HMM which maximizes the production probability along state sequence $\mathbf{s}^k$, so it holds

$$p(O, \mathbf{s}^k|\lambda^k) \ge p(O, \mathbf{s}^k|\lambda^{k-1})$$

Combining the above two inequations, we obtain

$$p(O, \mathbf{s}^k|\lambda^k) \ge p(O, \mathbf{s}^{k-1}|\lambda^{k-1})$$

which means the probability of the optimal state sequence in each round is no less than that in the last round.

6. THE CORRELATED CASE: $\mathcal{M}_{DEP}$

The model class $\mathcal{M}_{ind}$ assumes that occurrences of different event types are independent from each other. In reality, correlations between different event types exist in most applications. They are informative to users who want to understand the behavior of a system, and they are important factors to consider if users want to reduce the encoding cost. In this section, we introduce a new model class, $\mathcal{M}_{dep}$, that exploits the correlations between event occurrences.

We illustrate the advantage of exploiting the correlations with an example. Assume we know E_1 occurs, and also assume the occurrence of E_2 depends on that of E_1 in the sense that if E_1 occurs, very likely E_2 will also occur. In this case, conditional probability $p(E_2|E_1)$ is close to 1. To describe the occurrences of E_1 and E_2, we first describe E_1 with its occurrence probability $p(E_1)$. Then we use $p(E_2|E_1)$ to describe E_2's occurrences. So, the encoding cost of E_2's occurrence is $-\log(p(E_2|E_1))$. In most cases, it is much smaller than the cost of encoding it with its occurrence probability, $-\log(E_2)$.

6.1 Definition of $\mathcal{M}_{dep}$

We use rooted trees to express correlations between different event types. In general, the occurrence of an event type can depend on any number of event types. But in this work, we assume each event type depends on only one event type. The reasons are: 1) In many applications, the majority of correlation is in the form of an event type depending on another event type, instead of on multiple other event types. Focusing on this type of correlations gives us much extra insights about the behavior of the system and much additional savings on encoding cost. 2) Describing correlations among multiple event types will lead to very complex models that are hard for users to understand.

For each model M_i in $\mathcal{M}_{dep}$, we organize the correlations between event types as a correlation tree, denoted as $CTree_i$, where each node represents an event type. The event type represented by the root node is called the *root event type*, and is denoted as E_{r_i}. Event types represented by other nodes are *non-root event types.* A directed edge from E_l to E_j indicates that occurrences of E_j depend on E_l.

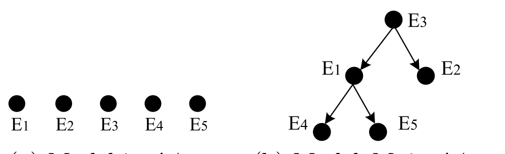
We assume the root event type does not depend on any other event types, and we represent it by the occurrence probability, $p_i(E_{r_i})$. For non-root event type E_j, the event type it depends on in model M_i is denoted as $E_{d_{ij}}$. Its occurrence is described by two conditional probabilities: $p_i(E_j|E_{d_{ij}})$ and $p_i(E_j|\bar{E}_{d_{ij}})$, where the former is the conditional probability of E_j's occurrence given $E_{d_{ij}}$ occurs, and the latter is that given $E_{d_{ij}}$ does not occur. If a certain event type E_j does not depend on any event type, it holds $p(E_j|E_l) = p(E_j)$, where E_l is any event type. Thus, using conditional probabilities will not lead to extra encoding cost. In fact, the probabilities of event occurrences in $\mathcal{M}_{ind}$ and $\mathcal{M}_{dep}$ can be represented by Eq. 7 and Eq. 8 respectively, which illustrate clearly the difference between the two model classes.

$$p_i(E_1, E_2, \cdots, E_m) = p_i(E_1)p_i(E_2)\cdots p_i(E_m) \qquad (7)$$

$$p_i(E_1, E_2, \cdots, E_m) = p_i(E_{r_i}) \prod_{j=1, j\neq r_i}^{m} p_i(E_j|E_{d_{ij}}) \qquad (8)$$

Fig 4(b) shows an example of $CTree_i$, in which $r_i = 3$. As for non-root event type, we have $d_{i4} = 1$, $d_{i5} = 1$, $d_{i1} = 3$ and $d_{i2} = 3$. Since in the model of $\mathcal{M}_{ind}$, event types are independent from each other, it can be regarded as a forest with five root nodes, as shown in Fig 4(a).

