Structural Event Detection from Log Messages

Fei Wu*
Pennsylvania State University
College of Information Sciences and Technology
fxw133@psu.edu

Pranay Anchuri
NEC Laboratories America
pranay@nec-labs.com

Zhenhui Li
Pennsylvania State University
College of Information Sciences and Technology
jessieli@ist.psu.edu

ABSTRACT

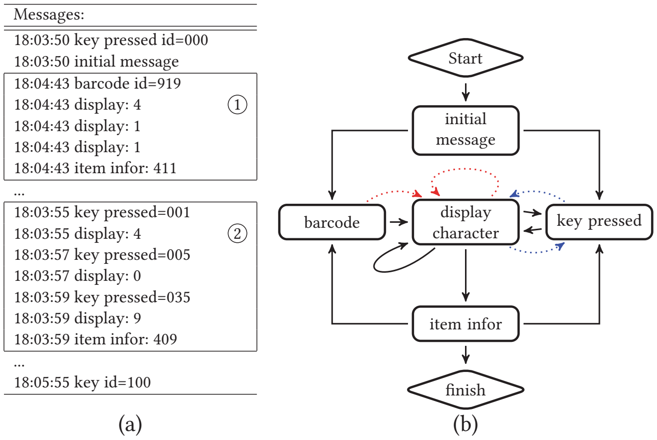
A wide range of modern web applications are only possible because of the composable nature of the web services they are built upon. It is, therefore, often critical to ensure proper functioning of these web services. As often, the server-side of web services is not directly accessible, several log message based analysis have been developed to monitor the status of web services. Existing techniques focus on using clusters of messages (log patterns) to detect important system events. We argue that meaningful system events are often representable by groups of cohesive log messages and the relationships among these groups. We propose a novel method to mine structural events as directed workflow graphs (where nodes represent log patterns, and edges represent relations among patterns). The structural events are inclusive and correspond to interpretable episodes in the system.

The problem is non-trivial due to the nature of log data: (i) Individual log messages contain limited information, and (ii) Log messages in a large scale web system are often interleaved even though the log messages from individual components are ordered. As a result, the patterns and relationships mined directly from the messages and their ordering can be erroneous and unreliable in practice. Our solution is based on the observation that meaningful log patterns and relations often form workflow structures that are connected. Our method directly models the overall quality of structural events. Through both qualitative and quantitative experiments on real world datasets, we demonstrate the effectiveness and the expressiveness of our event detection method.

ACM Reference format:
Fei Wu, Pranay Anchuri, and Zhenhui Li. 2017. Structural Event Detection from Log Messages. In *Proceedings of KDD '17, Halifax, NS, Canada, August 13-17, 2017,* 10 pages.
https://doi.org/10.1145/3097983.3098124

1 INTRODUCTION

In today's connected world, the web applications help with all aspects of life. Most of the modern web applications are served by loosely coupled web services. Enterprises spend great resources to ensure proper functioning of these web services as they directly impact the quality and availability of the applications. At the same time, the ubiquitous logging behavior generates rich text messages that are useful for monitoring the performance of services and identifying their potential risks. However, the sheer amount of messages and its highly dynamic nature render the problem challenging.



```
Messages:
18:03:50 key pressed id=000
18:03:50 initial message
18:04:43 barcode id=919
18:04:43 display: 4                ①
18:04:43 display: 1
18:04:43 display: 1
18:04:43 item infor: 411
...
18:03:55 key pressed=001
18:03:55 display: 4                ②
18:03:57 key pressed=005
18:03:57 display: 0
18:03:59 key pressed=035
18:03:59 display: 9
18:03:59 item infor: 409
...
18:05:55 key id=100
```

(a) (b)

Figure 1: Motivating example: (a) Log messages generated by the Retail Management Service (RMS) at a grocery store. Symbol ① and ② mark the logs corresponding to manual entry and barcode scan events respectively. (b) A structural event detected from the messages. Arrows of the same color represent a event sequence.

To tackle the problem, researchers and businesses have been heavily investigated in mining various system events from logs, such as, log patterns [3, 6, 22], and relations between log patterns [10, 11, 20, 21, 26]. While mined patterns are useful, most studies do not consider the high level structures composed by those patterns. In this paper, we argue that the high level structures often represent more meaningful system events, which can be naturally expressed by directed workflow graphs. We motivate our problem by using log messages collected from a Retail Management Service (RMS)[1]. A RMS is a set of applications used by retailers to manage their business.

Example 1.1. Consider a user shopping in a retail store, the cashier working at the till uses both keyboard and scanner based input methods to enter products purchased by the user. The actions of the cashier are registered as log messages by the store's RMS. Figure 1(a) shows the log messages generated for this transaction. Two

*Work performed while the author was an intern at NEC Labs.

[1] https://blog.springboardretail.com/what-is-a-retail-management-system/

KDD '17, August 13-17, 2017, Halifax, NS, Canada
© 2017 Association for Computing Machinery.
ACM ISBN 978-1-4503-4887-4/17/08…$15.00
https://doi.org/10.1145/3097983.3098124

major actions of this transaction are labeled by system administrators: A) Scanning a product (id=411) using its barcode. B) Product with an id 409 entered via keyboard. We can see that individual log message contains limited semantic information, for example the log at 18:03:55 just shows that the key *4* is pressed. Format of the log messages may indicate some patterns, e.g., *key press* and *display character*, but the patterns are still hard to interpret and does not completely represent the intentions of the cashier.

The key observation here is that the entire event (transaction) is reflected by structures composed of multiple transitions and patterns of logs. Figure 1(b) shows the directed workflow graph composed by the patterns and the transitions. The graph representation associates isolated log patterns into structures that embed semantic information. It is easy to see that left part of the graph corresponds to scanning barcode, and the right part corresponds to manually entering the item code.

The example shows that *important/meaningful system events are revealed by structures spanning multiple log patterns and their transitions*. The directed graph does not merely visualize the intermediate transitions between patterns. More importantly, it reveals structural relations beyond just the pairs of patterns. Therefore, we name such a directed workflow graph a *structural event*, and aim to detect them from logs. Meaningful structural events have shown to be very valuable in various application domains, such as, monitoring system workflow [28], detecting sequence anomalies [19], and program workflow inspection [4, 5, 27].

