Mining Temporal Lag from Fluctuating Events for Correlation and Root Cause Analysis

Chunqiu Zeng, Liang Tang and Tao Li
School of Computer Science
Florida International University
Miami, FL, USA
Email:{czeng001,ltang002, taoli}@cs.fiu.edu

Larisa Shwartz
Operational Innovations
IBM T.J. Watson Research Center
Yorktown Heights, NY, USA
Email: lshwart@us.ibm.com

Genady Ya. Grabarnik
Dept. of Math & Computer Science
St. John's University
Queens, NY, USA
Email:grabarng@stjohns.edu

***Abstract*—The importance of mining time lags of hidden temporal dependencies from sequential data is highlighted in many domains including system management, stock market analysis, climate monitoring, and more. Mining time lags of temporal dependencies provides useful insights into understanding the sequential data and predicting its evolving trend. Traditional methods mainly utilize the predefined time window to analyze the sequential items or employ statistic techniques to identify the temporal dependencies from the sequential data. However, it is a challenging task for existing methods to find time lag of temporal dependencies in the real world, where time lags are fluctuating, noisy, and tend to be interleaved with each other. This paper introduces a parametric model to describe noisy time lags. Then an efficient expectation maximization approach is proposed to find the time lag with maximum likelihood. This paper also contributes an approximation method for learning time lag to improve the scalability without incurring significant loss of accuracy. Extensive experiments on both synthetic and real data sets are conducted to demonstrate the effectiveness and efficiency of proposed methods.**

I. Introduction

More than ever, businesses heavily rely on IT service delivery to meet their current and frequently changing business requirements. In their quest to maximize customer satisfaction, Service Providers seek to employ business intelligent solutions that provide deep analytical and automation capabilities for optimizing problem detection, determination and resolution[2],[19]. Detection is usually provided by system monitoring, an automated system that provides an effective and reliable means of ensuring that degradation of the vital signs is flagged as a problem candidate (monitoring event) and sent to the service delivery teams as an incident ticket. When correlated, monitoring events, discrete in nature, could also provide effective and reliable means for a problem determination. There has been a great deal of effort spent on developing methodologies for event correlation and, subsequently, root cause analysis in IT Service Management. One fruitful line of research has involved the development of techniques for traversing dependencies graphs of a system or application configuration. Although these methods have been successful for reasoning about failures, they have had limited impact because of the overhead associated with constructing such graphs and keeping them up-to-date. Another approach has focused on mining temporal properties of events. The essence of this approach is to work as far as possible with temporal data from event management systems rather than relying on external data.

Time lag, one of the key features in temporal dependencies, plays an important role in discovering evolving trends of the coming events and predicting the future behavior of its corresponding system. For instance, a network adapter problem typically leads to disconnecting an instant messaging tool running on that machine after several failed retries for communication. In this scenario, the event of *network interface down* leads to a *disconnect event* of the instant messaging tool after several failed re-tries with a given time lag. The temporal dependencies among events are characterized by the time lags. Time lags provide temporal information for building fault-error-failure chain[3] which is useful for root cause analysis. In addition, events triggered by a single issue can be correlated given the appropriate time lags. Merging those correlated events in one ticket reduces the effort of an administrator for problem diagnosis and incident resolution. Thus, the discovery of the time lag is a very important task during temporal dependency mining.

The situation in real-world systems becomes complicated due to the limitation of sequential data collecting methods and the inherent complexity of the systems. However, events detected by monitoring systems are typically studied with an assumption that the time lag between correlated events is constant and fluctuations are limited and can be ignored [18]. Although such an approach is undoubtedly applicable to a wide range of systems, fluctuations can render the deterministic classical picture qualitatively incorrect, especially when correlating events are limited. Taking the randomness of the time lag into account makes the detection of the hidden time lags between interleaved events a challenging task.

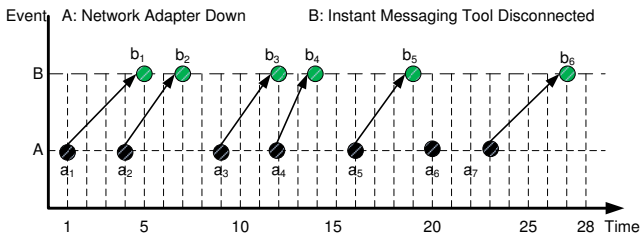

Fig. 1: The temporal dependencies between A and B are denoted as direct edges.

First, the fluctuating interleaved temporal dependencies pose a challenging problem when attempting to discover the correct hidden time lag between two events. For example, two events A and B, corresponding to the events *network interface down* and *disconnect event* of an Instant Messaging Tool, respectively, are shown in Fig. 1. Both A and B occur with multiple instances in the sequential data set. The i^{th} instance of A and the j^{th} instance of B are associated with their timestamp a_i and b_j. Because the true temporal dependencies are interleaved with each other, it is difficult to determine which b_j is implied by a given a_i. The different mapping relationships between a_i and b_j lead to varying time lags. In this example, a_1 can be mapped to any b_j. Therefore, the time lag ranges from $b_1 - a_1$ to $b_6 - a_1$ time units. It is infeasible to find the time lag with exhaustive search from large scale sequential event data.

Second, due to the clocks out of sync and the limitations of the data collecting method, the time lags presented in the sequential data may oscillate with noise. In Fig. 1, a_6 does not correspond to any instance of event B for several possible reasons: (1) its corresponding instance of B is missing from the sequential data set; (2) the Network Adapter returns to normality in such a short time that there is no need to eject an instance of B since the Instant Messaging Tool successfully continues to work after only one try; and (3) even though the correct instance mapping relation between A and B are provided, time lags are still observed with different values due to recording errors brought about by system monitoring.

In summary, the above difficulties pose a big challenge for time lag mining. To address those difficulties in this paper, we first apply a parametric model to formulate the randomness of time lags for temporal dependencies between events. Next, an EM-based approach is proposed to discover the maximal likelihood model of time lags for the temporal dependencies. Finally, we come up with an efficient approximation method which allows the discovery of time lags from massive events without much loss of accuracy.

The remainder of the paper is organized as follows: In Section 2, related work is discussed. We formulate the problem for finding time lag with a parametric model in Section 3. In Section 4, an EM-based solution is proposed to solve the formulated problem. Extensive experiments are conducted in Section 5. Section 6 provides a conclusion of our work.

II. Related Work

Mining temporal dependencies among events acts an essential part in enterprise system management and the discovered temporal dependencies have been successfully applied to tuning up monitoring system configurations [12], [14].

A precursor of the temporal dependency discovery was mining of frequent itemsets from the transactional data. Typical algorithms here are GSP [16], FreeSpan [6], PrefixSpan [13], and SPAM [4]. However, this paper focuses on discovering the temporal dependencies from sequential data, where no information is given to show what items belong to the same transaction and only the time stamps of items can be utilized as a basis for dependency instead. In this paper, items with time stamps and events are used interchangeably.

Based on the fact that potentially related items tend to happen within a certain time interval, some previous work of temporal mining focuses on frequent itemsets given a predefined time window [7]. However, it's difficult to determine a proper window size. A fixed time window fails to discover the temporal relationship longer than the window size. Simply setting time window size to some large number makes the problem intractable, due to the exponential complexity of finding frequent itemsets on the maximal number of items per transaction.