One important characteristics of the correlation tree is that it is acyclic, which can be inferred from Eq. 8. Moreover, in the process of learning HMM, the structure of the correlation tree for each model will be updated, so that it can describe the dependence better. Specifically, for each model, we need to decide which event type will be the best root event, and which is the event type a non-root event type depends on.



(a) Model in $\mathcal{M}_{ind}$ (b) Model M_i in $\mathcal{M}_{dep}$

Figure 4: Model in $\mathcal{M}_{ind}$ and $\mathcal{M}_{dep}$

6.2 The Objective Function Q

Just as in $\mathcal{M}_{ind}$, summarizing an event log with models in $\mathcal{M}_{dep}$, incurs three cost, C_m, C_d and C_s. The cost of encoding the models in $\mathcal{M}_{dep}$ is higher than that of $\mathcal{M}_{ind}$, since we need to encode both the (conditional) probabilities and the structure of the correlation tree. However, this part of cost does not increase as the size of the event log increases. So we ignore it in the objective function.

The definition of C_s is the same as that in Section 4. Now we discuss C_d, the cost of encoding event occurrences. Assume we use model M_i to encode event occurrences in segment I. In model M_i, the root event is E_{r_i}, and the event type a non-root event E_j depends on is $E_{d_{ij}}$. The probability of event occurrences in I, given M_i, is

$$p(I|M_i) = \prod_{j=1}^{m} p(E_j|M_i)$$

where if E_j is the root event, we have

$$p(E_j|M_i) = p_i(E_j)^{n(E_j, I)}(1 - p_i(E_j))^{n(\bar{E}_j, I)} \qquad (9)$$

otherwise, if it is a non-root event, we have

$$\begin{aligned} & p(E_j|M_i) \\ = \ & p_i(E_j|E_{d_{ij}})^{n(E_j, E_{d_{ij}}, I)}(1 - p_i(E_j|E_{d_{ij}}))^{n(\bar{E}_j, E_{d_{ij}}, I)} \\ & p_i(E_j|\bar{E}_{d_{ij}})^{n(E_j, \bar{E}_{d_{ij}}, I)}(1 - p_i(E_j|\bar{E}_{d_{ij}}))^{n(\bar{E}_j, \bar{E}_{d_{ij}}, I)} \end{aligned} \qquad (10)$$

where $n(E_j, E_{d_{ij}}, I)$ denotes the number of time points at which both E_j and $E_{d_{ij}}$ occur, $n(\bar{E}_j, E_{d_{ij}}, I)$ denotes the number of time points at which $E_{d_{ij}}$ occurs but $\bar{E}_j$ not, $n(E_j, \bar{E}_{d_{ij}}, I)$ and $n(\bar{E}_j, \bar{E}_{d_{ij}}, I)$ have the similar meaning.

The number of bits required to encode data in I_i comes to $-\log p(I_i|M_{\sigma(i)})$, and so the cost, C_d, is

$$C_d = \sum_{i=1}^{|\mathcal{I}|} -\log p(I_i|M_{\sigma(i)})$$

We define the objective function as:

$$Q(\mathcal{M}_{dep}, \mathcal{I}) = C_s + C_d$$

6.3 The HMM Corresponding to $\mathcal{M}_{dep}$

In the HMM corresponding to $\mathcal{M}_{dep}$, state i still represents model M_i. The initial and transition probabilities are the same as those in the HMM for $\mathcal{M}_{ind}$. Let $o =$