However, automatically detecting such structural events is a challenging problem, due to characteristics of log data. First, individual log messages contain limited information. For example, in Figure 1, the log at 18:03:55 just shows that the key *4* is pressed. The characteristic raises significant difficulties in detecting meaningful patterns (groups of logs). Second, a large proportion of the messages may be interleaving because of simultaneous task execution in distributed systems, as unique task identifiers may not be available [19, 27]. As a result, the temporal pattern relations mined from the raw data may be inaccurate and misleading. These characteristics require the structural event detection method to intelligently distinguish meaningful relations and patterns from the ones incurred by noise. In the literature, such structural events are extracted in a closed environment [4, 5, 28], where log messages are collected by running each application in isolation with as few background messages as possible. However, such learning process incurs a high cost and has limited usage. In this paper, we take a data-driven approach to detect structural events from noisy log messages directly.

Furthermore, we address the limitation of a workflow graph in expressing higher-order sequential relations. More specifically, in Figure 1, the two major events are reflected by two high order pattern sequences: i) *barcode* → *display* ⟲ marked by the red dashed arrows (a barcode scan followed by display of multiple characters), and ii) *key pressed* ⇄ *display* marked by the blue dashed arrows (each key press directly results in one character display). However, if we only consider the transition expressed by the edges, then the pattern sequence *barcode* → *display* → *key pressed* will be incorrectly considered as a valid transition. The higher-order information is particularly important for differentiating events with common log patterns (nodes).

While literature have been focusing on proposing quality measures for detected patterns and relations, few looked at the workflow graphs resulting from connecting patterns with edges. In this paper, we directly model the quality of the graph. In such an approach, we can not only consider the structure quality of the resulting events, but also account for errors in mined significant patterns and relations. We resort to the intuition that *meaningful log patterns and relations often form workflow structures that are connected*. We formulate our event detection problem as a graph editing task. Our proposed approach starts from a candidate graph containing all the mined patterns then gradually edit the graph (i.e., adding or deleting edges and nodes until a certain energy function is minimized). Intuitively, the structural events should include significant patterns and transitions for the system (i.e., high *precision* and high *coverage)*. More importantly, we favor patterns and relations that are part of connected structures. The latter property translates to graph *connectivity*. We further extend our energy function to embed higher-order transitions and present a block optimization technique to solve the problem.

In summary, our novel contributions are as follows:

- We study an important and challenging problem of detecting structural events from *noisy* log messages.
- We propose a novel data-driven approach that is readily applicable to any system logs and does not require domain knowledge about the system to learn the model.
- We propose an energy minimization formulation that can be solved efficiently. Compared with existing approaches, our proposed energy function better describes important structural events. We further extend our model to account for higher-order relations to eliminate ambiguity caused by edge representation.
- Experiments on three real datasets demonstrate the effectiveness of our approach on different systems.

The rest of the paper is organized as follows. Related studies are first discussed in Section 2. Section 3 presents preliminaries of our detection framework. We describe our approach in Section 4 followed by inference method in Section 5. Section 6 discusses the extension to include higher-order relations. Experimental results is in Section 7. Finally, the paper is concluded in Section 8.

2 RELATED WORK

In this section, we summarize related work from three aspects: i) Node discovery, ii) Dependency discovery, and iii) Model inference.

Node discovery (Log summarization). This line of studies is focused on providing a precise summarization (i.e., clusters) of logs. As traditional clustering methods (e.g., k-means) designed for numerical data are not directly applicable, (as logs are often categorical and textual), researchers have proposed methods to cluster logs by frequent words [24], text templates [3, 6, 19], textual hierarchies [22], and log categories obtained by supervised methods [15]. To further consider the temporal information in the logs, Jiang et al. [10] proposed to look at histograms of transition time between the log messages to find log patterns. The resulting log clusters consider the frequency of log appearances as well as the transition time among logs. Instead of improving the pattern discovery step, our proposal takes a complementary approach to model the quality of the graph. Aforementioned studies can be applied for node discovery in our framework.

Dependency discovery (Log dependency mining). Another line of studies has been focusing on mining dependency relations from the ordering of log messages. Various definitions of temporal dependency have been proposed, such as, forwarding conditional probabilities [20], transition invariants (e.g., A always follow B) [4, 5], and transition significance [12, 13]. These studies mainly focus only on mining reliable pattern relations from the data and do not consider the overall quality of the structural events.

Another long line of studies aims to mine higher-order sequential relations from data. Traditional frequent pattern mining approaches, such as sequential pattern mining [2, 8], frequent episode mining [17, 18] can be applied to find important higher-order relations (sequences). Frequent pattern mining approaches often output large number of sequences with little variation. Many studies further reduce the result redundancy by using minimum description length principle [11, 14, 23], or an interestingness measure [7]. However, the approaches do not consider how to summarize mined sequences into a workflow graph.

Both lines of studies do not consider the quality of the workflow graphs after aggregating mined pattern relations. In contrast, we directly model the characteristics of the global graph. By looking at the global structure, we are able to find transition patterns that are structurally important but may have low quality score locally. Again, our approach and aforementioned relation discovery methods are complementary. Our approach can build upon mined relations and sequences.

Workflow model inference. Beschastnikh et al. [4, 5] proposed a system to generate program execution workflow graphs from log data. The generated graphs are later used in system debugging tools. Yu et al. [28] proposed a system that utilizes pregenerated workflow graphs to monitor interleaved log messages on cloud computing services. Both studies use workflow graphs generated from log messages for monitoring and inspection purposes. Different from our work, the workflow graph generation methods assume that the logs are collected under a closed environment, i.e., log messages are collected by running each application in isolation with as few background messages as possible. Such learning process incurs a high cost and has limited usage in practice. In contrary, we detect structural events (i.e., graph) from noisy log messages directly. Perng et al. [21] also use Event Relation Networks (directed graphs) to represent temporal relations discovered. The graph construction step applies user-specified thresholds to filter insignificant relations. In practice, thresholds are hard to set. We will compare with threshold based methods and depict their problems in the experiments section. Furthermore, aforementioned studies do not consider higher-order sequential relations among patterns.