A temporal relationship is typically represented as a pair of items within a specific time lag. We denote it as $A \rightarrow_{[t_1,t_2]} B$. It means that an event B will happen within time interval $[t_1, t_2]$ after an event A occurs. A lot of work was devoted to finding such temporal dependencies characterized with time lag [11], [9], [10], [18]. They applied statistics to judge whether a given lag interval for two dependent items is meaningful or it is just caused by randomness. In this paper we consider the more realistic condition that the time lag L is random. We extract the probability distribution of L along with the dependent items. The lag probability distribution allows for more insights and flexibility than just a fixed interval.

Moreover, it is a challenging task in previous work to check a large number of possible time lags due to the complexity of combinatorial explosion, though an optimized algorithm is proposed with pruning techniques in [18]. In this paper, we propose an EM-based approximation method to efficiently learn the distribution of time lag in temporal dependency discovery.

III. Problem Formulation

A. Problem Description

In temporal pattern mining, the input data is a sequence of events. Given the event space $\mathbf{\Omega}$ of all possible events, an event sequence $\mathbf{S}$ is defined as ordered finite sequence $\mathbf{S} =< e_1, e_2, ..., e_i, ..., e_k >$, where e_i is an instance of an event. We consider temporal events, i.e., each e_i is a tuple $e_i = (E_i, t_i)$ of event $E_i \in \mathbf{\Omega}$ and a timestamp t_i of event occurrence.

Let A and B be two types of events from the event space $\mathbf{\Omega}$. Focusing on a specific event A, we define $\mathbf{S_A} =< (A, a_1), ..., (A, a_m) >$ to be a subsequence from $\mathbf{S}$, where only the instances of A are kept and a_i is the timestamp of i^{th} event A. Since all the instances happening in the sequence $\mathbf{S_A}$ belong to the same type of event A, $\mathbf{S_A}$ can be simply denoted as a sequence of timestamps, i.e., $\mathbf{S_A} =< a_1, ..., a_m >$. Similarly, $\mathbf{S_B}$ is denoted as $\mathbf{S_B} =< b_1, ..., b_n >$. Discovering the temporal dependency between A and B is equivalent to finding the temporal relation between $\mathbf{S_A}$ and $\mathbf{S_B}$.

Specifically, if the j^{th} instance of event B is associated with the i^{th} instance of event A after a time lag $(\mu+\epsilon)$, it indicates

$$b_j = a_i + \mu + \epsilon, \tag{1}$$

where b_j and a_i are the timestamps of two instances of B and A respectively, μ is the true time lag to describe the temporal

relationship between A and B, and ϵ is a random variable used to represent the noise during data collection. Because of the noise, the observed time lag between a_i and b_j is not constant. Since μ is a constant, the lag $L = \mu + \epsilon$ is a random variable.

Definition 1: Recall from section II that the temporal dependency between A and B is defined as $A \rightarrow_L B$, which means that the occurrence of A is followed by the occurrence of B with a time lag L. Here L is a random variable.

In order to discover the temporal dependency rule $A \rightarrow_L B$, we need to learn the distribution of random variable L.

We assume that the distribution of L is determined by the parameters $\mathbf{\Theta}$, which is independent from the occurrence of A. The occurrence of an event B is defined by the time lag L and the occurrence of A. Thus, the problem is equivalent to learning the parameter $\mathbf{\Theta}$ for the distribution of L. The intuitive idea to solve this problem is to find maximal likelihood parameter $\mathbf{\Theta}$ given both sequences S_A and S_B. It is expressed formally by the following Equation (2).

$$\hat{\mathbf{\Theta}} = \arg\max_{\mathbf{\Theta}} P(\mathbf{\Theta}|\mathbf{S_A}, \mathbf{S_B}). \tag{2}$$

The value of $P(\mathbf{\Theta}|\mathbf{S_A}, \mathbf{S_B})$ in Equation (3) is found using the Bayes Theory.

$$P(\mathbf{\Theta}|\mathbf{S_A}, \mathbf{S_B}) = \frac{P(\mathbf{S_B}|\mathbf{S_A}, \mathbf{\Theta}) \times P(\mathbf{\Theta}) \times P(\mathbf{S_A})}{P(\mathbf{S_A}, \mathbf{S_B})}. \tag{3}$$

Applying ln to both sides of Equation (3), we get:

$$\begin{aligned} \ln P(\mathbf{\Theta}|\mathbf{S_A}, \mathbf{S_B}) = \ln P(\mathbf{S_B}|\mathbf{S_A}, \mathbf{\Theta}) + \ln P(\mathbf{\Theta}) \\ + \ln P(\mathbf{S_A}) - \ln P(\mathbf{S_A}, \mathbf{S_B}). \end{aligned} \tag{4}$$

In (4), only $\ln P(\mathbf{S_B}|\mathbf{S_A}, \mathbf{\Theta})$ and $\ln P(\mathbf{\Theta})$ are related to $\mathbf{\Theta}$. We assume that a large number of small factors contribute to distribution of L, i.e., it is uniformly distributed. As a result, the problem is reduced to maximizing the likelihood defined by

$$\hat{\mathbf{\Theta}} = \arg\max_{\mathbf{\Theta}} \ln P(\mathbf{S_B}|\mathbf{S_A}, \mathbf{\Theta}). \tag{5}$$

Therefore, the temporal dependency $A \rightarrow_L B$ can be found by solving Equation (5).

B. Computing Log-likelihood

To solve Equation (5) we need to compute the log-likelihood $\ln P(\mathbf{S_B}|\mathbf{S_A}, \mathbf{\Theta})$.

We assume that timestamps b_j in $\mathbf{S_B}$ are mutually independent given the sequence $\mathbf{S_A}$ and value of parameters $\mathbf{\Theta}$ if event B is caused by A. Therefore,

$$P(\mathbf{S_B}|\mathbf{S_A}, \mathbf{\Theta}) = \prod_{j=1}^{n} P(b_j|\mathbf{S_A}, \mathbf{\Theta}). \tag{6}$$

Given the sequence of timestamps $\mathbf{S_A}$ of event A, the instance of event B occurring at b_j is identified by possible instances of A happening at a specific timestamp a_i in the sequence $\mathbf{S_A}$ as shown in Fig. 2. In order to model the relation between a_i and b_j, we introduce a latent variable z_{ij} defined as follows.

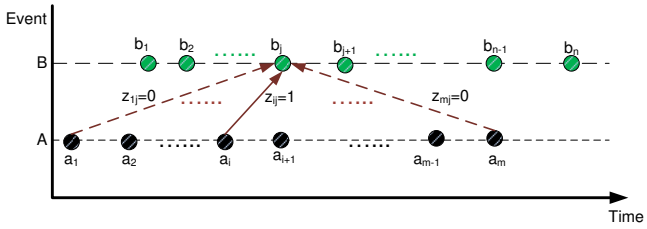

Fig. 2: The j^{th} event B occurring at b_j can be implied by any event A. Variable $z_{ij} = 1$ if the j^{th} event B is associated with i^{th} event A, and 0 otherwise.

$$z_{ij} = \begin{cases} 1, & \textit{the } i^{th} \textit{ event } A \textit{ implies the } j^{th} \textit{ event } B; \\ 0, & \textit{otherwise.} \end{cases} \tag{7}$$

Thus, given the sequence $\mathbf{S_A}$, each b_j is associated with a binary vector $\mathbf{z_{\bullet j}}$, where each component is either 1 or 0 and only one component of $\mathbf{z_{\bullet j}}$ is 1. For instance in Fig. 2, only the i^{th} component z_{ij} is 1 since a_i implies b_j, and the remaining components are 0s. We apply the latent matrix $\mathbf{Z} = \{z_{ij}\}_{n \times m}$ to denote the relation mapping between two sequences $\mathbf{S_A}$ and $\mathbf{S_B}$.