$(o^1, \cdots, o^m)$ denote an observation, where $o^j = 1$ mean E_j occurs, and $o^j = 1$ means it doesn't occur. The output probability of state i generating o changes to

$$b_i(o) = \prod_{j=1}^{m} q_{ij}(o)$$

There are two cases of q_{ij}. In the first case, E_j is a root event type of M_i, so q_{ij} only depends on o^j, and we have:

$$q_{ij}(o) = \begin{cases} p_i(E_j), & o^j = 1; \\ 1 - p_i(E_j), & o^j = 0. \end{cases}$$

In the second case, E_j is a non-root event type, so q_{ij} depends on both o^j and $o^{d_{ij}}$, where $E_{d_{ij}}$ is E_j's dependent event type. We have

$$q_{ij}(o) = \begin{cases} p_i(E_j|E_{d_{ij}}), & o^j = 1 \text{ and } o^{d_{ij}} = 1; \\ 1 - p_i(E_j|E_{d_{ij}}), & o^j = 0 \text{ and } o^{d_{ij}} = 1; \\ p_i(E_j|\bar{E}_{d_{ij}}), & o^j = 1 \text{ and } o^{d_{ij}} = 0; \\ 1 - p_i(E_j|\bar{E}_{d_{ij}}), & o^j = 0 \text{ and } o^{d_{ij}} = 0; \end{cases}$$

Using the same technique as in Section 4, we can establish the equivalence between the production probability and the objective function Q. So, our goal is still to find a HMM λ and the optimal state sequence $\mathbf{s}$ so that $P(O, \mathbf{s}|\lambda)$ is maximized.

6.4 Algorithm of learning HMM

We still use an iterative process to learn $\mathcal{M}_{dep}$. In the initial phase, a random approach and a cluster based approach are used to initialize the HMM. In the random approach, we first generate the correlation tree for each model randomly. Then for each edge, we randomly generate the conditional probabilities. In the cluster based, we initiate a uniform correlation tree for all the models. Moreover, KL-divergence is computed based on the conditional probabilities.

In each iterative round, the step of learning the optimal state sequence is similar to the case of $\mathcal{M}_{ind}$, except that the definition of output probabilities is different. Now we discuss the second step, in which we update HMM.

Assume in round k, we already obtain the optimal state sequence $\mathbf{s}^k$, and try to learn λ^k. First, we update the transition probabilities and initial probabilities with the way in Section 5.

The approach of updating output probabilities is more complex. The reason is that besides updating the probabilities themselves, we also update the structure of correlation tree for each model. Specifically, for each model, we learn the new root event, and dependent event type for each non-root event. The update process contains two steps. First, we update all possible probabilities; then we learn a correlation tree, which achieves higher probability than the previous one, based on the updated probabilities.

Recall that in Section 5, P_1 is decomposed into K parts, $P(D_i|M_i)$, $(1 \leq i \leq K)$, and each of which can be further decomposed into m parts, $p(E_j|M_i)$, $(1 \leq j \leq m)$. In the case of $\mathcal{M}_{dep}$, this decomposition still holds, except that the formula of $p(E_j|M_i)$ changes to Eq. 9 and Eq. 10. We update all probabilities $(1 \leq j, l \leq m)$ in model M_i as follows:

$$p_i(E_j) = \frac{n(E_j, D_i)}{|D_i|} \tag{11}$$

$$p_i(E_j|E_l) = \frac{n(E_j, E_l, D_i)}{n(E_l, D_i)} \tag{12}$$

$$p_i(E_j|\bar{E}_l) = \frac{n(E_j, \bar{E}_l, D_i)}{n(\bar{E}_l, D_i)} \tag{13}$$

so that $P(D_i|M_i)$ is maximized.

To update the correlation tree, we build the connection between $P(D_i|M_i)$ and the correlation tree $CTree_i$.

