Learning Latent Events from Network Message Logs

Siddhartha Satpathi
UIUC
ssatpth2@illinois.edu

Supratim Deb
AT&T Labs
supratim@research.att.com

R Srikant
UIUC
rsrikant@illinois.edu

He Yan
AT&T Labs
yanhe@research.att.com

ABSTRACT

We consider the problem of separating error messages generated in large distributed data center networks into error events. In such networks, each error event leads to a stream of messages generated by hardware and software components affected by the event. These messages are stored in a giant message log. We consider the unsupervised learning problem of identifying the signatures of events that generated these messages; here, the signature of an error event refers to the mixture of messages generated by the event. One of the main contributions of the paper is a novel mapping of our problem which transforms it into a problem of topic discovery in documents. Events in our problem correspond to topics and messages in our problem correspond to words in the topic discovery problem. However, there is no direct analog of documents in our problem. Therefore, we use a non-parametric change-point detection algorithm, which has linear computational complexity in the number of messages, to divide the message log into smaller subsets called episodes, which serve as the equivalents of documents. After this mapping has been done, we use a well-known algorithm for topic discovery, called LDA, to solve our problem. We theoretically analyze the change-point detection algorithm, and show that it is consistent and has low sample complexity. We demonstrate the scalability of our algorithm (change-point detection and LDA together) on a real dataset consisting of 97 million messages collected over a period of 15 days, from a distributed data center network which supports the operations of a large wireless service provider.

KEYWORDS

Unsupervised Learning; Data Mining; Event Message Log; Change Point Detection; Bayesian Inference; Data center Networks

1 INTRODUCTION

The delivery of modern data and web-based services requires the execution of a chain of network functions at different elements in distributed data-centers. This is true for video-based services, gaming services, cellular data/voice services, etc., each of which requires processing from multiple coupled networked entities hosting different network functions. For example, modern wireless networks rely on servers and virtual machines (VM) residing in distributed data centers to establish voice calls or data sessions, authenticate users, check user compliance with monthly voice/data limits, verify if users have paid their monthly bills, add to users' bills for extra services, etc., all of which are done before completing a call. Efficient management and operations of these services is of paramount importance as networks grow increasingly complex with the advent of technologies like virtualization and 5G. An integral component of network management is the ability to identify and understand *error events*, when failures occur in the hardware and/or software components of the network. However, the complex interdependence between coupled networking functions poses a significant challenge in characterizing an error event due to the fact that error messages can be generated in network elements beyond the actual source of error. In this paper, we are interested in the problem of mining latent error event information from messages generated by servers, VMs, base stations, routers, and links in large-scale distributed data center networks. While our methodology is broadly applicable to any type of data center network, we validate our algorithms by applying them to a large data set provided by a major wireless network service provider, so we will occasionally use terminology specific to this application to motivate our problem and solution methodology.

In most operational networks, all messages and alarms from distributed network elements are logged with time stamps into a massive central message log. While mining error logs have been studied extensively in different contexts, (see [10, 11] for excellent surveys; also see Section 1.2), there are some fundamental differences in our setting. The key challenge stems from the fact that modern data center and communication networks consist of components bought from different vendors, and each component is designed to generate an error message when it cannot execute a job. The following example provides an illustration.

Motivating Example: Suppose Alice makes a cellphone call to Bob. This call is first routed through a base station which is attached to a data center verifying the caller credentials. If Alice is not at her home location, a VM at this data center must contact a database at her home location to verify her credentials. Once the credentials are verified, the caller's cellular base station connects to the base station near Bob through a complicated network spanning many geographical locations. Consider two potential error scenarios: (i) an error occurs at a router in the path from Bob to Alice's base station, (ii) error at a router connecting the data centers verifying the caller's credentials. In either scenario, the call will fail to be

The work of Satpathi and Srikant was partially supported by a grant from AT&T and NSF Grant NeTS 1718203.

MiLeTS '18, London, United Kingdom
© 2018 Copyright held by the owner/author(s). 123-4567-24-567/08/06. . . $15.00
DOI: 10.475/123_4

established leading to the generation of error messages not only at the failed routers but also at network functions (implemented in a cluster of VMs) responsible for call establishment. Furthermore, if the error leads to additional call failures, then respective base-stations could send alarms indicating higher than normal call failures.

Indeed, the source, timing, and message-components of the error are all latent. The goal of this paper is to mine event signatures (i.e., distribution of messages for each event) and event occurrences (i.e., the begin and the end time of each event) from the message log. Based on the motivating example, we now note the following fundamental characteristics which make our error event mining problem challenging:

- In our setting, the source of an error is usually not known and thus a log-message generated by an element could be due to failure of some other network element. For example, when the link between an authentication server and the network core fails, this could lead to call establishment failures which are logged by network functions responsible for call establishment. Furthermore, the same type of error log-message could be generated due to many different errors. From a data modeling point of view, each (latent) event can be viewed as a probabilistic-mixture of multiple log-messages at different elements and also, the set of log messages generated by different events could have non-zero intersection.
- Each error event can produce a sequence of messages, including the same type of message multiple times, and the temporal order between distinct messages from the same event could vary based on the latency between network elements, network-load, co-occurrence of other uncorrelated events, etc. Thus, the temporal pattern of messages may also contain useful information for our purpose. In our model, the message occurrence times are modeled as a stochastic process.
- All the messages generated in the network are stored in a centralized log. These messages could correspond to multiple simultaneous events without any further information on the start-time and end-time of each event.
- An additional challenge arises from the fact that network topology information is unknown, because modern networks are very complicated and are constantly evolving due to the churn (addition or deletion) of routers and switches. Third-party vendor software and hardware have no way of providing information to localize and understand the errors. Thus, topological information cannot be used for event mining purposes.

The practical novelty of our work comes from modeling for all of the above factors and proposing scalable algorithms that learns the latent event signatures (the notion of signature will be made precise later) along with their occurrence times.

Remark 1. We note that, in different works on event mining (see Section 1.2), the concept of event is different depending on the problem-context. It could either mean semantic-event or message template, or a cluster of such templates, or in some cases event itself is equivalent to message (where tagged event streams are available) or a transaction/system-event. In our work, an *event* simply refers to a real-world occurrence of an a fault/incident somewhere in the distributed/networking system such that each event leads to a generation of error messages at multiple network elements.