Using $\mathbf{Z}$ we obtain the following equations:

$$P(b_j|\mathbf{z_{\bullet j}}, \mathbf{S_A}, \mathbf{\Theta}) = \prod_{i=1}^{m} P(b_j|a_i, \mathbf{\Theta})^{z_{ij}}. \tag{8}$$

$$P(\mathbf{z_{\bullet j}}) = \prod_{i=1}^{m} P(z_{ij} = 1)^{z_{ij}}. \tag{9}$$

Combining Equations (8) and (9), we rewrite $P(b_j|\mathbf{S_A}, \mathbf{\Theta})$ as follows:

$$P(b_j, \mathbf{z_{\bullet j}}|\mathbf{S_A}, \mathbf{\Theta}) = \prod_{i=1}^{m} (P(b_j|a_i, \mathbf{\Theta}) \times P(z_{ij} = 1))^{z_{ij}}. \tag{10}$$

Furthermore, the joint probability $P(b_j|\mathbf{S_A}, \mathbf{\Theta})$ in Equation (10), is described by Proposition 1.

Proposition 1 (marginal probability): Given $\mathbf{S_A}$ and $\mathbf{\Theta}$, the marginal probability of b_j is as follows.

$$P(b_j|\mathbf{S_A}, \mathbf{\Theta}) = \sum_{i=1}^{m} P(z_{ij} = 1) \times P(b_j|a_i, \mathbf{\Theta}). \tag{11}$$

The proof of the Proposition is given in Appendix A-D

Based on Equation (5),(6) and (11), the log-likelihood is:

$$\ln P(\mathbf{S_B}|\mathbf{S_A}, \mathbf{\Theta}) = \sum_{j=1}^{n} \ln \sum_{i=1}^{m} P(z_{ij} = 1) \times P(b_j|a_i, \mathbf{\Theta}). \tag{12}$$

According to Equation (12), the evaluation of log-likelihood relies on the description of $P(b_j|a_i, \mathbf{\Theta})$. The explicit form of $P(b_j|a_i, \mathbf{\Theta})$ expressed in terms of time lag model is presented in the following section.

C. Modeling Time Lag

According to the discussion regarding Equation (1), the time lag L is a random variable that is the sum of the true time lag μ and the noise ϵ. The noise contributed to the true lag

is as a result of diverse factors, such as missing records, incorrect values, recording delay and so forth that happen during log collecting. In light of the Central Limit Theorem, we assume that the noise ϵ follows the normal distribution with zero-mean value, since we can always move the mean of the distribution to the constant μ. Let σ^2 be the variance of the lags distribution. Then,

$$\epsilon \sim \mathcal{N}(0, \sigma^2). \tag{13}$$

Since $L = \mu + \epsilon$ where μ is a constant, the distribution of L can be expressed as

$$L \sim \mathcal{N}(\mu, \sigma^2). \tag{14}$$

Parameters $\mathbf{\Theta}$ determines the distribution of L. Based on the model of L described in Equation (14), apparently $\mathbf{\Theta} = (\mu, \sigma^2)$. Thus, the discovery of time lag L is reduced to learning the parameters μ and σ^2.

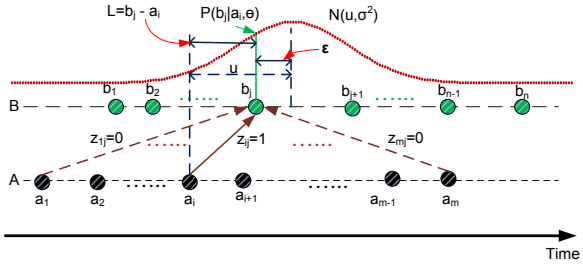

Fig. 3: $A \rightarrow_L B$, where $L \sim \mathcal{N}(\mu, \sigma^2)$. An event A that occurred at time a_i is associated to an event B that occured at b_j with probability $\mathcal{N}(b_j - a_i|\mu, \sigma^2)$. Here μ is the expected time lag of an occurrence of B after a_i.

Assume that the event A is followed by the event B with a time lag L, here $L \sim \mathcal{N}(\mu, \sigma^2)$. Specifically, as shown in Fig. 3, the i^{th} event A is associated to the j^{th} event B where the time lag $(b_j - a_i)$ between the two events is a random variable L distributed as $\mathcal{N}(b_j - a_i|\mu, \sigma^2)$. Thus,

$$\begin{aligned} P(b_j|a_i, \mathbf{\Theta}) &= P(b_j|a_i, \mu, \sigma^2) \\ &= \mathcal{N}(b_j - a_i|\mu, \sigma^2). \end{aligned} \tag{15}$$

Hence, by replacing $P(b_j|a_i, \mathbf{\Theta})$ based on Equation (15), the log-likelihood in equation (12) is expressed as:

$$\ln P(\mathbf{S_B}|\mathbf{S_A}, \mathbf{\Theta}) = \sum_{j=1}^{n} \ln \sum_{i=1}^{m} P(z_{ij} = 1) \times N(b_j - a_i|\mu, \sigma^2). \tag{16}$$

Here $P(z_{ij} = 1)$ denotes the probability that the j^{th} event B is implied by the i^{th} event A. Assume that there are m events A, so we assume that $\sum_{i=1}^{m} P(z_{ij} = 1) = 1$. To simplify the description, let $\pi_{ij} = P(z_{ij} = 1)$.

Based on the expression of log-likelihood in Equation (16), the Equation (5) is equivalent to the following

$$\begin{aligned} &(\hat{\mu}, \hat{\sigma}^2) = \arg\max_{\mu, \sigma^2} \sum_{j=1}^{n} \ln \sum_{i=1}^{m} \pi_{ij} \times \mathcal{N}(b_j - a_i|\mu, \sigma^2). \\ &\textbf{s.t.} \sum_{i=1}^{m} \pi_{ij} = 1 \end{aligned} \tag{17}$$

We describe algorithms to maximize the log-likelihood of parameters μ and σ^2 in the following section.

IV. Algorithm and Solution

A. Maximize Log-likelihood

Equation (17) is an optimization problem. Gradient ascent method is supposed to be used to solve it. However, this method is not applicable here since we cannot directly derive the closed-form partial derivatives with respect to the unknown parameters μ and σ^2. The problem described by Equation (17) is a typical mixture model. It can be solved by using iterative expectation maximization i.e., EM-based method [5].