3 PRELIMINARIES

In this section, the pipeline of our approach is first described in Section 3.1. The process of learning the nodes (Section 3.2) and learning the edges (Section 3.3) is explained.

3.1 Pipeline

Our approach consists of three steps. Given a sequence of n log messages, $M = \langle m_1, m_2, ..., m_n \rangle$, the first step converts raw messages into a stream of log patterns $S = \langle p(s_1), p(s_2), ..., p(s_n) \rangle$, where $p(s_i)$ represents the pattern id of message s_i. We denote the set of all log patterns as $\mathcal{P} = \{p_1, p_2, ..., p_l\}$, where l is the total number of patterns mined. Log messages with similar syntactical structure usually correspond to system events that have the same semantic meaning. For example, in Figure 1(a), messages following the regular expression " $*$ *barcode id* : $*$" ($*$ denotes wild cards characters) correspond to the barcode scanning event. Therefore, we discuss our proposed framework at the pattern level.

We follow [28] to use regular expressions to cluster messages. More specifically, a regular expression tree is built using all the log messages, where different levels of the tree represent regular expressions at different specificity. We use the level where the number of clusters falls into a pre-defined range. Other log message clustering methods can also be applied to find patterns.

We further mine transitional (sequential) relations among the patterns. As a result, we can obtain an initial workflow graph $\mathbf{G}^* = (\mathbf{V}^*, \mathbf{E}^*)$ from the log pattern stream, where each node $v \in \mathbf{V}^*$ represents a log pattern (i.e., a cluster of messages), and each $e \in \mathbf{E}^* \subseteq \mathbf{V}^* \times \mathbf{V}^*$ denotes a temporal relation mined from the pattern stream S. As the initial event graph may contain spurious edges, we seek important substructures that represent the system behavior.

Therefore, our goal is to find:

$$\mathbf{G} = \arg min_{\mathbf{G}^l \subseteq \mathbf{G}^*} E(\mathbf{G}^l),$$

where $\mathbf{G}^l$ is a subgraph of the initial event graph $\mathbf{G}^*$, and function $E()$ measures the quality of the summarized graph. We will discuss the detail of $E()$ in following sections.

3.2 Learning Nodes

In the workflow graph, each node i is associated with a weight $m(\cdot)$ denoting the importance of the log pattern. Formally, we use normalized clustering size as the weight for each node, i.e.,

$$m(i) = \frac{|\{p(s) = p_i | i \in \{1, \ldots, n\}\}|}{n},$$

where $|\cdot|$ is the cardinality of a set, and $m(i) \in [0, 1]$.

Note that other log clustering methods (mentioned in Section 2) can also be applied for discovering meaningful log patterns, and more sophisticated measures can be used. Here, for simplicity, we only consider normalized clustering size as the weight function, as it is not the focus of this paper.

3.3 Learning Edges

To construct edges in the initial workflow graph, each node is connected with its neighbors. Node i, and j are neighbors, if and only if there exists a transition from log pattern p_i to pattern p_j, i.e., $\exists t$ s.t. $s_t.p = p_i \wedge s_{t+1}.p = p_j$, where $p \in \mathcal{P} = \{p_1, p_2, ..., p_l\}$ is the set of all log patterns. The edges are weighed according to a quality measure $q(\cdot)$ quantifying the strength of the relation. Formally, we use the forwarding transitional probability as our edge quality measure $q(\cdot)$, i.e.,

$$q(i, j) = \frac{m(i, j)}{m(i)},$$

where $m(i,j)$ is the number of times transition $p_i \rightarrow p_j$ occurs, i.e., $m(i,j) = |\{\langle s_t.p, s_{t+1}.p\rangle : s_t.p = p_i \wedge s_{t+1}.p = p_j\}|$. Note that we have $q(i,j) \in [0,1]$. Similarly, one may choose to use other formulations for the quality measure $q(\cdot)$ (mentioned in Section 2).

4 STRUCTURAL EVENT DETECTION

Given an initial graph $\mathbf{G}^* = (\mathbf{V}^*, \mathbf{E}^*)$, where $\mathbf{E}^*$ denotes a set of mined pairwise relations, the structural event detection is a graph editing process. The goal is to return a graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ (possibly disconnected) that represents important structural events of the system, where $\mathbf{V} \subseteq \mathbf{V}^*$, and $\mathbf{E} \subseteq \mathbf{E}^*$, and $\mathbf{E}^* \subseteq \mathbf{V}^* \times \mathbf{V}^*$. Intuitively, the resulting graph $\mathbf{G}$ should include significant patterns and transitions of the system (i.e., high *precision* and high *coverage)*. More importantly, as *events often span multiple patterns and their transitions*, we favor resulting structures that are more connected.

The best structural events should therefore minimize the following energy function:

$$E = E_{\mathbf{E}} + E_{\mathbf{V}} + E_{\mathbf{G}},$$

where $E_{\mathbf{V}}$ is a measure for the cost of including node set $\mathbf{V}$, $E_{\mathbf{E}}$ measures the cost of including set of edges $\mathbf{E}$, and $E_{\mathbf{G}}$ is a graph regularization term. We first give our complete energy function as:

$$E(\mathbf{G}) = \underbrace{\lambda_e \sum_{e\in\mathbf{E}} (1 - q(e))}_{\text{precision (edge)}} + \underbrace{\lambda_r \sum_{e\in\mathbf{E}^*\setminus\mathbf{E}} q(e)}_{\text{coverage (edge)}} + \underbrace{\lambda_n \sum_{i\in\mathbf{V}} -m(i)}_{\text{coverage (node)}} + \underbrace{\lambda_c |\mathbf{G}|_d}_{\text{connectivity}}, \tag{1}$$

where λ_e, λ_n, and λ_c are hyper-parameters controlling the effect of different components.

Edge precision and coverage: As we want to include significant pattern relations in detected structural events, we define the energy term on the edges as:

$$E_{\mathbf{E}} = \lambda_e \sum_{e\in\mathbf{E}} (1 - q(e)) + \lambda_r \sum_{e\in\mathbf{E}^*\setminus\mathbf{E}} q(e),$$

where $\mathbf{E}$ is the set of edges in $\mathbf{G}$, $\mathbf{E}^* \setminus \mathbf{E}$ is a set of edges not included.