- For the root node E_{r_i}, we set its weight w_{r_i} as $p(E_{r_i}|M_i)$ (computed as Eq. 9).
- For the edge from E_j to $E_{d_{ij}}$, we set its weight $w_{d_{ij},j}$ as $p(E_j|M_i)$ (computed as Eq. 10).

Thus the product of weights of all edges and the root node is equivalent to $p(D_i|M_i)$. So our goal of updating correlation tree for M_i becomes finding a correlation tree, so that the product of its weights is maximized.

Now we introduce our approach. It is easy to see that learning the optimal correlation tree, which maximizes $p(D_i|M_i)$, is an NP problem. In this paper, we propose a greedy algorithm to learn a correlation tree $CTree_i'$, whose weight's product is larger than that of the current tree, $CTree_i$. A node list NL maintains the event types already processed, and it is empty in the beginning. The algorithm contains two steps:

Step 1. Learn the new root event type. For any event type E_j, we check whether setting it as the new root can increase the weight product. Specifically, we find the event type $E_j, j \neq r_i$, which satisfies

$$w_{r_i} \cdot w_{d_{ij},j} \leq w_j \cdot w_{j,r_i}$$

The left part of the inequality is the product of weights of E_{r_i} and E_j in previous tree $CTree_i$. The right part for the new tree, in which we switch E_j and E_{r_i}, i.e., we set E_j as the new root, and E_{r_i} as its dependent.

If there exist event types satisfying the above inequality, we choose the one that has maximal weight product as the new root event type E'_{r_i}. We also change the previous root event type E_{r_i} to a non-root event type and set the new root as the event type it depends on. If not, we keep the previous root unchanged. In both cases, the new root is added into NL.

Step 2. Learn new dependent event types. We traverse the previous tree $CTree_i$ in the post order. Assume the current event type is E_j, we check whether $w_{d_{ij},j}$ is larger than $w_{l,j}$, for any l. There are two cases:

- If it is true, E_j is still the event type $E_{d_{ij}}$ depends on. So we add E_j into NL.
- If not, we try to find another new event type for E_j. To avoid generating a circle in the correlation tree, we only check event type E_l not in NL that satisfies

  $$w_{l,j} > w_{d_{ij},j}$$

  If it is true, we choose one of them to be the new event type that E_j depends on. Finally, we add E_j to NL.

It is easy to see that the dependent tree does not contain cycles. Furthermore, since we adjust dependence only when the weights will increase, the weight product of the new tree $CTree_i'$ is no less than that of $CTree_i$. However, since the method is greedy, the process may not lead to optimal solutions.

7. EXPERIMENTAL EVALUATION

We have conducted extensive experiments on both synthetic and real life datasets. We compare the performance of our approach with the approach proposed in [10] under a variety of settings. All algorithms are implemented in Matlab 7.0. The experiments are conducted on a PC with Intel Pentium 4 2.4GHz CPU and 2GB RAM.

7.1 Algorithms for Comparison

We test the effectiveness of our approach in 4 cases: two model classes ($\mathcal{M}_{ind}$ and $\mathcal{M}_{dep}$) combining with two HMM initializers (random and clustering based).

	Model Class	HMM Initializer
RAN-IND	$\mathcal{M}_{ind}$	Random
CLU-IND	$\mathcal{M}_{ind}$	Clustering
RAN-DEP	$\mathcal{M}_{dep}$	Random
CLU-DEP	$\mathcal{M}_{dep}$	Clustering

Table 2: Our approach in 4 cases

We compare the above 4 algorithms with the approach in [10], which we name GREEDY, as it uses a greedy algorithm both in finding the optimal segmentation and in learning local models for each segment.

7.2 Experiments on Synthetic Data

First we conduct experiments on synthetic data.

Datasets.

We use the following parameters to characterize a synthetic event sequence: K, the number of models (or hidden states) that we artificially "plant" in the generated event sequence; K_{ini}, the number of models in the initial phase; m, the number of event types (the size of $\mathcal{E}$); n, the number of segments contained in the event sequence; d, the number of dependence between event types; and $l_i, 1 \leq i \leq K$, the average length of segments generated by model M_i.