Given $\mathbf{S_A}$ and $\mathbf{\Theta}$, by the Equation (10), the expectation of $\ln P(\mathbf{S_B}, \mathbf{Z}|\mathbf{S_A}, \mathbf{\Theta})$ with respect to $P(z_{ij}|\mathbf{S_B}, \mathbf{S_A}, \mathbf{\Theta'})$ is as follows:

$$\begin{aligned} &E(\ln P(\mathbf{S_B}, \mathbf{Z}|\mathbf{S_A}, \mathbf{\Theta})) = \\ &\sum_{j=1}^{n} \sum_{i=1}^{m} E(z_{ij}|\mathbf{S_B}, \mathbf{S_A}, \mathbf{\Theta'}) \times (\ln \pi_{ij} + \ln \mathcal{N}(b_j - a_i|\mu, \sigma^2)). \end{aligned} \tag{18}$$

where $\mathbf{\Theta'}$ is the parameter estimated on the previous iteration.

Since z_{ij} is an indicator variable, $E(z_{ij}|\mathbf{S_B}, \mathbf{S_A}, \mathbf{\Theta'}) = P(z_{ij} = 1|\mathbf{S_B}, \mathbf{S_A}, \mathbf{\Theta'})$. Let $r_{ij} = E(z_{ij}|\mathbf{S_B}, \mathbf{S_A}, \mathbf{\Theta'})$. Then,

$$r_{ij} = \frac{\pi'_{ij} \times \mathcal{N}(b_j - a_i|\mu', \sigma'^2)}{\sum_i^m \pi'_{ij} \times \mathcal{N}(b_j - a_i|\mu', \sigma'^2)}. \tag{19}$$

The new parameters π_{ij} as well as μ and σ^2 can be learned by maximizing $E(\ln P(\mathbf{S_B}, \mathbf{Z}|\mathbf{S_A}, \mathbf{\Theta}))$.

$$\mu = \frac{1}{n} \sum_{j=1}^{n} \sum_{i=1}^{m} r_{ij}(b_j - a_i), \tag{20}$$

$$\sigma^2 = \frac{1}{n} \sum_{j=1}^{n} \sum_{i=1}^{m} r_{ij}(b_j - a_i - \mu)^2. \tag{21}$$

$$\pi_{ij} = r_{ij}. \tag{22}$$

Based on Equation (22), Equation (19) is equivalent to the following:

$$r_{ij} = \frac{r'_{ij} \times \mathcal{N}(b_j - a_i|\mu', \sigma'^2)}{\sum_i^m r'_{ij} \times \mathcal{N}(b_j - a_i|\mu', \sigma'^2)}. \tag{23}$$

To find maximum likelihood estimates of parameters, we use EM-based algorithm $lagEM$ (details described in Appendix A-A). The time cost of Algorithm $lagEM$ is $O(rmn)$, where m and n are the number of events A and B, respectively, and r is the number of iterations needed for parameters to stabilize. As the time span of event sequence grows, more events will be collected. Since m and n are the counts of two types of events, it is reasonable to assume that m and n have the same order of magnitude. Therefore, the time cost of Algorithm $lagEM$ is a quadratic function of events count.

Observation 1: During each iteration of Algorithm $lagEM$, the probability r_{ij} describing the likelihood that the j^{th} event B is implied by the i^{th} event A, becomes smaller

when the deviation of $b_j - a_i$ from the estimated time lag μ increases.

Thus, as $|b_j - a_i - \mu|$ becomes larger, r_{ij} approaches 0. Further, if r_{ij} is small enough, the contribution by b_j and a_i to estimate the new parameters μ and σ^2 according to Equation (20) and (21) is negligible. As a matter of fact, the time span of the sequence of events is very long. Hence, most of r_{ij} are small. Therefore, we can estimate new parameters μ and σ^2 without significant loss of accuracy by ignoring those $r_{ij}(b_j - a_i)$ and $r_{ij}(b_j - a_i - \mu)$ with small r_{ij} in both Equation (20) and (21). During each iteration of Algorithm $lagEM$, given b_j, we can boost Algorithm $lagEM$ by not summing up all the m components for parameters estimation.

Given b_j, let ϵ_j be the sum of the probabilities r_{ij} whose component is neglected during the iteration. That is, $\epsilon_j = \sum_{\{i|a_i \ is \ neglected\}} r_{ij}$. Let ϵ be the largest one among all the ϵ_j, i.e., $\epsilon = \max_{1\leq j\leq n}\{\epsilon_j\}$. Let μ_δ and σ_δ^2 be neglected parts in the estimate μ and σ^2 during each iteration. Formally, we get,

$$\mu_\delta = \frac{1}{n}\sum_{j=1}^{n} \sum_{\{i|a_i \ is \ neglected\}} r_{ij}(b_j - a_i),$$

$$\sigma_\delta^2 = \frac{1}{n}\sum_{j=1}^{n} \sum_{\{i|a_i \ is \ neglected\}} r_{ij}(b_j - a_i)^2.$$

The following lemma allows to bound the neglected part μ_δ and σ_δ^2.

Lemma 2: Let $\bar{b}$ be the mean of all the timestamps of event B, i.e. $\bar{b} = \frac{1}{n}\sum_{j=1}^{n} b_j$. Let $\bar{b^2}$ be the second moment of the timestamps of event B, i.e., $\bar{b^2} = \frac{1}{n}\sum_{j=1}^{n} b_j^2$. Then we get:

$$\mu_\delta \in [\epsilon(\bar{b} - a_m), \epsilon(\bar{b} - a_1)]. \tag{24}$$

Let $\phi = \max\{\bar{b^2} - 2\bar{b}a_1 + a_1^2, \bar{b^2} - 2\bar{b}a_1 + a_m^2\}$, then

$$\sigma_\delta^2 \in [0, \epsilon\phi]. \tag{25}$$

The proof of Lemma 2 is provided in the Appendix A-D.

Lemma 2 can tell, if the ϵ is small enough, $|\mu_\delta|$ and σ_δ^2 approach 0 and the parameters μ and σ^2 are more closed to the ones without ignoring components.

Given a timestamp b_j, there are m possible corresponding timestamps of event A. Our problem is how to choose a subset C_j of timestamps of event A to estimate the parameters during each iteration. To guarantee that the probability of the neglected part is less than ϵ, the probability for the subset C_j should be greater than $1 - \epsilon$. In order to optimize the time complexity, our goal is to minimize the size of C_j. It can be solved efficiently by applying a greedy algorithm, which adds a_i to C_j with its r_{ij} in decreasing order until summation of r_{ij} is greater than $1 - \epsilon$.

Based on Observation 1 and the fact that all the timestamps of event A are in increasing order, the index i for timestamps of event A in C_j should be consecutive. Given b_j, the minimum and maximum indexes of a_i in C_j can be found by Algorithm $greedyBound$ listed in Appendix A-B.

The time cost of $greedyBound$ is $O(\log m + K)$ where $K = |C_j|$ and m is the number of events A.

Based on Lemma 2 and Algorithm $greedyBound$, we propose an approximation algorithm $appLagEM$. The detail of Algorithm $appLagEM$ is given in Appendix A-C.

The total time cost of Algorithm $appLagEM$ is $O(rn(\log m + K))$ where r is the number of iterations, and K is the average size of all C_j. Typically, in the event sequence, $K << n$ and $\log m << n$. Therefore, the time cost of algorithm $appLagEM$ is closed to a linear function of n in each iteration.

V. Experiments

A. Setup

The performance of proposed algorithms is evaluated by using both synthetic and real event data. The importance of an experiment conducted over synthetic data lies in the fact that the ground truth can be provided in advance. To generate synthetic data, we can fix time lag between dependent events and add noise into synthetic data. The empirical study over the synthetic data allows us to demonstrate the effectiveness and efficiency of proposed algorithms.