The edge energy consists of components measuring the precision and the coverage of edges respectively. The edge precision term favors including transition relations that have high strength. The second term favors the case where all strong transitions are also covered in detected events. Without considering the coverage term, adding new edges within already connected components (without introducing new nodes) will not decrease the energy value. As a result, edges forming cyclic structures cannot be detected. For example, as shown in Figure 1, when the cashier manually inputs an item code, the system first registers a key press event and displays the corresponding character. The action corresponds to *key pressed* ⇄ *display* patterns in the structural events. Even though both directions of the edge has similar importance, not considering the coverage on edges will likely to miss either the edge *key pressed* → *display* or the edge *key pressed* ← *display*.

Node coverage: We define node energy to measure the coverage on node as:

$$E_{\mathbf{V}} = -\lambda_n \sum_{i\in\mathbf{V}} m(i),$$

where $m(i)$ measures the fraction of the times log pattern i appears. The energy term favors including log patterns that appear more frequently. Similar formulations to include important nodes in the graphs are also used in other works [12, 13].

Graph connectivity. Our key observation is that *important system events often span multiple patterns and transitions of logs*, the intuition translates to measuring the connectivity of the structural events. We define a term on the resulting graph structure using a graph regularization term as follows

$$E_{\mathbf{G}} = |\mathbf{G}|_d,$$

where $|\mathbf{G}|_d$ is the number of connected components. Other connectivity measures, such as, pairwise node distances, are also applicable and yield similar results. We choose to use the number of connected components for ease of computation. A simple depth-first or breath-first search takes linear time complexity with respect to the number of nodes and edges.

5 ENERGY MINIMIZATION VIA GRAPH EDITING

Our goal is to mine sub-graph structures that minimizes the energy function as in Equation (1). The energy function is not differentiable, as unknowns are discrete variables, and the connectivity term does not have closed form expression. Moreover, we can see that a naive search solution is infeasible because of the exponential number of possible subgraphs. We use a Monte Carlo Markov Chain (MCMC) method [1] to explore the search space more effectively.

5.1 MCMC

In a stochastic optimization approach, algorithms generate a new candidate based on the previous ones. In each candidate generation step, a newly generated candidate is compared to the previous candidate. If the new candidate has a better objective value, it will be accepted as the new solution, otherwise, it will be accepted with a probability proportional to its quality. The sequence of candidates is a Markov Chain. Metropolis-Hasting algorithm [9] approaches the optimal solution using such a Markov Chain.

Metropolis-Hasting algorithm consists of two main steps: the proposal and the acceptance steps. In the proposal step, a new graph configuration $\mathbf{G}'$ is proposed by the function Q. Given the newly proposed configuration, the algorithm decides whether to accept the new configuration with a probability γ defined as follows:

$$\gamma = \min\left[1, \frac{f(\mathbf{G}')}{f(\mathbf{G})} \frac{Q(\mathbf{G};\mathbf{G}')}{Q(\mathbf{G}';\mathbf{G})}\right], \tag{2}$$

where $Q(\mathbf{G}';\mathbf{G})$ is the proposal density function. The algorithm repeats the two steps until a stopping criterion is met. A common definition of $f(\mathbf{G}')$ is:

$$f(\mathbf{G}') = \frac{exp^{-E(\mathbf{G}')/T}}{Z},$$

Algorithm 1 SED($\mathbf{E}^*$, Q, E)

```
Input: Mined relation set $\mathbf{E}^*$, proposal function $Q$, energy function
    $E$.
Output: Structural event graph $\mathbf{G}$
while Stopping criteria not met do
    $\mathbf{G} \leftarrow \mathbf{G}^i$
    Propose $\mathbf{G}' \leftarrow Q(\mathbf{G}'; \mathbf{G})$
    Compute $\gamma(i)$ (Eq.6) with $E(\mathbf{G})$ and $E(\mathbf{G}')$.
    if $U[0,1] < \gamma(i)$ then
        $\mathbf{G}^{i+1} \leftarrow \mathbf{G}'$
    else
        $\mathbf{G}^{i+1} \leftarrow \mathbf{G}$
    end if
end while
return G
```

where $Z = \sum_{\forall \mathbf{G}'} e^{-E(\mathbf{G}')/T}$ is the partition function (i.e., normalizing constant), and T is the temperature parameter. Note that since γ is a ratio, we only need $f(\mathbf{G})$ upto a constant factor. Hence, we do not explicitly compute Z. As we have introduced the basics for a stochastic optimization framework, we now proceed to explain our proposed method in detail.

5.2 Proposal Density Function

While the proposal density function can be an arbitrary one, the choice affects the convergence significantly. In the extreme case, an uniform proposal function will perform no better than doing a naive search. Following [12, 13], the proposal function Q is designed to include modifications of graph edges and is defined as follows:

$$Q(\mathbf{G}'; \mathbf{G}) = \begin{cases} Q_a(\mathbf{G}'; \mathbf{G}), & p = 0.5 \\ Q_d(\mathbf{G}'; \mathbf{G}), & p = 0.5, \end{cases} \tag{3}$$

where Q_a adds an edge $e = i \rightarrow j$ to G with a probability $p_a(e)$ defined as follows:

$$p_a(e) = \frac{exp^{-(1-q(e))}}{\sum_{\forall e' \in \mathbf{E}^* \setminus \mathbf{E}} exp^{-(1-q(e'))}}, \tag{4}$$

and $\mathbf{E}^* \setminus \mathbf{E}$ is the set of edges that are not already in the graph. The intuition is that the edges of higher quality are more likely to be included in the structural event graph.

Q_d deletes one edge $e = i \rightarrow j$ from $\mathbf{G}$ with a probability $p_d(e)$ defined as follows:

$$p_d(e) = \frac{exp^{-q(e)}}{\sum_{\forall e' \in \mathbf{E}} exp^{-q(e')}}, \tag{5}$$

where $\mathbf{E}$ is the list of selected edges. The intuition is that an edge of lower quality is more likely to be deleted from the structural event graph. We do not define proposal functions on nodes, as selections on edges implicitly determines node selection as well.

5.3 Simulated Annealing

Metropolis-Hasting algorithm could suffer from long-mixing time (slow-convergence) because of low acceptance rate. Simulated Annealing adaptively sets the T in the Equation (3) to control the acceptance ratio γ.