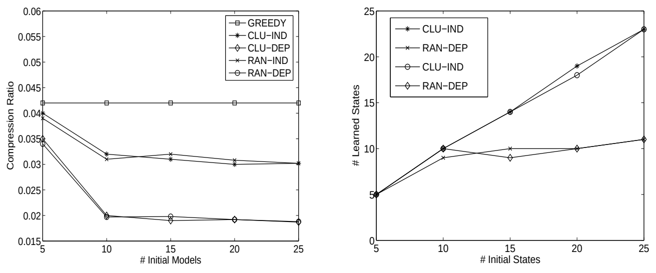
We generate K models as follows. First, we randomly generate d correlated pairs of event types for each model. Then all output probabilities in each model, which include $p(E|E')$ and $p(E|\bar{E}')$ for dependent event types and $p(E)$ for non-dependent event types, are generated randomly. All initial probabilities and transition probabilities are also generated randomly.

After the models are generated, we generate n segments of event occurrences based on models. First, we set l_i according to $-\log p_{ii}$, $1 \leq i \leq K$. Then based on initial probabilities and transition probabilities, we generate a state sequence with length n. Based on this state sequence, we generate a sequence of segments. Assume the t-th state is i, we generate event occurrences in segment I_t according to model M_i. The length of I_t is set according to Gaussian distribution with mean l_i and variance $5\% * l_i$. For non-dependent event type, the event occurrences are generated from the Gaussian distribution with output probability $p(E)$ as the mean and $10\% * p(E)$ as the variance. For event type depending on certain event type, the occurrences are generated from the Gaussian distribution with the conditional probability, $p(E|E')$ or $p(E|\bar{E}')$, as the mean and $10\% * p(E|E')$ or $10\% * p(E|\bar{E}')$ as the variance;

Compression Ratio. Compression ratio measures how well the algorithm can compress the event sequence. We adopt the formula in [10]:

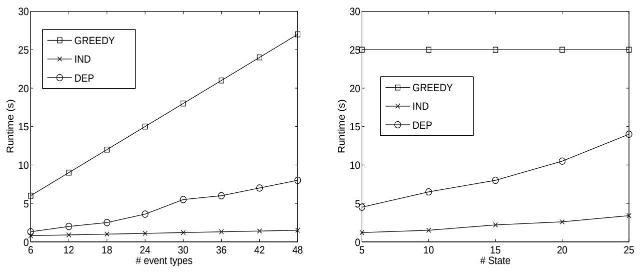
$$CR(A) = \frac{Q(A)}{Q_{unit}}$$

where A can be any of the five approaches: CLU-IND, RAN-IND, RAN-DEP, CLU-DEP, or GREEDY; $Q(A)$ is the encoding cost of event sequence with approach A. In the proposed approach, we have $Q = C_m + C_s + C_d$; while in the GREEDY approach, the cost is computed as in [10]. Q_{unit} is the measurement used as the baseline cost where each time point is considered as a segment, and in the segments, each event type is described by a distinct probability. The lower the compression ratio $CR(A)$, the better the approach A summarizes the event sequence.



(a) Compression Ratio (b) Detection of Hidden States

Figure 8: Effect of K_{ini}, $K = 10$, $n = 100$, $m = 20$, $d = 10$



(a) VS. m (b) VS. K_{ini}

Figure 9: Runtime (a) $n = 100$, $d = m/2$, $K = 10$, $K_{ini} = 10$; (b) $n = 100$, $m = 20$, $d = 10$, $K_{ini} = K$