The experiment over the real data collected from real production environments is to show that temporal dependencies with time lags can be discovered by running our proposed algorithm. Detailed analysis of discovered temporal dependencies allows us to demonstrate the effectiveness and usefulness of our algorithm in practice.

All algorithms are implemented using Java 1.7. All experiments are conducted on the experimental environment running Linux 2.6.32. The computer is equipped with Intel(R) Xeon(R) CPU with 24 cores running at speed of 2.50GHZ. The total volume of memory is 158G.

B. Synthetic Data

1) Synthetic data generation: In this part we describe experiments conducted on six synthetic data sets. The synthetic data generation is defined by the parameters shown in Table II.

TABLE II: Parameters for synthetic data generation

Name	Description
β_{min}	Describes the minimum value for choosing the average inter-arrival time β.
β_{max}	Describes the maximum value for choosing the average inter-arrival time β.
N	The number of events in the synthetic event sequence.
μ_{min}	Describes the minimum value for the true time lag μ.
μ_{max}	Describes the maximum value for the true time lag μ.
σ_{min}^2	Describes the minimum value for the variance of time lag.
σ_{max}^2	Describes the maximum value for the variance of time lag.

We employ the exponential distribution to simulate the inter-arrival time between two adjacent events [9]. The average inter-arrival time β is randomly generated in the range $[\beta_{min}, \beta_{max}]$. The true lag μ is randomly generated in the

TABLE I: The experimental result for synthetic data. The size of data ranges from 200 to 40k; $\bar{\mu}$ and $\bar{\sigma}^2$ are the average values of μ and σ^2; LL_{opt} is the maximum log-likelihood. Assuming μ and σ^2 follow normal distribution, $\bar{\mu}$ and $\bar{\sigma}^2$ are provided with their 95% confidence interval for each algorithm over every data set. Entries with "N/A" are not available since it takes more than 1 day to get corresponding parameters.

	Ground Truth		lagEM			appLagEM $\epsilon = 0.001$			appLagEM $\epsilon = 0.05$			appLagEM $\epsilon = 0.1$		
N	μ	σ^2	$\bar{\mu}$	$\bar{\sigma}^2$	LL_{opt}	$\bar{\mu}$	$\bar{\sigma}^2$	LL_{opt}	$\bar{\mu}$	$\bar{\sigma}^2$	LL_{opt}	$\bar{\mu}$	$\bar{\sigma}^2$	LL_{opt}
200	77.01	44.41	77.41 [73.46, 81.36]	20.74 [18.75, 22.73]	-292.47	77.85 [73.52, 82.18]	24.68 [20.64, 28.72]	-290.99	78.21 [74.38, 82.03]	24.79 [20.62, 28.96]	-299.89	78.135 [74.16, 82.11]	25.02 [21.24, 28.80]	-300.05
1k	25.35	12.51	25.5 [25.0, 25.98]	8.66 [8.52, 8.80]	-1275.5	25.45 [25.12, 25.78]	8.62 [8.52, 8.72]	-1247.01	25.94 [24.39, 27.49]	9.296 [5.83, 12.77]	-1248.34	25.97 [24.35, 27.59]	9.36 [5.71, 13.0]	-1248.35
2k	38.54	30.88	38.68 [38.19, 39.17]	16.57 [16.38, 16.75]	-2847.0	38.81 [38.42, 39.40]	16.51 [16.45, 16.57]	-2820.6	39.78 [37.76, 41.78]	17.82 [14.17, 21.47]	-2822.60	39.32 [37.49, 41.14]	17.26 [14.36, 20.16]	-2822.57
10k	54.92	13.51	55.07 [54.60, 55.54]	8.84 [8.68, 9.0]	-12525.0	55.82 [53.97, 57.66]	10.92 [5.24, 16.60]	-12523.68	55.29 [54.23, 56.34]	9.40 [7.28, 11.52]	-12526.0	55.80 [53.99, 57.60]	10.85 [5.17, 16.53]	-12526.04
20k	59.35	17.22	59.42 [59.27, 59.57]	11.35 [11.32, 11.40]	-26554.2	59.67 [58.86, 60.50]	11.7 [10.35, 13.05]	-26332.68	59.38 [59.1, 59.70]	11.38 [11.30, 11.5]	-26332.06	59.34 [58.96, 59.72]	11.42 [11.2, 11.63]	-26336.39
40k	80.18	8.48	N/A	N/A	N/A	82.51 [77.76, 87.25]	5.26 [0.3, 10.3]	-40024.01	81.7 [78.15, 85.25]	4.45 [0.86, 8.04]	-40187.73	81.59 [77.9, 85.3]	4.4 [0.85, 7.94]	-40185.64

range $[\mu_{min}, \mu_{max}]$. And the variance of time lag σ^2 is generated between σ^2_{mix} and σ^2_{max} randomly.

With chosen parameters β, μ and σ^2, the procedure of generating synthetic data for the temporal dependency $A \rightarrow_\mu B$ is given below.

- Generate N timestamps for event A, where the inter-arrival time between two adjacent events follows the exponential distribution with parameter β.
- For each timestamp a_i for event A, the time lag is randomly generated according to normal distribution with parameters μ and σ^2.
- Combine all the timestamps associated with their types to build a synthetic data set.

We set $\beta_{min} = 5$, $\beta_{max} = 50$, $\mu_{min} = 25$, $\mu_{max} = 100$, $\sigma^2_{min} = 5$ and $\sigma^2_{max} = 400$ to synthesize the six data sets with different parameters N. The numbers of events for the synthetic data sets are 200, $1k$, $2k$, $10k$, $20k$ and $40k$, respectively. Recall that the number of events only describes the number of events of two types we are interested in. In practice, a real data set typically gets more than hundreds of events types in addition to the two considered types of events. Thus $40k$ events of two types compare with a real data set containing 2 million events of 100 types.

2) Synthetic data evaluation: Since the EM based approach cannot guarantee the global optimum [5], we define a batch as running the experiments 20 rounds with different initial parameters chosen at random, where we empirically find out 20 rounds is reasonable for our problem. We choose the one with the maximum likelihood among 20 rounds as the solution of a batch. Ten such batches are conducted over each data set. With 10 pairs of parameters μ and σ^2 learnt from 10 batches, $\bar{\mu}$ and $\bar{\sigma}^2$ are calculated as an average values of μ and σ^2, respectively. Furthermore, 95% confidence intervals of μ and σ^2 are provided by assuming both μ and σ^2 follow the normal distribution as the prior [5]. Additionally, LL_{opt} denotes the maximum log-likelihood value learn by our proposed algorithms. As shown in Table I, results of experiments running $lagEM$ and $appLagEM$ are presented.

Each algorithm stops searching as it converges or the number of iterations exceeds 500. Algorithm $appLagEM$ takes one more parameter ϵ as its input, where ϵ determines the proportion of the neglected components during the parameter estimation of each iteration. Herein, ϵ has been set to 0.001, 0.05 and 0.1. For all data sets listed in Table I, time lags μs learnt by $lagEM$ and $appLagEM$ are quite close to the ground truth. In addition, the smaller ϵ is, the more probable that Algorithm $appLagEM$ will get a larger log likelihood.