1.1 Contributions

We model each event as a probabilistic mixture of messages from different sources. In other words, the probability distribution over messages characterizes an event, and thus acts as the signature of the event. Each event also has a start/end time and several messages can be generated during the occurrence of an event. We only observe the messages and their time-stamps while the event signatures and duration window is unknown; also there could be multiple simultaneous events occurring in the network. Given this setting, we study the following unsupervised learning problem: *given collection of time-stamped log-messages, learn the latent event signatures and event start/end times.* The main contributions of the paper are as follows:

- *Novel algorithmic framework:* We present a novel way of decomposing the problem into simpler sub-problems. Our method, which we will call CD-LDA, decomposes the problem into two parts: the first part consists of a change-point detection algorithm which identifies time instants at which either a new event starts in the network or an existing event comes to an end, and, the second part of the algorithm uses Latent Dirichlet Algorithm (LDA) (see [2]) to classify messages into events. This observation that one can use change-point detection, followed by LDA, for event classification is one of the novel ideas in the paper.
- *Scalable change-point detection:* While the details of the LDA algorithm itself are standard, non-parametric change-point detection as we have used in this paper is not as well studied. We adapt an idea from [16] to design an $O(n)$ algorithm where n is the number of messages in the message log. The corresponding computational complexity in [16] is $O(n^2)$. Also, since the messages in [16] are real-valued it is not clear how to directly apply the results of [16] to our case where the messages are categorical in nature. The key insight for achieving linear complexity is that, an easy to compute distance measure, namely total-variation (TV) distance, can be used to detect changes in message distribution due to an event. Since we are dealing with millions of messages, the linear complexity of change-point detection is critical to making our algorithm useful in practice. We also analyze the sample complexity of (i.e, the number of samples required to detect change points with a high-degree of accuracy) of our change-point detection algorithm using the method of types and Pinsker's inequality from information theory. To the best of our knowledge, no such sample complexity results exist for the algorithm in [16].
- *Experimental validation:* We use two different real-word data sets from a large operational network to perform the following validation of our approach. First, we compare our algorithm to two existing approaches adapted to our setting: a Bayesian inference-based algorithm and Graph-based clustering algorithm. We show the benefits of our approach compared to these methods in terms of scalability and performance, by applying it to small samples extracted from a large data set consisting of 97 million messages. Second, we validate our method against two real world events by comparing the event signature learned by our method with event signature manually inferred by domain experts in a smaller data set consisting of 700K messages[1]. Finally, we also

[1]Note that manual inference of event signatures is not scalable; we did this for the purpose of validation.

show results to indicate scalability of our method by applying to the entire 97 million message data set.

1.2 Context and Related Work

Data-driven techniques have been shown to be very useful in extracting meaningful information out of system-logs and alarms for large and complex systems. The primary goal of this "knowledge" extraction is to assist in diagnosing the underlying problems responsible for log-messages and events. Two excellent resources for the large body of work done in the area are [10, 11]. Next, we outline some of the key challenges in this knowledge extraction, associated research in the area, and our problem in the context of existing work.

Mining and clustering unstructured logs: Log-messages are unstructured textual data without any annotation for the underlying fault. A significant amount of research has focused on converting unstructured logs to common *semantic events* [11]. Note that the notion of *semantic events* is different from the actual real-world events responsible for generating the messages, nevertheless, such a conversion helps in providing a canonical description of the log-messages that enables subsequent correlation analysis . These works exploit the structural similarity among different messages to either compute an intelligent log-parser or cluster the messages based on message texts [9, 11, 13, 20]. Each cluster can be viewed as an semantic event which can help in diagnosing the underlying root-cause. Our setting is somewhat different, as events are messages-distributions from different elements with certain start and end times; the messages belonging to an event and the associated event time-window are hidden (to be learned). A more recent work [26] develops algorithms to mine underlying structural-event as a work-flow graph. The main differences are that, each transaction is a fixed sequence of messages unlike our setting where each message could be generated multiple times based on some hidden stochastic process, and furthermore, in our setting, there could be multiple events manifested in the centralized log-server.

Mining temporal patterns: Log-messages are time-series data and thus the temporal patterns contain useful information. Considerable amount of research has gone into learning latent patterns, trends and relationship between events based on timing information in the messages [1, 3, 27]. We refer to [11, 14, 18] for survey of these approaches. Extracted event-patterns could be used to construct event correlation graphs that could be mined using techniques such as graph-clustering. Specifically, these approaches are useful when event-streams are available as time-series. We are interested in scenarios where each event is manifested in terms of time-series of unstructured messages and furthermore, same message could arise from multiple events. Nevertheless, certain techniques developed for temporal event mining could be adapted to our setting as we describe in Section 4.1.2; our results indicate that such an adaptation works well under certain conditions. Note that, our goal is to also the learn the event-occurrence times.

Event-summarization: In large dynamic systems, messages could be generated from multiple components due to reasons ranging from software bugs, system faults, operational activities, security alerts etc. Thus it is very useful to have a global summarized snapshot of messages based on logs. Most works in this area exploit the inter-arrival distribution and co-occurrence of events [7, 11, 15, 21, 24] to produce summarized correlation between events. These methods are useful when the event-stream is available and possible event-types are known in advance. This limits the applicability to large-systems like ours where event types are unknown along with their generation time-window.

The body of work closest to out work are the works on event-summarization. However, there are some fundamental differences in our system: (i) we do not have a readily available event-stream, instead, our observables are log-messages, (ii) the event-types are latent variables not known in advance and all we observe are message streams, (iii) the time-boundaries of different latent-events is a learning objective, and (iv) since we are dealing with large system with multiple components where different fault-types are correlated, the same message could be generated for different root-causes (real-world events).

2 PROBLEM STATEMENT AND PRELIMINARIES

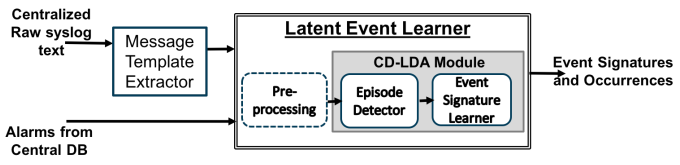

Figure 1: Figure showing the machine-learning pipeline. Our main contribution is in "Latent Event Learner" module, specifically proposing the CD-LDA algorithm.