Comparison of the Compression Ratio. In the first experiment, we compare the compression ratio of 5 approaches with varying m (number of event types). The results, as well as the settings of other parameters, are shown in Figure 5. When m increase, the compression ratio of all 5 approaches decreases. For the GREEDY algorithm, the compression ratio decreases because it clusters event types and use one probability to represent the event occurrences in each cluster, which reduces the cost of encoding models. CLU-IND and RAN-IND have lower compression ratio than GREEDY. The reason is that we use a specific probability to describe an event type, and so our model can represent the occurrences of events more accurately. As for the cost to encode the model, our approaches have much lower cost, since we only encode each model once. CLU-DEP and RAN-DEP

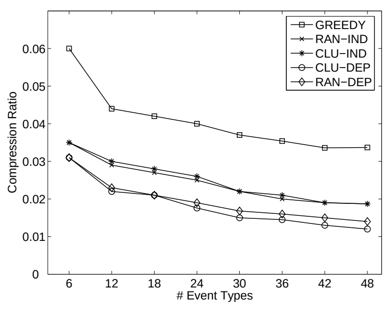


Figure 5: CR VS. m.
$n = 100,\ d = m/2,\ K = 10,\ K_{ini} = 10$

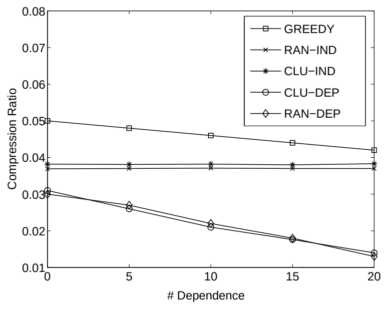


Figure 6: CR VS. d.
$n = 100,\ m = 30,\ K = 10,\ K_{ini} = 10$

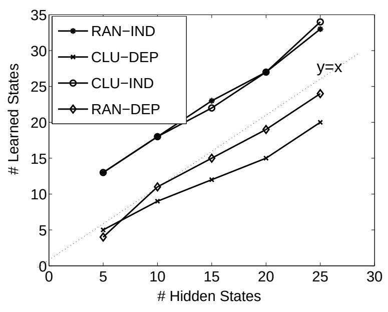


Figure 7: Detection of Hidden States.
$n = 100,\ m = 20,\ d = 10,\ K_{ini} = K + 10$

have lower compression ratio, because they exploit the correlations between event types.

In the second experiment, we test the effectiveness of exploiting correlations between event types. The compression ratio is compared when d varies. The results are shown in Figure 6. It can be seen that CLU-IND and RAN-IND have relatively fixed compression ratio, while the compression ratio decreases in the other 3 approaches. The compression ratio of CLU-IND and RAN-IND doesn't change, since they don't use event correlations to describe the event occurrences. In GREEDY algorithm, when d increases, more correlations occur between event types, and the opportunity of event types having similar occurrence probabilities increases. So the number of clusters of event types decreases, which will reduce the cost of encoding models. Compression ratio in CLU-DEP and RAN-DEP decreases most. Specifically, in GREEDY, compression ratio decreases from 0.05 to 0.042, while in CLU-DEP and RAN-DEP, it decreases from 0.03 to 0.013. The results clearly verify the effectiveness of exploiting correlations between event types.

Effect of Initial States. In general, users have no idea about the hidden states in the system. So it is preferable that the approaches can detect the true hidden states correctly. One of the symptoms is that the number of initial states, K_{ini}, can converge to the true number of hidden states automatically. So in this experiment, we test the compression ratio and number of learned states when K_{ini} varies. The results are shown in Figure 8(a) and Figure 8(b) respectively. When $K_{ini} < K$, it can be seen that all proposed approaches have similar behaviors. They cannot learn enough states, and so the compression ratio suffers. But even in this case, our approaches are still better than GREEDY, which verifies the advantage of encoding model once. When $K_{ini} \geq K$, the compression ratios in all proposed approaches stay stable when K_{ini} increase. But the capabilities of detection hidden states are quite different for cluster-based approaches and random approaches. In both RAN-IND and RAN-DEP, the numbers of states converge to the true number correctly, while in CLU-IND and CLU-DEP, the final numbers of learned states are still similar with K_{ini}. The reason is that in cluster-based approaches, the initial states can describe certain segments well, so during iteration, they are less likely to be deleted. In fact, the states learned by these two approaches are also meaningful, which can be seen from their compression ratios. In contrast, in RAN-IND and RAN-DEP, since the states are initiated randomly, some meaningless states will be deleted. They are more likely to converge to the true value K. So, the random approaches are better choices to learn the true number of states.