Further, we employ the Kullback-Leibler(KL) divergence as the metric to measure the difference between the distribution of time lag given by the ground truth and the discovered results [8]. Since each algorithm with a different initial setting of parameters runs for 10 batches over a given data set, we take the average KL divergence of 10 batches to evaluate the experimental result. As shown in Fig.4, the KL divergence caused by $appLagEM$ is almost as small as the one produced by $lagEM$. Besides, as ϵ increases, the KL divergence of $appLagEM$ becomes larger.

Fig.5 presents the comparison of time cost over the synthetic data sets. It shows that the approximation algorithm $appLagEM$ is much more efficient than $lagEM$. It also shows that the larger the ϵ is, the less time $appLagEM$ takes to find the optimal distribution of the time lags. Algorithm $appLagEM$ even with $\epsilon = 0.001$ is about two orders of magnitude faster than Algorithm $lagEM$.

In conclusion, based on the comparative discussion of both $lagEM$ and $appLagEM$, it is possible to achieve a good balance in terms of accuracy and efficiency.

C. Real Data

We perform the experiment over two real event data sets collected from several IT outsourcing centers by IBM Tivoli

TABLE III: The snippet of discovered time lags

	Dependency	μ	σ^2	Signal-to-noise ratio
dataset1	$TEC\_Error \rightarrow_L Ticket\_Retry$	0.34059	0.107178	1.04
	$AIX\_HW\_ERROR \rightarrow_L AIX\_HW\_ERROR$	10.92	0.98	11.03
	$AIX\_HW\_ERROR \rightarrow_L NV390MSG\_MVS$	33.89	1.95	24.27
	$AIX\_HW\_ERROR \rightarrow_L Nvserverd\_Event$	64.75	2.99	37.45
	$AIX\_HW\_ERROR \rightarrow_L generic\_postemsg$	137.17	18.81	31.63
	$generic\_postemsg \rightarrow_L TSM\_SERVER\_EVENT$	205.301	39.36	32.72
	$generic\_postemsg \rightarrow_L Sentry2\_0\_diskusedpct$	134.51	71.61	15.90
	$MQ\_CONN\_NOT\_AUTHORIZED \rightarrow_L TSM\_SERVER\_EVENT$	1167.06	142.54	97.75
dataset2	$MSG\_Plat\_APP \rightarrow_L Linux\_Process$	18.53	2053.46	**0.408**
	$SVC\_TEC\_HEARTBEAT \rightarrow_L SVC\_TEC\_HEARTBEAT$	587.6	7238.5	6.90

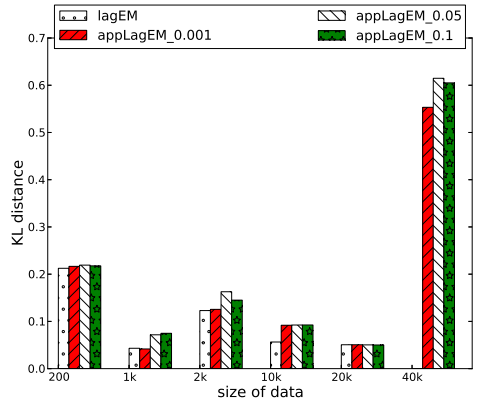

Fig. 4: The KL distance between the ground truth and the one learnt by each algorithm.

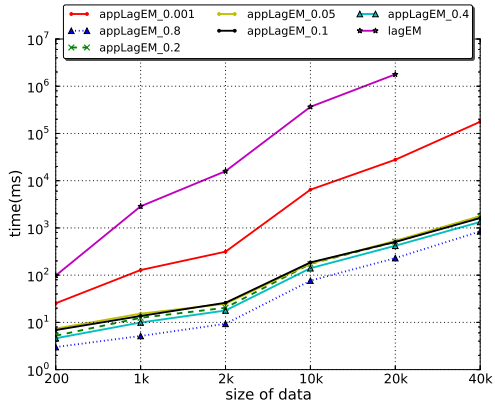

Fig. 5: Time cost comparison. ϵ of $appLagEM$ is set with 0.001, 0.05, 0.1, 0.2, 0.4, 0.8. The size of data set ranges from 200 to $40k$.

monitoring system [1] [17]. These events are generated by the automatic monitoring system with software agents running on the servers of an enterprise customer, which computes metrics for the hardware and software performance at regular intervals. The metrics are then compared to acceptable thresholds, known as monitoring situations, and any violation results in an alert. If the alert persists beyond a certain delay specified in the situation, the monitor emits an event. Therefore, a monitoring event corresponds to one type of system alert and one monitoring situation configured in the IBM Tivoli monitoring system. In this experiment, discovering temporal dependencies with time lags on monitoring events has several practical usages:

Monitoring Redundancy Removal: Many temporal dependent monitoring events are caused by correlated monitoring situations. For example, two situations monitoring CPU utilizations with similar thresholds are correlated. If the CPU has a high utilization, the two situations will generate one CPU event almost simultaneously. Then two CPU events are temporally dependent. Therefore, the discovered temporal dependencies can reveal the correlation of monitoring situations, in parallel monitoring. Removing this redundancy can reduce the running cost of monitoring agents on customer servers.

Event Correlation: Dependent monitoring events are usually triggered by the same system issue. The event correlation can merge dependent events into one ticket and help system administrator diagnose the system issue.

Root Cause Determination: Some temporal dependencies of system alerts can be seen as a fault-error-failure chain indicating the origin of the system issue. This chain can help the system administrator find the root cause of the related system issue and carry out effective system diagnosis.

Each real event set is collected from one IT environment of an enterprise customer. The number of events and types are listed in Table IV. The $dataset1$ consists of a sequence of events including 104 distinct event types, which are collected within the time span of 32 days. There are 136 types of events in $dataset2$ and $1000k$ events happen within 54 days. In both data sets, hundreds of types of events result in tens of thousands of pairs of event types. Since our algorithm takes a pair of events as the input, it would be time-consuming to consider all the pairs. In order to efficiently find the time lag of most possible dependent events, we filter out the types of events that appear less than 100 times in a corresponding data set.

TABLE IV: Real event data set

Name	# of events	# of types	Time span
dataset1	$100k$	104	32 days
dataset2	$1000k$	136	54 days

We employ the $appLagEM$ with $\epsilon = 0.001$ to mine the time lag of temporal dependency between two events. To increase the probability of getting the global optimal value, we run the algorithm in a batch of 50 rounds by feeding in random initial parameters every round. The snippet of some interesting time lags discovered is shown as Table III. The metric signal-to-noise ratio [15], a concept in signal processing, is used to measure the impact of noise relative to the expected time lag. Signal-to-noise is given as below:

$$SNR = \frac{\mu}{\sigma}.$$

The larger the SNR, the less relative impact of noise to the expected time lags.

$TEC\_Error \rightarrow_L Ticket\_Retry$ is a temporal dependency discovered from $dataset1$, where time lag L follows the normal distribution with $\mu = 0.34$ and the variance $\sigma^2 = 0.107178$. The small expected time lag μ less than 0.1 seconds indicates that the two events appear almost at the same time. And the small variance shows that most of time lags between the two event types are around the expected time lag μ. In fact, TEC_Error is caused whenever the monitoring system fails to generate an incident ticket to the ticket system. And $Ticket\_Retry$ is raised when the monitoring system tries to generate the ticket again.