Usually, the algorithm starts at a high temperature (a large T), where the distribution of $f(\mathbf{G})$ is closer to a uniform distribution. Later, the temperature gradually reduces according to a cooling schedule. The process corresponds to a broad search at the beginning and gradually narrows down to a promising area for fine-grained exploration.

In this work, we adapt an exponential cooling schedule [1]:

$$T(i) = T_0 exp\{-\alpha i^{1/N}\},$$

where N is the dimensionality of the model space, and we let $N = 2$, $\alpha = 0.8$ and $T_0 = 1$. The new acceptance rate $\gamma(i)$ varies over iterations as follows:

$$\gamma(i) = \min\left[1, \frac{exp^{-E(\mathbf{G}')/T(i)}}{exp^{-E(\mathbf{G})/T(i)}} \frac{Q(\mathbf{G}; \mathbf{G}')}{Q(\mathbf{G}'; \mathbf{G})}\right] \tag{6}$$

The optimization process is presented in Algorithm 1. The algorithm takes an edge set $\mathbf{E}^*$, initial temperature T_0, proposal function Q, and energy function E. While the stopping criterion is not met, the algorithm continues to examine new proposed structural events.

Several possibilities exist for the stopping criterion. Empirically, we found that stopping the algorithm when the energy value remains unchanged for 100 continuous iterations to be most effective.

Finally, we study the time complexity of algorithm 1. In each iteration, computing the graph energy, $E(\mathbf{G})$, is the most expensive operation. It requires the computation of three terms: Edge energy, Node energy and Connectivity, each of which can be computed in linear time using graph traversal algorithms such as Depth-first search. Given N_{max} iterations of the SED algorithm, the time complexity is, therefore, $O((|\mathbf{V}| + |\mathbf{E}^*|) \times N_{max})$.

6 HIGHER-ORDER SEQUENCES

As we discussed in Section 1, the edge formulation can only represent transitions between pairs of patterns. However, the log patterns may inherently embed higher-order sequential relations. We use $\mathbf{E}_k^*$ to denote a set of high-order relations of length k, e.g., we have $\mathbf{E}^* = \mathbf{E}_2^*$, and $\mathbf{E}_3^* = \{(i, j, k)\}$. Similar to the edge case, the higher-order relations are also weighed by a quality measure $q(\cdot)$. Our goal here is to select important high-order relations $\mathbf{E}_k \subseteq \mathbf{E}_k^*$ to enrich the structural event graph. We can similarly define an energy term that measures the precision and coverage of included relations,

$$E_{\mathbf{E}_k} = \lambda'_e \sum_{e \in \mathbf{E}_k} (1 - q(e)) + \lambda'_r \sum_{e \in \mathbf{E}_k^* \setminus \mathbf{E}_k} q(e). \tag{7}$$

We further constraint that sub-relations of a higher-order relation $e \in \mathbf{E}_k$ should be included in the selected edge set $\mathbf{E}$. For example, we have $(i, j, k) \in \mathbf{E}_3 \Leftrightarrow (i, j) \in \mathbf{E}_2 \wedge (j, k) \in \mathbf{E}_2$. Correspondingly, we want the higher-order relations to explain important log patterns and have:

$$E_{\mathbf{V}_k} = -\lambda_n \sum_{i \in \mathbf{V}(\mathbf{E}_k)} m(i),$$

where $\mathbf{V}(\mathbf{E}_k)$ is a set of log patterns (i.e., nodes) that selected higher-order relations. In this paper, we only consider second order relations, i.e., $\mathbf{E}_3$. The generalization to a larger k is straight-forward. Here, we define weights for high-order relations (of order 2) as:

Algorithm 2 BlockSED($\mathbf{E}^*, \mathbf{E}_k^*$)

```
Input: Mined relation set $\mathbf{E}^*, \mathbf{E}_k^*$
Output: Structural event graph $\mathbf{G}$
1: $\mathbf{G}(\mathbf{V}, \mathbf{E}) \leftarrow SED(\mathbf{E}^*, Q, E)$
2: $\mathbf{E}_{filtered}^* \leftarrow \{(i,j,k) : (i,j) \in \mathbf{E} \wedge (j,k) \in \mathbf{E}, (i,j,k) \in \mathbf{E}_k^*\}$
3: $\mathbf{G}(\mathbf{V}, \mathbf{E}, \mathbf{E}_k) \leftarrow SED(\mathbf{E}_{filtered}^*, H, E)$
4: return $\mathbf{G}(\mathbf{V}, \mathbf{E}, \mathbf{E}_k)$
```

$$q(i,j,k) = q((i,j),k) \times q(i,(j,k)) \quad = \frac{m(i,j,k)}{m(i,j)} \frac{m(i,j,k)}{m(j,k)},$$

where $m(i,j,k)$ is the frequency of transition $i \rightarrow j \rightarrow k$. The energy terms related to higher-order sequences are $E_{\mathbf{E}_k}$ and $E_{\mathbf{V}_k}$. The higher-order energy $E(\mathbf{G}_k)$ is defined as follows:

$$E(\mathbf{G}_k) = E_{\mathbf{E}_k} + E_{\mathbf{V}_k}. \tag{8}$$

Accordingly, the joint energy function is given by:

$$E = E(\mathbf{G}) + E(\mathbf{G}_k), \tag{9}$$

where $E(\mathbf{G})$ is defined as in Equation 1.

6.1 Block Optimization

To optimize the new energy function, we again use a MCMC approach with a proposal function H defined as:

$$H(\mathbf{G}'; \mathbf{G}) = \begin{cases} H_a(\mathbf{G}'; \mathbf{G}), & p = 0.5 \\ H_d(\mathbf{G}'; \mathbf{G}), & p = 0.5 \end{cases}, \tag{10}$$

where H is similar to the function Q defined in Section 5.2 with H_a and H_d representing addition and deletion operations. We can still use Equation 4 and Equation 5 to define editing probabilities, by replacing $\mathbf{E}^*$ and $\mathbf{E}$ with high-order set $\mathbf{E}_k^*$, and $\mathbf{E}_k$ respectively.