We are given a data set $\mathcal{D}$ consisting of messages generated by error events in a large distributed data-center network. We assume that the messages are generated in the time interval $[0, T]$. The set of messages in the data set come from discrete and finite set $\mathcal{M}$. Note that, for the purpose of developing our methods, each message need not be raw-text; it could either be templates extracted out of raw textual messages using techniques described in [11], or some alarm id. In fact, converting the raw-textual messages to templates is an important pre-processing step in our experimental set-up (see [17]). For the purpose of developing and analyzing our algorithms, we use the term message to mean either a template extracted from a message or an alarm-id. Each message has a timestamp associated with it, which indicates when the message was generated. Suppose that an event e started occurring at time S_e and finished at time F_e. In the interval of time $[S_e, F_e]$, event e will generate a mixture of messages from a subset of $\mathcal{M}$, which we will denote by $\mathcal{M}_e$. In general, an event can occur multiple times in the data set. If an event e occurs multiple times in the data set, then each occurrence of the event will have start and finish times associated with it.

An event e is characterized by its message set $\mathcal{M}_e$ and the probability distribution with which messages are chosen from $\mathcal{M}_e$, which we will denote by $p^{(e)}$, i.e., $p_m^{(e)}$ denotes the probability that event e will generate a message $m \in \mathcal{M}_e$. For compactness of notation, one can simply think of $p^{(e)}$ as being defined over the entire set of messages $\mathcal{M}$, with $p_m^{(e)} = 0$ if $m \notin \mathcal{M}_e$. Thus, $p^{(e)}$ fully characterizes the event e and can be viewed as the signature of the event. We assume that the support set of messages for two different events are not identical.

It is important to note that the data set simply consists of messages from the set $\mathcal{M}$; there is no explicit information about the events in the data set, i.e., the event information is latent. The goal of the paper is to solve the following inference problem: from the given data set $\mathcal{D}$, identify the set of events that generated the messages in the data set, and for each instance of event, identify when it started and finished. In other words, the output of the inference algorithm should contain the following information:

- The number of E events which generated the data set.
- The signatures of these events: $p^{(1)}, p^{(2)}, \ldots, p^{(E)}$.
- For each event $e \in \{1, 2, \ldots, E\}$, the number of times it occurred in the data set and, for each occurrence, its start and finish times.

Notation: We use the notation $X_i \in \mathcal{M}$, for the i^{th} message. Also, let t_i be the timestamp associated with the i^{th} message. Thus the data set $\mathcal{D}$ can be characterized by tuples $(X_1, t_1), (X_2, t_2), \ldots (X_n, t_n)$ of n data points.

Machine-learning pipeline: In Figure 1, we show the machine-learning pipeline for completeness. This paper focuses on the module "Latent Event Learner" which has data-processing step followed by the key proposed algorithm in the paper, namely CD-LDA algorithm which we describe in Section 3. In the real data set to which we applied our algorithm, there are two types of messages generated by an event: one is called syslog text and the other called an alarm. Syslog texts require more pre-processing while alarms do not. We have shown the two types of messages in the pipeline figure, but for the purposes for developing an algorithm, in the rest of the paper, we only refer to messages without distinguishing between them.

3 ALGORITHM CD-LDA

We now present our solution to this problem which we call CD-LDA (Change-point Detection-Latent Dirichlet Analysis). The key novelty in the paper is the connection that we identify between event identification in our problem and topic modeling in large document data sets, a problem that has been widely studied in the natural language processing literature. In particular, we process our data set into a form that allows us to use a widely-used algorithm called LDA to solve our problem. In standard LDA, we are given multiple documents, with many words in each document. The goal is to identify the mixture of latent topics that generated the documents, where each topic is identified with a collection of words and a probability distribution over the words. Our data set has similar features: we have events (which are the equivalents of topics) and messages (which are the equivalents of words) which are generated by the events. However, we do not have a concept of documents. A key idea in our paper is to divide the data set into smaller data sets, each of which will be called an episode. The episodes will be the equivalents of documents in our problem. We do this using a technique called non-parametric change-point detection.

Now we describe the concept of an episode. An episode is an interval of time over which the same set of events occur, and at time instants on either side of the interval, the set of events that occur are different from the set of events in the episode. Thus, we can divide our data set of events such that no two consecutive episodes have the same set of events. We present an example to clarify the concept of an episode. Suppose the duration of the message data set $T = 10$. Suppose event 1 occurred from time 0 to time 5, event 2 occurred from time 4 to time 8, and event 3 occurred from time 5 to time 10. Then there are four episodes in this data set: one in the time interval $[0, 4]$ where only one event occurs, one in the time interval $[4, 5]$ where events 1, 2 occur, one in the time interval $[5, 8]$ where events 2, 3 occur and finally one in $[8, 10]$ where only event 3 occurs. We assume that between successive episodes, at most one new event starts or one existing event ends.

We use change-point detection to identify episodes. To understand how the change-point detection algorithm works, we first summarize the characteristics of an episode:

- An episode consists of a mixture of events, and each event consists of a mixture of messages.
- Since neighboring episodes consist of different mixtures of events, neighboring episodes also contain different mixtures of messages (due to our assumption that different events do not generate the same set of messages).
- Thus, successive episodes contain different message distributions and therefore, the time instances where these distributions change are the episode boundaries, which we will call *change points*.
- In our data set, the messages contain time stamps. In general, the inter-arrival time distributions of messages are different in successive episodes, due to the fact that the episodes represent different mixtures of events. This fact can be further exploited to improve the identification of change points.

Based on our discussion so far in this section, CD-LDA has two-phases as follows:

(1) *Change-point detection:* In this phase, we detect the start and end time of each episode. In other words, we identify the time-points where a new event started or an existing event ended. This phase is described in detail in Section 3.1.
(2) *Applying LDA:* In this phase, we show that, once episodes are known, LDA based techniques can be used to solve the problem of computing message distribution for each event. Subsequently, we can also infer the occurrence times for each event. This phase along with the complete algorithm is described in Section 3.2.

3.1 Change-point Detection

Suppose we have n data points and a known number of change points k. The data points between two consecutive change points are drawn i.i.d from the same distribution. In the inference problem, each data point could be a possible change point. A naive exhaustive search to find the k best locations would have a computational complexity of $O(n^k)$. Nonparametric approaches to change-point detection aim to solve this problem with much lower complexity even when the number of change points is unknown and there are few assumptions on the family of distributions, [8], [16],[12].