$AIX\_HW\_Error \rightarrow_L AIX\_HW\_Error$ in $dataset1$ describes a pattern related to the event AIX_HW_Error. With the discovered μ and σ^2, the event AIX_HW_Error happens with an expected period about 10 seconds with small variance less than 1 seconds. In a real production environment, the event AIX_HW_Error is raised when monitoring system polls an AIX server which is down. The failure to respond to the monitoring system leads to an event AIX_HW_Error almost every 10 seconds.

In $dataset2$, the expected time lag between MSG_Plat_APP and $Linux\_Process$ is 18.53 seconds. However, the variance of the time lags is quite large relative to the expected time lag with $SNR = 0.4$. It leads to a weak confidence in temporal dependency between these two events because the discovered time lags get involved in too much noise. In practice, MSG_Plat_APP is a periodic event which is the heartbeat signal sent by the applications. However, the event $Linux\_Process$ is related to the different processes running on the Linux. So it is reasonable to assume a weak dependency between them.

The event $SVC\_TEC\_HEARTBEAT$ is used to record the heartbeat signal for reporting the status of service instantly. The temporal dependency discovered from the $dataset2$ shows that $SVC\_TEC\_HEARTBEAT$ is a periodic event with an expected period of 10 minutes. Although the variance seems large, the standard deviation is relatively small compared with the expected period μ. Therefore, it still strongly indicates the periodic temporal dependency.

The inter-arrival pattern can also be employed to find the time lag between events such as $TEC\_Error \rightarrow_{[t-\delta,t+\delta]} Ticket\_Retry$ where t and δ is very small. However, it fails to find the temporal pattern such as $MQ\_CONN\_NOT\_AUTHORIZED \rightarrow_L TSM\_SERVER\_EVENT$ with a large expected time lag about of 20 minutes. The reason is that inter-arrival pattern is discovered by only considering the inter-arrival time lag, and the inter-arrival time lags are exactly the small time lags.

In [18], Algorithm $STScan$ based on the support and the χ^2 test is proposed to find the interleaved time lags between events. Algorithm $STScan$ can find the temporal pattern such as $AIX\_HW\_Error \rightarrow_{[25,25]} AIX\_HW\_Error$ and $AIX\_HW\_Error \rightarrow_{[8,9]} AIX\_HW\_Error$ by setting the support threshold and the confidence level of χ^2 test. In our algorithm, we describe temporal patterns through expected time lag and its variance.

VI. Conclusion and Future Work

In this paper, we propose a novel parametric model to discover the distribution of interleaved time lags of the fluctuating events by introducing EM-based algorithm. In order to find the distribution of time lag for a massive events set, a near linear approximation algorithm is proposed. Extensive experiment conducted on both synthetic and real data show its efficiency and effectiveness.

In the future, we will extend our model to discover temporal patterns with more complicated distributions of time lags, such as patterns with possibly multiple time lags existing between two events and satisfying more complicated distribution laws. Moreover, it is more challenging to discover the dependencies among multiple events other than pairwise dependencies. Those more realistic conditions of real world will be considered in our future work.

Acknowledgement

The work of C. Zeng, L. Tang and T. Li is partially supported by the National Science Foundation under grants CNS-1126619 and IIS-1213026 and Army Research Office under grant number W911NF-10-10366 and W911NF-12-1-0431.

References

[1] IBM Tivoli Monitoring. http://www-01.ibm.com/software/tivoli/products/monitor/.

[2] ITIL. http://www.itil-officialsite.com/.

[3] Algirdas Avizienis, Jean-Claude Laprie, and Brian Randell. *Fundamental concepts of dependability*. University of Newcastle upon Tyne, Computing Science, 2001.

[4] Jay Ayres, Jason Flannick, Johannes Gehrke, and Tomi Yiu. Sequential pattern mining using a bitmap representation. In *Proceedings of KDD*, pages 429–435, 2002.

[5] Christopher M Bishop et al. *Pattern recognition and machine learning*, volume 1. springer New York, 2006.

[6] Jiawei Han, Jian Pei, Behzad Mortazavi-Asl, Qiming Chen, Umeshwar Dayal, and Meichun Hsu. Freespan: frequent pattern-projected sequential pattern mining. In *Proccedings of KDD*, pages 355–359, 2000.

[7] Manilla Heikki, Toivonen Hannu, and Verkamo A. Inkeri. Discovering frequent episodes in sequences. In *Proceedings of the First International Conference on Knowledge Discovery and Data Mining*, pages 210–215, 1995.

[8] Solomon Kullback and Richard A Leibler. On information and sufficiency. *The Annals of Mathematical Statistics*, pages 79–86, 1951.

[9] Tao Li, Feng Liang, Sheng Ma, and Wei Peng. An integrated framework on mining logs files for computing system management. In *Proceedings of ACM KDD*, pages 776–781, August 2005.

[10] Tao Li and Sheng Ma. Mining temporal patterns without predefined time windows. In *Data Mining, 2004. ICDM'04. Fourth IEEE International Conference on*, pages 451–454. IEEE, 2004.

[11] Sheng Ma and Joseph L Hellerstein. Mining mutually dependent patterns. In *Data Mining, 2001. ICDM 2001, Proceedings IEEE International Conference on*, pages 409–416. IEEE, 2001.

[12] Sheng Ma, Joseph L Hellerstein, Chang-shing Perng, and Genady Grabarnik. Progressive and interactive analysis of event data using event miner. In *Data Mining, 2002. ICDM 2003. Proceedings. 2002 IEEE International Conference on*, pages 661–664. IEEE, 2002.

[13] Jian Pei, Jiawei Han, Behzad Mortazavi-Asl, Helen Pinto, Qiming Chen, Umeshwar Dayal, and Meichun Hsu. Prefixspan: Mining sequential patterns by prefix-projected growth. In *Proceedings of EDBT*, pages 215–224, 2001.

[14] C.S. Perng, D. Thoenen, G. Grabarnik, S. Ma, and J. Hellerstein. Data-driven validation, completion and construction of event relationship networks. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge Discovery and Data Mining*, pages 729–734. ACM, 2003.

[15] D. J. Schroeder. *Astronomical optics*. Academic Press, 1999.
[16] Ramakrishnan Srikant and Rakesh Agrawal. Mining sequential patterns: Generalizations and performance improvements. In *Proceedings of EDBT*, pages 3–17, 1996.
[17] Liang Tang, Tao Li, Florian Pinel, Larisa Shwartz, and Genady Grabarnik. Optimizing system monitoring configurations for non-actionable alerts. In *Proceedings of IEEE/IFIP Network Operations and Management Symposium*, pages 34–42, 2012.
[18] Liang Tang, Tao Li, and Larisa Shwartz. Discovering lag intervals for temporal dependencies. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 633–641. ACM, 2012.
[19] Chunqiu Zeng, Tao Li, Larisa Shwartz, and Genady Ya Grabarnik. Hierarchical multi-label classification over ticket data using contextual loss. In *Proceedings of IEEE/IFIP Network Operations and Management Symposium*, pages 1–8. IEEE, 2014.