However, the minimization problem is easily stuck at some local optima, as we will show in Section 7. To address this problem, we propose a block optimization technique, where we optimize for each order of the relation in an increasing order. A key observation is that the proposal step on high-order relations will not change the energy terms computed on lower-order relations. The detailed steps are shown in Algorithm 2. In line 1, we execute the *SED* algorithm only using the proposal function related to pairwise edge update, i.e., Q. Based on the result, we filter the set of high-order sequences in line 2. In line 3, we again run the *SED* algorithm with the proposal function H. The graph $\mathbf{G}$ with selected edges ($\mathbf{E}$) and the higher-order sequences ($\mathbf{E}_k$) is the structural event graph.

7 EXPERIMENTS

In this section, we performed experiments on log messages collected from three different domains: back-end servers, management systems, and user applications. Results consistently show that our method outperforms various other approaches. Our qualitative results are backed by user studies and case studies.

7.1 Datasets

For all three datasets, we generate ground truth workflow graphs on labeled data, which simulates a perfectly closed environment. The labeled data was provided by domain experts different from the users participated in user study for Windows Server and RMS datasets. For the Web Browser dataset, we separate the logs by user id (as the unique identifier is presented in the dataset) and manually generate workflow subgraphs.

Log Source	# messages	# patterns	# labels
Windows Server	61,190	140	12
RMS	21,736	106	10
Web Browser	997,176	26	11

Table 1: Statistics of the datasets. #labels column shows the number of labeled patterns we have for each dataset, i.e., number of patterns in the ground truth structural event.

Windows Server[2]**.** The Windows server data consists of log messages from a Windows server at a data center. The log messages are collected over a two-month period. The server primarily runs two types of services: (i) database back-up services, and (ii) log-collection processes for the data center. The back-up services are automatically invoked periodically and the log-collection processes are invoked by user requests. As we do not force the server to run under a closed-environment, large amount of the logs are irrelevant to the two services. We manually labeled the log data for these two types of services.

Retail Management Service (RMS)[2]**.** The RMS data consists of log messages from a retail management system. The log messages are collected over a one-month period and has 21,736 messages in total. Domain experts have provided us with expected events during a normal operations of the RMS. These include events corresponding to product scanning, which we use for comparison. The ground truth graph contains 10 log patterns.

Web Browser. The web browser dataset consists of log messages generated from a Firefox browser on a computer for one week[3]. The dataset contains $997,176$ messages. Each log message is associated with an event code reflecting the corresponding browser event, e.g., loading plugins, opening tabs, or allocating memory. We manually label log messages that correspond to common browsing actions: open/close tab, add/delete/move bookmark, follow links, and install plugin. We generate a ground truth workflow graph from the labeled data.

Table 1 summarizes the statistics of the datasets. In each case, the ground truth only describes a fraction of the system functionality, i.e., there may exist other meaningful log patterns and pattern transitions that are not included in the ground truth. Therefore, we only consider log patterns that are included in the ground truth and evaluate the structure induced by those selected patterns.

7.2 Evaluation Metrics

The output of our problem is a directed graph $\mathbf{G} = (\mathbf{V}, \mathbf{E})$. Therefore, we evaluate the result based on similarity between resulting graph and the ground truth graph. Specifically, we use precision and recall of the edges as the metric (measures on the nodes give similar results). Given a ground truth graph $\mathbf{G}_g = (\mathbf{V}_g, \mathbf{E}_g)$, precision

[2] Names are not revealed due to non-disclosure issues.

[3] https://datahub.io/dataset/a-week-in-the-life-of-a-browser-version-2

measures the fraction of edges in **G** that are also in the ground truth graph, i.e., $P = \frac{|\mathbf{E} \bigcap \mathbf{E}_g|}{|\mathbf{E}|}$. Recall measures the fraction of edges in the ground truth graph that are recovered in the result graph **G**, i.e., $R = \frac{|\mathbf{E} \bigcap \mathbf{E}_g|}{|\mathbf{E}_g|}$. We also report F_1 score that considers both precision and recall, i.e., $F_1 = 2\frac{PR}{P+R}$. We only report the precision and recall for the edge set **E**.

7.3 Comparisons

In this paper, we compare our method against four state-of-the-art and baseline methods that extract structural events.

Threshold method. In this method, structural events are detected from an initial workflow graph by simply filtering out all edges with $q(e) < \theta$, θ is a threshold parameter. We use two thresholds 0.1 and 0.5 for comparison. The threshold method considers only the quality of each relation.

StoryLine. Wang et al. [25] and Lin et al. [16] proposed a story line extraction method for summarizing progressing news events. Given a text query, a subgraph is retrieved based on the textual similarity between the query and the documents. In this subgraph, each node represents a text document and each directed edge represents the similarity between documents (with temporal ordering). Each node is also weighed by its dissimilarity to the query. StoryLine extracts minimum weight dominating set of the subgraph and searches for a directed Steiner tree that connects nodes in the set. We use $1 - m(i)$ as the weight for log pattern (node) i and directly use the log patterns appeared in the ground truth as the retrieved subgraph. The method can extract tree like events.

K-cores. We compare with a purely connectivity based detection method. K-cores of a graph are maximally connected subgraphs in which each vertex has degree more or equal to k. We set $K = 3$. The K-cores represents densely connected components of the graph. We further filter edges with quality lower than 0.1. This baseline considers the connectivity of resulting structural events.

ESRE. Kwon and Lee [12, 13] proposed a unified event summarization and detection framework (ESRE). ESRE aims to detect sequential events, such as, a person getting on a bus and sitting, from surveillance videos. The proposed approach first extracts important image segments from video frames. Image segments are connected based on their temporal and spatial proximity. The images segments and their connections are fed into a graph editing algorithm to mine causal events via minimizing an energy function. Compared with our energy function, their energy function does not consider the connectivity and coverage of the resulting graph. As a result, the method is likely to miss important cyclic structures and split complete structural events into smaller ones. We compare our method with the graph-editing step of ESRE.

7.4 Performance on Real Datasets

In this section, we report the performance of compared methods on all three datasets. Table 2 summarizes the results of all compared methods. We can see that our Structural Event Detection SED method achieves the best F_1 score compared against other methods, i.e., 0.9, 1 and 0.86 on Server, RMS, and Browser datasets respectively.