The change point detection algorithm we use is hierarchical in nature. This is inspired by the work in [16]. Nevertheless our algorithm has certain key differences as discussed in section 3.3.1. It is easier to understand the algorithm in the setting of only one change point, i.e., two episodes. Suppose that τ is a candidate change point among the n points. The idea is to measure the change in distribution between the points to the left and right of τ. We use the TV distance (i.e. L_1 distance between two distributions) between the empirical distributions estimated from the points to the left and right of the candidate change point τ. This is maximized over all values of τ to estimate the location of the change point. If the distributions are sufficiently different in the two episodes the TV distance between the empirical distributions is expected to be highest for the correct location of the change point in comparison to any other candidate point τ (we rigorously prove this in the proof Theorem 3.1, 3.3).

Further, we also have different inter-arrival times for messages in different episodes. Hence we use a combination of TV distance and mean inter-arrival time as the metric to differentiate the two distributions. We denote this metric by $\widehat{D}(\tau)$.

$$\widehat{D}(\tau) = \|\widehat{p}_L(\tau) - \widehat{p}_R(\tau)\|_1 + |\widehat{\mathbb{E}}S_L(\tau) - \widehat{\mathbb{E}}S_R(\tau)|, \tag{1}$$

where $\widehat{p}_L(\tau), \widehat{p}_R(\tau)$ are empirical estimates of message distributions to the left and right τ and $\widehat{\mathbb{E}}S_L(\tau), \widehat{\mathbb{E}}S_R(\tau)$ are empirical estimates of the mean inter-arrival time to the left and right of τ, respectively. Algorithm 1 describes the algorithm in the one change point case. To make the algorithm more robust, we declare a change point only when the episode length is at least αn and the maximum value of the metric (1) is at least δ.

Let us consider a simple example to illustrate the idea of change-point detection with one change-point. Suppose we have a sequence of messages with unequal inter-arrival times as shown in Fig. 2. All the messages are the same, but the first half of the messages arrive at a rate higher than the second half of the messages. In this scenario, our metric reduces to the difference in the mean inter-arrival times between the two episodes.

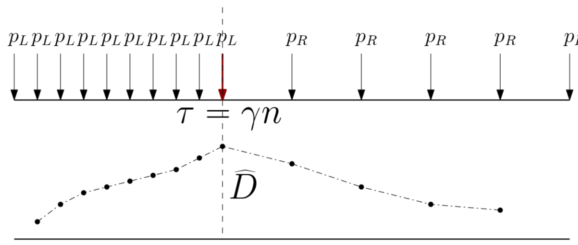
p_L p_L p_L p_L p_L p_L p_L p_L p_L p_L
p_R p_R p_R p_R p_R
$\tau = \gamma n$
$\widehat{D}$

Figure 2: Example change point with two episodes

Next, we consider the case of multiple change points. When we have multiple change points, we apply Algorithm 1 hierarchically until we cannot find a change point. Algorithm 2 CD($\mathcal{D}, \alpha, \delta$) is presented below.

The above algorithm tries to detect a single change point first, and if such a change point is found, it divides the data set into two parts, one consisting of messages to the left of the change point and the other consisting of messages to the right of the change point. The single change-point detection algorithm is now applied to each

Algorithm 1 Change point detection with one change point

```
1: Input: parameter $\delta > 0, \alpha > 0$.
2: Output: changept denoting whether a change point exists and
   the location of the change point $\tau$.
3: Find $\tau \in \arg\max_l \widehat{D}(l)$
4: if $\widehat{D}(\tau) > \delta$ and $\alpha n < \tau < 1 - \alpha n$ then
5:     return changept = 1, $\tau$.
6: else
7:     return changept = 0.
```

Algorithm 2 CD($\mathcal{D}, \alpha, \delta$)

```
1: Input: data points $\mathcal{D}$, minimum value of TV distance $\delta$, mini-
   mum episode length $\alpha$.
2: Output: Change points $\tau_1, \ldots, \tau_k$.
3: Run FindChangept(1, n).
4: procedure FindChangept(L, H)
5:     changept, $\tau$ ← Algorithm 1 ($X_L, X_{L+1}, \ldots, X_H, \alpha, \delta$).
6:     if changept exists then
7:         $\tau_l$ ← FindChangept(L, $\tau$),
8:         $\tau_h$ ← FindChangept($\tau$, H).
9:         return {$\tau_l, \tau, \tau_h$}
10:    else
11:        return
```

of the two smaller datasets. This is repeated recursively till no more change points are detected.

3.2 Latent Dirichlet Allocation

In the problem considered in this paper, each episode can be thought of as a document and each message can be thought of as a word. Like in the LDA model where each topic is latent, in our problem, each event is latent and can be thought of as a distribution over messages. Unlike LDA-based document modeling, we have time-stamps associated with messages, which we have already used to extract episodes from our data set. Additionally, this temporal information can also be used in a Bayesian inference formulation to extract events and their signatures. However, to make the algorithm simple and computationally tractable, as in the original LDA model, we assume that there is no temporal ordering to the episodes or messages within the episodes. Our experiments suggest that this choice is reasonable and leads to very good practical results. However, one can potentially use the temporal information too as in [22, 25], and this is left for future work.

If we apply the LDA algorithm to our episodes, the output will be the event signatures $p^{(e)}$ and episode signatures $\theta^{(\mathcal{E})}$, where an episode signature is a probability distribution of the events in the episode. In other words, LDA assumes that each message in an episode is generated by first picking an event within an episode from the episode signature and then picking a message from the event based on the event signature. In order to perform this inference of event and episode signatures using LDA, many inference techniques exist: Gibbs sampling [4], variational inference [2], online variational inference [5], stochastic variational inference [6]. We use the Gibbs sampling approach from [4].

For our event mining problem, we are interested in event signatures and finding the start and finish times of each occurrence of an event. Therefore, the final step (which we describe next) is to extract the start and finish times from the episode signatures.

Putting it all together: In order to detect all the episodes in which the event e occurs prominently, we proceed as follows. We collect all episodes $\mathcal{E}$ for which the event occurrence probability $\theta_e^{(\mathcal{E})}$ is greater than a certain threshold $\eta > 0$. We declare the start and finish times of the collected episodes as the start and finish times of the various occurrences of the event e. If an event spans many contiguous episodes, then the start time of the first episode and the end time of the last contiguous episode can be used as the start and finish time of this occurrence of the event. However, for simplicity, this straightforward step in not presented in the detailed description of the algorithm in Algorithm 3.