Detection of Hidden States. In this experiment, we test the proposed approaches' accuracy of detecting hidden states when K varies. We always set $K_{ini} = K + 10$. The results are shown in Figure 7. It is consistent with the results in last experiment. In CLU-IND and CLU-DEP, the number of learned states are close to K_{ini}. In other words, they tend to keep the number of initial states unchanged. In contrast, RAN-IND and RAN-DEP can detect K correctly.

Efficiency. In this experiment, we test the efficiency of the proposed approaches. Two factor are considered, one is the number of event types, m, and the other is number of hidden states, K. Since the round of iterations is different, it is hard to compare the proposed approaches by the total runtime. So we compare the runtime of the proposed approaches in each iterative round. The results are shown in Figure 9(a) and (b) respectively.

It can be seen that when m increases, runtime in GREEDY increases more rapidly than our approach. The reason is that at each time point, GREEDY needs to cluster the event types, while our approaches only need to compute the probabilities of event occurrences, which is much more efficient. In CLU-DEP and RAN-DEP, we need to update correlation tree at each iteration, which causes them slower than CLU-IND and CLU-DEP. Number of hidden states, K, has more influence on efficiency than m, since the time complexity of each round in our approaches is $O(K^2)$. As for the round of iteration, it is slightly different for cluster-based approaches and random approaches. In general, the former iterates 3-4 rounds, and the latter iterates 5-6 rounds. Hence, the total runtime of our approach is comparable to the GREEDY algorithm.

7.3 Experiments on Real Data

In this section, we further test the performance of our approach in a real-life scenario. By using event logs managed by Windows XP, we show that our approaches not only improve the compression ratio, but also find some meaningful contexts hidden in event log.

The real dataset consists of the system log produced by Windows XP Event Viewer. It contains 56 different event types ($m = 56$). The log data spans a period from March 2009 to July 2009. It includes 11365 event occurrences. We extract 4 datasets from the log file. The first one, denoted by "Original", includes 6,200 timestamps at which at least one event occurs. Each timestamp corresponds to one second. To test our approach to mine the correlations between event types, we build 3 other datasets based on the original data. We split the whole span into sequence of intervals with fixed duration. In the first dataset, denoted by MIN, the duration is one minute. It contains 120,829 minutes ($N = 120,829$). In the second dataset, denoted by $10MIN$, the duration is 10 minute ($N = 12,083$). In the third dataset, denoted by $HOUR$, the duration is one hour ($N = 2,000$). Tables 3 shows the compression ratio in all the four datasets, which is compared to GREEDY. It can be seen our approaches have apparently lower compression ratio compared to GREEDY algorithm, especially in 10MIN and HOUR dataset.

Data	Original	MIN	10MIN	HOUR
GREEDY	1	1	1	1
CLU-IND	0.912	0.738	0.594	0.732
RAN-IND	0.943	0.762	0.621	0.712
RAN-DEP	0.826	0.619	0.512	0.528
CLU-DEP	0.791	0.595	0.486	0.471

Table 3: Compression Ratio On Log Data

Moreover, we found some meaningful states. For example, in MIN dataset, state 4 corresponds to periods of starting up the computer, in which events 6005 and 6009 always occur together. Its duration is about 5 minutes. State 22 corresponds to periods of surfing the Internet, in which events 2504 and 4021 always occur frequently. Its duration is always longer than 100 minutes. Other states include shutdown, windows update $\cdots$. The results definitely verify that the proposed model can find the hidden states of the system. In the contrast, GREEDY only segments the event sequence into 3 intervals. It is hard to obtain the meaningful contexts from its result.