		Threshold ($\theta = 0.1$)	Threshold ($\theta = 0.5$)	StoryLine	K-cores	ESRE	SED
Server	P	0.76	0.82	0.33	0.46	1	0.87
	R	0.82	0.64	0.28	0.93	0.5	0.93
	F1	0.33	0.72	0.31	0.61	0.67	**0.9**
RMS	P	0.8	1	0.75	0.72	1	1
	R	1	0.75	0.37	1	0.25	1
	F1	0.88	0.86	0.5	0.84	0.4	**1**
Browser	P	0.67	0.77	0.3	0.18	0.83	0.75
	R	1	0.83	0.25	1	0.41	1
	F1	0.8	0.8	0.27	0.31	0.56	**0.86**

Table 2: Precision, recall and F-1 scores for compared method on the three datasets respectively.

By varying the threshold from 0.1 to 0.5 in threshold, the precision increases by nearly 0.1 across the three datasets but at the same time, the recall decreases by nearly 0.3. This depicts the problem of a threshold based method. While a higher threshold keeps edges having higher quality, many edges in the complete events may be missed. With a lower threshold, edges of complete events may all be included, however, many incorrect relations will also be included. A precise threshold value is hard to know, and even non-existent. In our approach, such a trade-off is measured based on the contribution of an edge to the overall quality instead.

StoryLine has F_1 score no more than 0.5 across the datasets, as the method explicitly assumes a tree structure connecting important nodes. However, structural events often contain cyclic strictures as illustrated in Figure 1. Both major events, i.e., scanning barcode, and input item code, contain cyclic structures of log patterns.

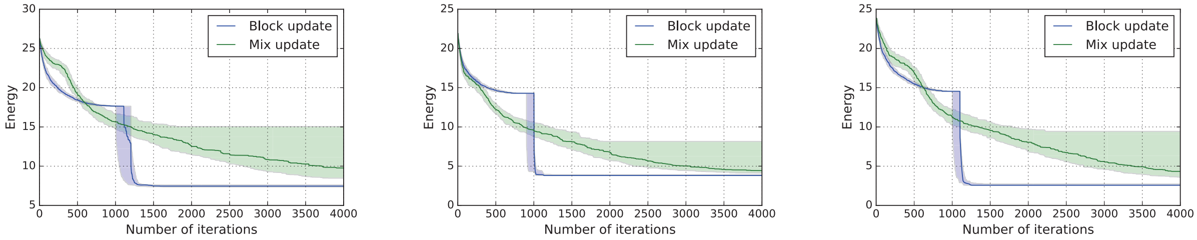
ESRE achieves the best precision, i.e., 1, 1, and 0.83 precision on the three datasets respectively. However the recall values are low, i.e., 0.5, 0.4, and 0.41 on the three datasets respectively. This is because the energy function does not consider coverage of the edges in the result. Adding new edges within already connected components (does not introduce new node) will not decrease the energy value. As a result, edges forming cyclic structures cannot be detected. Furthermore, the energy function does consider the connectivity of the graph. Therefore, edges connecting important sub-structures (while may appear infrequently) will be missed.

K-cores method achieves high recall, i.e., 0.93, 1 and 1 on all the three datasets. However, the precision is low as it purely focuses on the connectivity of the resulting model.

The experiment shows that our proposed method performs the best as it considers precision, coverage, and connectivity of the resulting graph jointly.

7.5 Convergence of Block Optimization

In this section, we study the convergence of our SED Algorithm 2. We compare our block update strategy with vanilla stimulated annealing approach (i.e., mix update), where we use the following

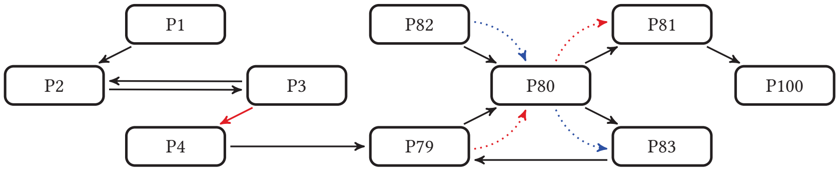


(a) Windows Server

(b) RMS

(c) Browser

Figure 2: Energy value with respect to number of iterations for alternating update and mix update.



(a)

Pattern id	Labeled semantics
P1	Starting input item code (action 1)
P2	Key pressed
P3	Character display
P4	Complete input
P79	Get item property
P80	Display unique item id
P81, P100	Display detailed item information
P83	Repeat
P82	Quantity

(b)

Figure 3: One structural event detected from RMS data. The event corresponds to the cashier inputs an item manually via keyboard (a) Structural event detected, where each node represents log patterns. (b) Semantics for each log pattern.

proposal function Q':

$$Q'(\mathrm{G}';\mathrm{G}) = \begin{cases} Q(\mathrm{G}';\mathrm{G}), & p = 0.5 \\ H(\mathrm{G}';\mathrm{G}), & p = 0.5 \end{cases}.$$

There is an equal chance for a high-order update and an edge update operation to happen. Figure 2 shows the energy value with respect to the number of iterations for both inference approaches on three datasets for 100 runs. The solid line represents the median energy value, and the color bands mark the runs between the first and the third quantile. We can see that block-update approach reaches convergence at iterations 1500, 1100, and 1200 for Windows Server, RMS and Web Browser datasets respectively, while the mixed approach needs about 4000 iterations to converge on the three datasets. At the same time, our proposed approach reaches a lower energy state compared against the mix update approach.

Furthermore, we can see that these results of mixed update approach are unstable as the first and the third quantile cover a large area. These results suggest that the update is easily stuck at some ill-posed local optima. This is because once an ill-posed update gets accepted, it is very hard for the algorithm to undo the step after a few edge updates have occurred. Therefore, ill-posed higher-order updates occuring at the early iterations of the methods would affect the results significantly. The large variation in the result of the vanilla stimulated annealing makes the method impractical.