Algorithm 3 CD-LDA($\mathcal{D}, \alpha, \delta, \eta$)

```
1: Input: data points $\mathcal{D}$, threshold of occurrence of an event in an episode $\eta$, the minimum value of TV distance $\delta$, minimum episode length $\alpha$.
2: Output: Event signatures $p^{(1)}, p^{(2)}, \dots, p^{(E)}$, Start and finish time $S_e, F_e$ for each event $e$.
3: Change points $\tau_1, \dots, \tau_k \leftarrow$ CD($\mathcal{D}, \alpha, \delta$). Episode $\mathcal{E}_i \leftarrow \{X_{\tau_{i-1}}, \dots, X_{\tau_i}\}$ for $i = 1$ to $k+1$.
4: $p^{(1)}, \dots, p^{(E)}; \theta^{(\mathcal{E}_1)}, \dots, \theta^{(\mathcal{E}_{k+1})} \leftarrow$ LDA($\mathcal{E}_1, \dots, \mathcal{E}_{k+1}$)
5: Consider event $e$. $\mathcal{G}_e \leftarrow$ Set of all episodes $\mathcal{E}$ such that $\theta_e^{(\mathcal{E})} > \eta$. $S_e, F_e \leftarrow$ start and finish times of all episodes in set $\mathcal{G}_e$.
```

Note that the LDA algorithm requires an input for the number of events E. However, one can run LDA for different values of E and choose the one with maximum likelihood [2]. Hence E need not be assumed to be an input to CD-LDA. One can also use the Hierarchical Dirichlet Process (HDP) algorithm [23] which is an extension of LDA and figure out the number of topics from the data. In our experiments, we use the maximum likelihood approach to estimate the number of events.

3.3 Analysis of CD

As mentioned earlier, the novelty in the CD-LDA algorithm lies in the connection we make to topic modeling in document analysis. In this context, our key contribution is an efficient algorithm to divide the data set of messages into episodes (documents). Once this is done, the application of the LDA of episodes (documents), consisting of messages (words) generated by events (topics) is standard. Therefore, the correctness and efficiency of the CD part of the algorithm will determine the correctness and efficiency of CD-LDA as a whole. We focus on analyzing the CD part of the algorithm in this section. Due to space limitations, we only present the main results here, and the proofs can be found in [17].

Section 3.3.1 shows that the computational complexity of CD algorithm is linear in the number of data points. Section 3.3.2 contains the asymptotic analysis of the CD algorithm while section 3.3.3 has the finite sample results.

3.3.1 Computational complexity of CD. In this section we discuss the computational complexities of Algorithm 1 and Algorithm 2. We will first discuss the computational complexity of detecting a change point in case of one change point. Algorithm 1 requires us to compute $\arg\max_l \widehat{D}(l)$ for $1 \le l \le n$. From the definition of $\widehat{D}(l)$ in (1), we only need to compute the empirical probability estimates $\widehat{p}_L(l), \widehat{p}_R(l)$, and the empirical mean of the inter arrival time $\widehat{\mathbb{E}S}_L(l)$, $\widehat{\mathbb{E}S}_R(l)$ for every value of l between 1 to n.

We focus on the computation of $\widehat{p}_L(l), \widehat{p}_R(l)$. Consider any message m in the distribution. For each m, we can compute $\widehat{p}_{L,m}(l)$, $\widehat{p}_{R,m}(l)$ in $O(n)$ for every value of l by using neighbouring values of $\widehat{p}_{L,m}(l-1), \widehat{p}_{R,m}(l-1)$.

$$\widehat{p}_{L,m}(l) = \frac{(l-1)\widehat{p}_{L,m}(l-1) + \mathbb{1}\{X_{l-1} = m\}}{l},$$
$$\widehat{p}_{R,m}(l) = \frac{(n-l+1)\widehat{p}_{R,m}(l-1) - \mathbb{1}\{X_{l-1} = m\}}{n-l} \quad (2)$$

The computation of $\widehat{\mathbb{E}S}_L(l)$, $\widehat{\mathbb{E}S}_R(l)$ for every value of l from 1 to n is similar.

Performing the above computations for all M messages, results in a computational complexity of $O(nM)$. In the case of k change points, it is straightforward to see that we require $O(nMk)$ computations. In much of our discussion, we assume M and k are constants and therefore, we present the computational complexity results in terms of n only.

Related work: Algorithm 2 executes the process of determining change points hierarchically. This ideas was inspired by the work in [16]. However, the metric $\widehat{D}$ we use to detect change points is different from that of [16]. The metric used in [16] leads to an $O(n^2)$ computational complexity. The change in metric necessitates a new analysis of the consistency of the CD algorithm which we present in the next subsection. Further, for our metric, we are also able to derive sample complexity results which are presented in a later subsection.

3.3.2 The consistency of change-point detection. In this section we discuss the consistency of the change-point detection algorithm, i.e., when the number of data points n goes to infinity one can accurately detect the location of the change points. In both this subsection and the next, we assume that the inter-arrival times of messages within each episode are i.i.d., and are independent (with possibly different distributions) across episodes.

Theorem 3.1. *For $\widetilde{\gamma} \in (0,1)$, $D(\widetilde{\gamma}) = \lim_{n\to\infty} \widehat{D}(\widetilde{\gamma}n)$ is well-defined and $D(\widetilde{\gamma})$ attains its maximum at one of the change points if there is at least one change point.*

The proof of the above theorem is easy when there is only one change point. To study the case of multiple change points, [16] exploits the fact that their metric for change-point detection is convex between change points. However, the TV distance we use is not convex between two change points. But we work around this problem in the proof of Theorem 3.1 by showing that $D(\widetilde{\gamma})$ is increasing to the left of the first change point, unimodal/increasing/decreasing between any two change points and decreasing to the right of the last change point. Hence, any global maximum of $D(\widetilde{\gamma})$ for $0 < \widetilde{\gamma} < 1$ is located at a change point.

3.3.3 The sample complexity of change-point detection. In the previous subsection, we studied the CD algorithm in the limit as $n \to \infty$. In this section, we analyze the algorithm when there are only a finite number of samples. For this purpose, we assume that the inter-arrival distribution of messages have sub-Gaussian tails.

We say that Algorithm CD is correct if the following conditions are satisfied. Let $\epsilon > 0$ be a desired accuracy in estimation of the change point.

Definition 3.2. Given $\epsilon > 0$, Algorithm CD is correct if

- there are change points $0 < \frac{\tau_1}{n} = \gamma_1, \ldots, \frac{\tau_k}{n} = \gamma_k < 1$ and the algorithm gives $\widehat{\gamma}_1, \ldots, \widehat{\gamma}_k$ such that $\max_i |\widehat{\gamma}_i - \gamma_i| < \epsilon$.
- there is no change point and $\widehat{D}(\gamma n) < \delta, \forall \gamma \in \{\gamma_1, \ldots, \gamma_k\}$.