APPENDIX A

A. Algorithm $lagEM$

Algorithm 1 lagEM

1: **procedure** $LagEM(\mathbf{S_A},\mathbf{S_B})$
▷**Input:** two event sequences $\mathbf{S_A}$ and $\mathbf{S_B}$ with length m and n respectively.
▷**Output:** the estimated parameters μ and σ^2.
2: define r'_{ij}, μ' and σ'^2 as parameters of previous iteration
3: define r_{ij}, μ and σ^2 as the parameters of current iteration
▷ initialization
4: initialize $r'_{ij} = \frac{1}{m}$
5: initialize μ' and σ'^2 randomly
6: **while** true **do**
▷ expectation
7: evaluate the r_{ij} following equation (23)
▷ maximization
8: update μ following equation (20)
9: update σ^2 following equation (21)
▷test convergence
10: **if** parameters converge **then**
11: **return** μ and σ^2
12: **end if**
13: **end while**
14: **end procedure**

In Algorithm 1, the parameters are initialized in lines 2 to 5. There are $m \times n$ entries r'_{ij}, the time complexity for initialization is $O(mn)$. The optimized parameters are evaluated by an iterative procedure in lines 6 to 14, by finding expectation and maximizing it. The iterative procedure terminates when the parameters converge. Let r be the total number of iterations executed. The time cost of expectation (line 7) is $O(mn)$ since $m \times n$ entries r_{ij} need to be evaluated. The maximization part (lines 8 to 10) takes $O(mn)$ to update parameters of current iteration according to Equation (20), (21). Thus, the time complexity of iterative procedure is $O(rmn)$.

B. Algorithm $greedyBound$

In Algorithm $greedyBound$, line 3 employs a binary searching algorithm to locate the nearest a_i. It takes $O(\log m)$ time cost. The loop between line 6 and line 16 consumes $|C_j|$ time units. Let $K = |C_j|$. Then the total time complexity is $O(\log m + K)$.

C. Algorithm $appLagEM$

In $appLagEM$, let K be the average size of all C_j. Then the time complexity of line 8 is $O(\log m + K)$ and it takes $O(K)$ for line 9. Thus, from line 6 to line 10, the complexity is $O(n(\log m + K))$. Both line 11 and line 12 consume $O(nK)$. Therefore, the total time cost of $appLagEM$ is $O(rn(\log m + K))$ where r is the number of iterations.

D. Proofs of Propostion 1 and Lemma 2

Proof: (Proposition 1). The marginal probability is acquired by summing up the joint probability over all the $\mathbf{z_{\bullet j}}$, i.e.,

$$P(b_j|\mathbf{S_A},\mathbf{\Theta}) = \sum_{\mathbf{z_{\bullet j}}} \prod_{i=1}^{m} (P(b_j|a_i,\mathbf{\Theta}) \times P(z_{ij}=1))^{z_{ij}}.$$

Algorithm 2 greedyBound

1: **procedure** $greedyBound(\mathbf{S_A},b_j,\mu,\epsilon)$
▷**Input:** $\mathbf{S_A}$ contains all the possible timestamps of event A; b_j is the timestamp of the j^{th} event B; μ is the mean of time lags estimated in the previous iteration; ϵ is the probability of the timestamps of event $A \notin C_j$.
▷**Output:** min_j and max_j are the minimum and maximum indexes in C_j.
2: $t = b_j - \mu$
3: Locate the a_i to which t is closed using binary search.
4: $min_j = i$ and $max_j = i$
5: $prob = 0.0$
6: **while** $prob < 1 - \epsilon$ **do**
7: **if** $r_{(min_j-1)j} \geq r_{(max_j+1)j}$ **then**
8: $i = min_j - 1$
9: $min_j = i$
10: **else**
11: $i = max_j + 1$
12: $max_j = i$
13: **end if**
14: add a_i to C_j
15: $prob = prob + r_{ij}$
16: **end while**
17: **return** min_j and max_j.
18: **end procedure**

Algorithm 3 appLagEM

1: **procedure** $appLagEM(\mathbf{S_A},\mathbf{S_B},\epsilon)$
▷**Input:** two event sequences $\mathbf{S_A}$ and $\mathbf{S_B}$ with length m and n respectively. ϵ is the probability of the neglected part for estimating parameters.
▷**Output:** the estimated parameters μ and σ^2.
2: define r'_{ij}, μ' and σ'^2 as parameters of previous iteration
3: define r_{ij}, μ and σ^2 as the parameters of current iteration
▷ initialization
4: initialize $r'_{ij} = \frac{1}{m}$
5: initialize μ' and σ'^2 randomly
6: **while** true **do**
7: **for** each b_j **do**
▷find the index bound of a for each b_j
8: Get min_j and max_j by greedyBound
▷ expectation
9: evaluate the r_{ij} where $i \in [min_j, max_j]$
10: **end for**
▷ maximization
11: update μ by equation (20) within the bound
12: update σ^2 by equation (21) within the bound
▷test convergence
13: **if** parameters converge **then**
14: **return** μ and σ^2
15: **end if**
16: **end while**
17: **end procedure**

Among all m components in vector $\mathbf{z_{\bullet j}}$, there is only one component with value 1. Without any loss of generality, let $z_{ij} = 1$ given $\mathbf{z_{\bullet j}}$. Thus,

$$\prod_{i=1}^{m} (P(b_j|a_i,\mathbf{\Theta}) \times P(z_{ij}=1))^{z_{ij}} = P(b_j|a_i,\mathbf{\Theta}) \times P(z_{ij}=1).$$

Then, $P(b_j|\mathbf{S_A},\mathbf{\Theta}) = \sum_{\mathbf{z_{\bullet j}}} P(b_j|a_i,\mathbf{\Theta}) \times P(z_{ij}=1)$ There are m different $\mathbf{z_{\bullet j}}$ with $z_{ij} = 1$ where i ranges from 1 to m. Thus,

$$P(b_j|\mathbf{S_A},\mathbf{\Theta}) = \sum_{i=1}^{m} P(z_{ij}=1) \times P(b_j|a_i,\mathbf{\Theta}).$$

■

Proof: (Lemma 2) Since $< a_1, a_2, ..., a_m >$ is a time sequence, we can assume that $a_1 \leq a_2 \leq ... \leq a_m$. Thus, $b_j - a_i \in [b_j - a_m, b_j - a_1]$. Moreover, $\epsilon_j = \sum_{i|a_i\ is\ neglected} r_{ij}$, where $\epsilon_j \leq \epsilon$. Therefore, $\frac{1}{n}\sum_{j=1}^{n} \epsilon(b_j - a_m) \leq \mu_\delta \leq \frac{1}{n}\sum_{j=1}^{n} \epsilon(b_j - a_1)$. Then, we get $\mu_\delta \in [\epsilon(\bar{b} - a_m), \epsilon(\bar{b} - a_1)]$. In addition, $(b_j - a_i)^2 \leq \max\{(b_j - a_1)^2, (b_j - a_m)^2\}$. Thus, $\sigma_\delta^2 \leq \frac{1}{n}\epsilon\sum_{j=1}^{n} \max\{(b_j^2 - 2b_j a_1 + a_1^2, b_j^2 - 2b_j a_m + a_m^2)\}$.
Then, we get $\sigma_\delta^2 \leq \epsilon \max\{\bar{b^2} - 2\bar{b}a_1 + a_1^2, \bar{b^2} - 2\bar{b}a_1 + a_m^2\}$. So, $\sigma_\delta^2 \in [0, \epsilon\phi]$.
■