8. CONCLUSIONS

In this paper, we learn the dynamics of a system from its event log. Instead of segmenting event sequences and describing each segment with a local model, we learn a set of models each representing a specific state of the system. Furthermore, we learn the relationships between different states. This effectively produces an HMM that describe the system. We show the task of summarizing the event data based on the minimal description length principle can be achieved through learning an HMM from the event data. In this paper, we propose an iterative approach to learn an HMM. We also exploit the correlations between different event types to further improve the summary. We test the proposed technique on both synthetic dataset and real dataset. The experimental results show that our approach not only summarizes event sequence well, but also reveals hidden states in the system that generates the events, giving us a better understanding of the system.

9. REFERENCES

[1] Rakesh Agrawal and Ramajrishnan Srikant. Mining sequential patterns. In *ICDE*, 1995.

[2] Gemma Casas-Garriga. Discovering unbounded episodes in sequential data. In *PKDD*, 2003.

[3] Shixi Chen, Haixun Wang, and Shuigeng Zhou. Concept clustering of evolving data. In *ICDE*, 2009.

[4] Shixi Chen, Haixun Wang, Shuigeng Zhou, and Philip S. Yu. Stop chasing trends: Discovering high order models in evolving data. In *ICDE*, 2008.

[5] Yannis Manolopoulos Christos Faloutsos, M. Ranganathan. Fast subsequence matching in time-series databases. In *SIGMOD*, 1994.

[6] Sudipto Guha, Nick Koudas, and Kyuseok Shim. Data-streams and histograms. In *STOC*, 2001.

[7] Panagiotis Karras, Dimitris Sacharidis, and Nikos Mamoulis. Exploiting duality in summarization with deterministic guarantees. In *SIGKDD*, 2007.

[8] Eamonn Keogh, Kaushik Chakrabarti, Sharad Mehrotra, and Michael Pazzani. Locally adaptive dimensionality reduction for indexing large time series databases. In *SIGMOD*, 2001.

[9] Eamonn Keogh, Selina Chu, David Hart, and Michael Pazzani. An online algorithm for segmenting time series. In *ICDM*, 2001.

[10] Jerry Kiernan and Evimaria Terzi. Constructing comprehensive summaries of large event sequences. In *SIGKDD*, 2008.

[11] Mikko Koivisto, Markus Perola, T. Varilo, W. Hennah, J. Ekelund, Margus Lukk L. Peltonen, Esko Ukkonen, and Heikki Mannila. An mdl method for finding haplotype blocks and for estimating the strength of haplotype block boundaries. In *Pacific Symposium on Biocomputing*, 2003.

[12] Srivatsan Laxman, Vikram Tankasali, and Ryen White. Stream prediction using a generative model based on frequent episodes in event sequences. In *SIGKDD*, 2008.

[13] Srivatsan Laxman and P. S. Sastryand K. P. Unnikrishnan. Discovering frequent episodes and learning hidden markov models: A formal connection. *IEEE Transactions on Knowledge and Data Engineering*, 17(11):1505–1517, 2005.

[14] Heikki Mannila and Marko Salmenkivi. Finding simple intensity descriptions from event sequence data. In *SIGKDD*, 2001.

[15] Heikki Mannila, Hannu Toivonen, and Inkeri Verkamo. Discovery of frequent episodes in event sequences. In *DMKD*, 1997.

[16] Nicolas Meger and Christophe Rigotti. Constraint-based mining of episode rules and optimal window sizes. In *PKDD*, 2004.

[17] Kin pong Chan and Ada Wai chee Fu. Efficient time series matching by wavelets. In *ICDE*, 1999.

[18] J. Rissanen. Modeling by shortest data description. *Automatica*, 14:465–471, 1978.

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

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

[21] Ying Yang, Xindong Wu, and Xingquan Zhu. Combining proactive and reactive predictions for data streams. In *SIGKDD*, 2005.