Now we can state the correctness theorem for Algorithm 2. The sample complexity is shown to scale logarithmically with the number of change points.

Theorem 3.3. *Algorithm 2 is correct in the sense of Definition 3.2 with probability* $(1-\beta)$ *if*

$$n = \Omega\left(\max\left(\frac{\log\left(\frac{2k+1}{\beta}\right)}{\epsilon^2}, \frac{M^{1+c}}{\epsilon^{2(1+c)}}\right)\right),$$

for sufficiently small α, δ, ϵ *and for any* $c > 0$.

The proof of this theorem uses the method of types and Pinsker's inequality [17].

4 EVALUATION WITH REAL DATASETS

We now present our experimental results with real data sets from large operational network. The purpose of experiments is three-fold. First, we wish to compare our proposed CD-LDA algorithm with other techniques proposed (adapted to our setting) in the literature. Second, we want to validate our results against manual expert-derived event signature for a prominent event. Third, we want to understand the scalability of our method with respect to very large data sets.

Datasets used: We use two data sets: one from a legacy network of physical elements like routers, switches etc., and another from a recently deployed virtual network function (VNF).

- **Dataset-1:** This data set consists of around 97 million raw syslog messages collected from 3500 distinct physical network elements (mostly routers) from a nationwide operational network over a 15-day period in 2017.
- **Dataset-2:** The second data set consists of around 728, 000 messages collected from 285 distinct physical/virtual network elements over a 3 month period from a newly deployed *virtual network function* (VNF) which is implemented on a data-center using multiple VMs.

Before we can apply our algorithms to these data sets, a number of preprocessing steps have to be performed. Due to space limitations, these steps are not discussed here, but the interested reader can find them in [17].

4.1 Benchmark Algorithms

We compare CD-LDA with the following algorithms.

4.1.1 Algorithm B: A Bayesian inference based algorithm. We consider a fully Bayesian inference algorithm to solve the problem. A Bayesian inference algorithm requires some assumptions on the statistical generative model by which the messages are generated. Our model here is inspired by topic modelling with time stamped documents [25]. In our model, a message and its time stamp are chosen as follows: an event is chosen from a distribution over event types. A message is chosen according to a message distribution given the event. Each event is associated with a beta distribution which generates the time stamp for the message. Further details can be found in [17].

There are two key differences between Algorithm B and proposed CD-LDA. CD-LDA first breaks up the datasets into smaller episodes whereas Algrothm-B models each message as an episode and uses prior distributions to model the fact that different events happen at different times. We show that, such an algorithm works, but the inference procedure is dramatically slow due to additional parameters to infer.

4.1.2 Algorithm C: A Graph-clustering based algorithm. For the purposes of comparison, we will also consider a very simple graph-based clustering-based algorithm to identify events. This algorithm is inspired from graph based clustering used in event log data in [19]. The basic idea behind the algorithm is as follows: we construct a graph whose nodes are the messages in the set $\mathcal{M}$. We divide the continuous time interval $[0, T]$ into T/w timeslots, where each timeslot is of duration w. For simplicity, we will assume that T is divisible by w. We draw an edge between a pair of nodes (messages) and label the edge by a distance metric between the messages, which roughly indicates the likelihood with which two messages are generated by the same event. Then, any standard distance-based clustering algorithm on the graphs will cluster the messages into clusters, and one can interpret each cluster as an event. Clearly, the algorithm has the following major limitation: it can detect $\mathcal{M}_e$ for an event e and not $p^{(e)}$. In some applications, this may be sufficient. Therefore, we consider this simple algorithm as a possible candidate algorithm for our real data set.

We now describe how the similarity metric is computed for two nodes i and j. Let N_i be the number of timeslots during which a message i occurs and let N_{ij} be the number of timeslots during which both i and j appear in the same timeslot. Then, the distance metric between nodes i and j is defined as

$$\rho_{ij} = 1 - \frac{N_{ij}}{N_i + N_j - N_{ij}}.$$

Thus, a smaller ρ_{ij} indicates that i and j co-occur frequently. The idea behind choosing this metric is as follows: messages generated by the same event are likely to occur closer together in time. Thus, ρ_{ij} being small indicates that the messages are more likely to have been generated by the same event, and thus are closer together in distance.

4.2 Results: Comparison with Benchmark Algorithms

For the purposes of this section only, we consider a smaller slice of data from Dataset-1. Instead of considering all the 97 million messages, we take a small slice of 10,000 messages over a 3 hour duration from 135 distinct routers. Let us call this data set $\mathcal{D}_s$. There are two reasons for considering this smaller slice. Firstly, it is much easier to visually compare the results from different methods

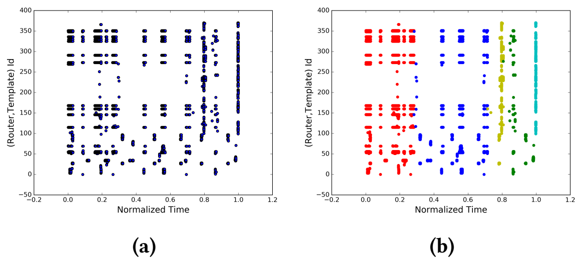

Figure 3: Left panel shows scatter plot of different message-ids over the period of comparison and right panel shows the episodes detected by CD phase of Algorithm CD-LDA.

with a smaller dataset. Since there is no standard metric to compare the results of the different algorithms, visual comparison allows us to interpret the results. Secondly, as we show later in this section, the Bayesian inference Algorithm-B is dramatically slow and so running it over the full dataset is not feasible. Nevertheless, the smaller dataset allows us to validate the key premise behind our main algorithm, i.e., the decomposition of the algorithm into the CD and LDA parts.

Applying CD-LDA on this dataset slice: Figure 3a shows the data points in x-axis and the message-ids on y-axis. Figure 3b shows the 5 episodes after the CD part of CD-LDA, where we chose $\alpha = 0.1$ and $\delta = 0.5$. For the LDA part, instead of specifying the number of events, we use maximum likelihood to find the optimal number of events and based on this, the number of events was found to be 2.

We next compare event signatures produced by CD-LDA with Algorithm B and Algorithm C.

CD-LDA versus Algorithm B: For all unknown distributions, we assume a uniform prior in Algorithm B. Algorithm B is run with input number of events as 2, 3, 4, 5. It turns out that, with 3 events the algorithm converges to a solution which has maximum likelihood. However, upon clustering the event signatures $p^{(e)}$ based on TV-distance between the event signatures, we find only two events. *The maximum TV-distance between the events signatures found from the two algorithms is* 0.068. Hence, we can conclude that the event signatures found by both the algorithms are very similar.

Despite the fact that Algorithm B using fewer hyper-parameters, it is not fast enough to run on large data sets. Figure 4a shows the time taken by CD-LDA and Algorithm B as we increase the size of the data set from 10,000 to 40,000 points. With 40,000 data points and 12 events as input Algorithm B takes 3 hours whereas CD-LDA only takes 26.57 seconds. Clearly, we cannot practically run Algorithm B on large data sets with millions of points.

CD-LDA versus Algorithm C: In this section we compare CD-LDA versus algorithm C on data set $\mathcal{D}_s$. Algorithm C can produce the major event clusters as CD-LDA, but does not provide the start and end time for the events. We form the co-occurrence graph for Algorithm C with edge weight as described in section 4.1.2 and nodes as messages which occur more than at least 5 times in the data set $\mathcal{D}_s$. All the edges with weight more than 0.6 are discarded and we run a clique detection algorithm in the resulting graph. We quantitatively compare the event signature $\mathcal{M}_e$ of the top two cliques found by Algorithm C with those found by CD-LDA. Suppose that message sets identified by Algorithm C for the two events are $\mathcal{M}_{e1}$ and $\mathcal{M}_{e2}$ respectively. Message sets (messages with probability more than 0.007) identified by CD-LDA for the two events are denoted by $\mathcal{S}_{e1}$ and $\mathcal{S}_{e2}$. We can now compute the Jaccard Index between the two sets.

$$\frac{|\mathcal{M}_{e1} \cap \mathcal{S}_{e1}|}{|\mathcal{M}_{e1} \cup \mathcal{S}_{e1}|} = 0.73 \qquad \frac{|\mathcal{M}_{e2} \cap \mathcal{S}_{e2}|}{|\mathcal{M}_{e2} \cup \mathcal{S}_{e2}|} = 0.68.$$

Since the full Bayesian inference (Algorithm B) agrees with CD-LDA closely, we can conclude that Algorithm C gets a large fraction of the messages associated with the event correctly. However, it also misses a significant fraction of the messages, and additionally Algorithm C does not provide any information about start and end times of the events. Also, the events found are sensitive to the threshold for choosing the graph edges, something we have carefully chosen for this small data set.

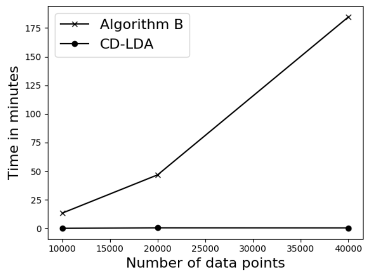

Figure 4: Time performance: CD-LDA vs Algrithm B

Event 1	Event 2	…	Event 8
mmscRuntimeError	ISCSI_multipath		SNMP_sshd
SUDBConnectionDown	Logmon_contrail		SNMP_crond
SocketConnectionDown	VRouter-Vrouter		SNMP_AgentCheck
SUDBConnectionUp	LogFile_nova		SNMP_ntpd
SocketConnectionUp	SUDBConnectionDown		SNMP_CPU
mmscEAIFUnavailable	IPMI		SNMP_Swap
bigipServiceUp	bigipServiceDown		SNMP_Mem
bigipServiceDown	bigipServiceUp		SNMP_Filespace
SNMP_Mem	HW_IPMI		Ping_vm

Table 1: Events generated by CD-LDA and the constituent messages in decreasing order of probability. Event 8 matches with expert provided event signature.

4.3 Results: Comparison with Expert Knowledge and Scalability

Validation by comparing with manual event signature: The intended use-case of our methodology is for learning events where the scale of data and system does not allow for manual identification of event signatures. However, we still wanted to validate our output against a handful of event signatures inferred manually by domain experts. For the purpose of this section, we ran CD-LDA for Dataset-2 which is for an operational VNF. For this data set, an expert had identified that a known service issue had occurred on two dates: 11-Oct and 26-Nov, 2017. This event generated messages with Ids Ping_vm, SNMP_AgentCheck, SNMP_ntpd, SNMP_sshd, SNMP_crond, SNMP_Swap, SNMP_CPU, SNMP_Mem, SNMP_Filespace.

We ran CD-LDA on this data set with parameters $\alpha = 0.01$ and $\delta = 0.1$. We chose 10 events for the LDA phase based on the maximum likelihood of the output. Table 1 shows the events detected by

CD-LDA in decreasing order of probability. Also, top 9 messages are listed for each event. Indeed, we note that *Event* 8 *resembles the expert provided event. CD-LDA detected this event as having occurred from 2017-10-08 17:35 to 2017-10-17 15:55 and 2017-11-25 13:45 to 2017-11-26 03:10.* The longer than usual detection window for 11-Oct is due to the fact that there were other events occurring simultaneously in the network and the Event 8 contributed to small fraction of messages generated during this time window. Finally, as shown in Table 1, our method also discovered several event signatures not previously known.

Scalability: To understand the scalability of CD-LDA with data size, we ran it on Dataset-1 with about 97 million data points. CD-LDA was run with the following input: $\alpha = 0.01$, $\delta = 0.1$, and the number of events equal to 20. The CD part of the algorithm detects 57 change points. The sensitivity of this output with respect to α, δ is discussed in more detail in [17]. Here, we only mention that the event signatures are quite robust to these parameter choice, unless alpha and delta are chosen to be very large, as shown empirically in [17]. Overall, CD-LDA takes about 6 hours to run, which is quite reasonable for a dataset of this size. Reducing the running time by using other methods for implementing LDA, such as variational inference, is a topic for future work.

5 CONCLUSION

In this paper we look at the problem of detecting events in an error log generated by a distributed data center network. The error log consists of error messages with time stamps. Our goal is to detect latent events which generate these messages and find the distribution of messages for each event. We solve this problem by relating it to the topic modelling problem in documents. We introduce a notion of episodes in the time series data which serves as the equivalent of documents. Also we propose a linear time change detection algorithm to detect these episodes. We present consistency and sample complexity results for this change detection algorithm. Further we demonstrate the performance of our algorithm on a real dataset by comparing it with two benchmark algorithms existing in the literature.

REFERENCES

[1] Rakesh Agrawal and Ramakrishnan Srikant. 1995. Mining Sequential Patterns. In *Proceedings of the Eleventh International Conference on Data Engineering (ICDE '95)*. IEEE Computer Society, Washington, DC, USA, 3–14. http://dl.acm.org/citation.cfm?id=645480.655281

[2] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. 2003. Latent Dirichlet Allocation. *J. Mach. Learn. Res.* 3 (March 2003), 993–1022. http://dl.acm.org/citation.cfm?id=944919.944937

[3] Dehua Cheng, Mohammad Taha Bahadori, and Yan Liu. 2014. FBLG: A Simple and Effective Approach for Temporal Dependence Discovery from Time Series Data. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '14)*. ACM, New York, NY, USA, 382–391. https://doi.org/10.1145/2623330.2623709

[4] Thomas L Griffiths and Mark Steyvers. 2004. Finding scientific topics. *Proceedings of the National academy of Sciences* 101, suppl 1 (2004), 5228–5235.

[5] Matthew Hoffman, Francis R Bach, and David M Blei. 2010. Online learning for latent dirichlet allocation. In *advances in neural information processing systems*. 856–864.

[6] Matthew D Hoffman, David M Blei, Chong Wang, and John Paisley. 2013. Stochastic variational inference. *The Journal of Machine Learning Research* 14, 1 (2013), 1303–1347.

[7] Yexi Jiang, Chang-Shing Perng, and Tao Li. 2011. Natural Event Summarization. In *Proceedings of the 20th ACM International Conference on Information and Knowledge Management (CIKM '11)*. ACM, New York, NY, USA, 765–774. https://doi.org/10.1145/2063576.2063688

[8] Yoshinobu Kawahara and Masashi Sugiyama. 2012. Sequential change-point detection based on direct density-ratio estimation. *Statistical Analysis and Data Mining: The ASA Data Science Journal* 5, 2 (2012), 114–127.

[9] Tao Li, Feng Liang, Sheng Ma, and Wei Peng. 2005. An Integrated Framework on Mining Logs Files for Computing System Management. In *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining (KDD '05)*. ACM, New York, NY, USA, 776–781. https://doi.org/10.1145/1081870.1081972

[10] Tao Li, Larisa Shwartz, and Genady Y. Grabarnik. 2017. System Event Mining: Algorithms and Applications. *KDD 2017 Tutorial* (2017). https://users.cs.fiu.edu/~taoli/event-mining/

[11] Tao Li, Chunqiu Zeng, Yexi Jiang, Wubai Zhou, Liang Tang, Zheng Liu, and Yue Huang. 2015. Data-Driven Techniques in Computing System Management. *ACM Comput. Surv.* 50, 3, Article 45 (July 2015), 43 pages. https://doi.org/10.1145/3092697

[12] Alexandre Lung-Yut-Fong, Céline Lévy-Leduc, and Olivier Cappé. 2011. Homogeneity and change-point detection tests for multivariate data using rank statistics. *arXiv preprint arXiv:1107.1971* (2011).

[13] Adetokunbo A.O. Makanju, A. Nur Zincir-Heywood, and Evangelos E. Milios. 2009. Clustering Event Logs Using Iterative Partitioning. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '09)*. ACM, New York, NY, USA, 1255–1264. https://doi.org/10.1145/1557019.1557154

[14] Carl H. Mooney and John F. Roddick. 2013. Sequential Pattern Mining – Approaches and Algorithms. *ACM Comput. Surv.* 45, 2, Article 19 (March 2013), 39 pages. https://doi.org/10.1145/2431211.2431218

[15] Wei Peng, Charles Perng, Tao Li, and Haixun Wang. 2007. Event Summarization for System Management. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '07)*. ACM, New York, NY, USA, 1028–1032. https://doi.org/10.1145/1281192.1281305

[16] David S. Matteson and Nicholas A. James. 2013. A Nonparametric Approach for Multiple Change Point Analysis of Multivariate Data. 109 (06 2013).

[17] Siddhartha Satpathi, Supratim Deb, R Srikant, and He Yan. 2018. Learning Latent Events from Network Message Logs. *arXiv preprint* (2018).

[18] Jonathan A. Silva, Elaine R. Faria, Rodrigo C. Barros, Eduardo R. Hruschka, André C. P. L. F. de Carvalho, and João Gama. 2013. Data Stream Clustering: A Survey. *ACM Comput. Surv.* 46, 1, Article 13 (July 2013), 31 pages. https://doi.org/10.1145/2522968.2522981

[19] E. Sy, S. A. Jacobs, A. Dagnino, and Yu Ding. 2016. Graph-based clustering for detecting frequent patterns in event log data. In *2016 IEEE International Conference on Automation Science and Engineering (CASE)*. 972–977. https://doi.org/10.1109/COASE.2016.7743509

[20] Liang Tang and Tao Li. 2010. LogTree: A framework for generating system events from raw textual logs. In *Data Mining (ICDM), 2010 IEEE 10th International Conference on*. IEEE, 491–500.

[21] Nikolaj Tatti and Jilles Vreeken. 2012. The Long and the Short of It: Summarising Event Sequences with Serial Episodes. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '12)*. ACM, New York, NY, USA, 462–470. https://doi.org/10.1145/2339530.2339606

[22] Chong Wang, David Blei, and David Heckerman. 2012. Continuous time dynamic topic models. *arXiv preprint arXiv:1206.3298* (2012).

[23] Chong Wang, John Paisley, and David Blei. 2011. Online variational inference for the hierarchical Dirichlet process. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. 752–760.

[24] Peng Wang, Haixun Wang, Majin Liu, and Wei Wang. 2010. An Algorithmic Approach to Event Summarization. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data (SIGMOD '10)*. ACM, New York, NY, USA, 183–194. https://doi.org/10.1145/1807167.1807189

[25] Xuerui Wang and Andrew McCallum. 2006. Topics over Time: A non-Markov Continuous-time Model of Topical Trends. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '06)*. ACM, New York, NY, USA, 424–433. https://doi.org/10.1145/1150402.1150450

[26] Fei Wu, Pranay Anchuri, and Zhenhui Li. 2017. Structural Event Detection from Log Messages. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '17)*. ACM, New York, NY, USA, 1175–1184. https://doi.org/10.1145/3097983.3098124

[27] Chunqiu Zeng, Qing Wang, Wentao Wang, Tao Li, and Larisa Shwartz. 2016. Online inference for time-varying temporal dependency discovery from time series. In *Big Data (Big Data), 2016 IEEE International Conference on*. IEEE, 1281–1290.