7.6 User Study on Higher-Order Relations

To evaluate the interpretability of resulting structural events with higher-order relations, we conducted a user study where 19 users were asked to rank the outputs from different methods. The user group is composed of 9 graduate students (majoring in computer science or related fields) and 10 domain experts. BlockSED is used as our method, as we also show the higher-order relations in the detected structural events. For browser data, we asked the users to rank the models based on whether the resulting models reflects normal browsing behavior. For server data, we inform the subjects that the server periodically runs back-up services and collects logs. We asked the users to mark the results that best reflects the two major events. For each user, models from five methods are shown. The method ranked at the best will gain two points and the method ranked at the second gains one point. Table 3 summarizes the user rating normalized by the maximum score a model can achieve. Events detected by SED are consistently ranked either as the first or the second. As a result, SED achieves better user rating on the datasets.

	BlockSED	ESRE	K-cores	StoryLine	Threshold
Server	**0.42**	0.08	0.37	0.37	0.2
Browser	**0.56**	0.23	0.29	0.08	0.5

Table 3: User ratings of compared methods.

7.7 Parameter Study

In this section, we study the effect of the four parameters: $\lambda_e, \lambda_r, \lambda_n$ and λ_c on the energy function given by the equation 1 and describe a process for tuning these parameters. For simplicity, we assume that all these parameters lie in the range $[0, 1]$.

Edge parameters λ_e and λ_r: We first derive a condition under which include an edge, e, when minimizing the graph energy. From

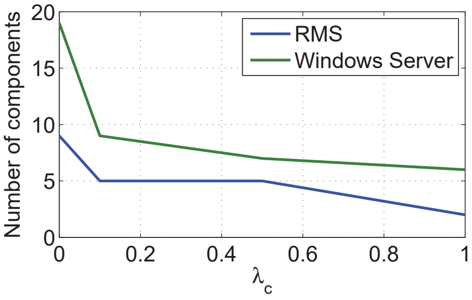


Figure 4: The number of components in the resulting graph with respect to different values of λ_c.

Equation 1 we can see that the net increase in energy by including the edge e is given by the Equation 11.

$$\delta(e) = \lambda_e \times (1 - q(e)) - \lambda_r \times q(e) \tag{11}$$

Since our objective is to minimize the energy, we want $\delta(e) < 0$. Therefore, we include an edge when $q(e) > \frac{\lambda_e}{\lambda_e + \lambda_r}$. This inequality serves as a guideline for choosing λ_e and λ_r based on empirical knowledge. Note that edges having $q(e) \leq \frac{\lambda_e}{\lambda_e + \lambda_r}$ may still be included. In our experiments, we let $\lambda_e = 0.3$ and $\lambda_r = 0.7$.

Node parameter λ_n: We found that the values of $\lambda_n \in [0, 1]$ do not affect the result for our datasets, as the selection of nodes is also implicitly considered in $E_{\mathbf{E}}$.

Connectivity parameter λ_c: We ran experiments on RMS and Windows Server datasets since they have a higher number of patterns as the Table 1 indicates. Figure 4 shows the number of components in the resulting event graph for different values of λ_c. We can see that when $\lambda_c = 0$ (without the connectivity constraints) the event graph is split into 9 and 19 disconnected components in the two datasets. Moreover the number of components vary less (6 to 2 and 9 to 6) as λ_c increases from 0.1 to 1. These results suggests that the detected events are not sensitive to the value of parameter λ_c.

7.8 Case Study

In this section, we perform qualitative analysis on the event detected in RMS data. We show that our model performs the best in unraveling the underlying event. Figure 3(a) shows the event detected by the algorithm 2. The raw logs are first clustered into log patterns using regular expressions. The semantics for patterns are shown in Figure 3(b). The entire structural event describes the message flow when the cashier inputs an item manually via keyboard.

Pattern $P1, P2, P3$ and $P4$ represents logs generated by pressing keys. Whenever a key is pressed, the corresponding character will be displayed on the screen. Therefore, we see a loop between pattern $P2$ and $P3$. The bidirectional transitions between $P2$ and $P3$ happen frequently. ESRE method is likely to miss either transition from $P2$ to $P3$ or from $P3$ to $P2$, as it does not consider the coverage of relations in the energy function. At the same time, StoryLine method cannot detect the loop structure, as it assumes that the progression of news events follows a tree structure. Moreover, compared to $P3 \rightarrow P2$, the transition $P3 \rightarrow P4$ happens far less frequently, as multiple keys need to be pressed to input an item. Threshold based method can easily miss transition $P3 \rightarrow P4$, as it is relatively infrequent. One may lower the threshold to include the transition. But, many irrelevant transitions will also be included as a side effect. Our proposed method can correctly include this transition by considering the connectivity of the graph.

Starting from the pattern $P79$, the rest of the structural event describes the message flow corresponding to displaying behavior of the system. The message flow after entering an item code should be $P79 \rightarrow P80 \rightarrow P81$ and then to $P100$. At the same time, $P82$ represents another action in the system that leads to displaying behavior (patterns leading to $P82$ are not shown for brevity), which generates message flow $P82 \rightarrow P80 \rightarrow P83$. If we only consider transitions between two patterns, $P80 \rightarrow P81$ and $P80 \rightarrow P83$ are both valid, which should not be the case. The contextual information (whether $P80$ is preceded by $P82$ or $P79$) is extremely important in anomaly detection applications. The dashed lines in Figure 3 represent the results of high-order constraints. Compared to all other methods, our proposed framework can easily incorporate the high-order information.

8 CONCLUSIONS AND FUTURE WORK

In this paper, we propose to mine structural events from log messages. The structural events are useful for status monitoring and detecting abnormal behavior sequences. We proposed a data driven approach that can be readily applied on normal system running logs (as oppose to logs generated under a closed environment). Our framework models the quality of the graph structure and embeds higher-order sequential relations. Our proposed framework can be further extended in the following directions.

The structural events can embed more temporal information and consider more sophisticated structures. In this paper, we only utilize the ordering information in log messages. Considering more fine-grained temporal information, e.g., the transition time distribution, can enrich mined structural events. We also focus on transition relations among log patterns in this paper. There are other useful relations among logs, such as running in parallel. Those relations can be further modeled in the workflow graph using undirected edges.

The current method requires manually tuning several hyper-parameters. We plan to reduce the number of parameters, as the energy terms on coverage have similar effects on the results (as shown in Section 7.7). We also believe that the proposed framework can achieve more utility in an interactive setting, where system admins can interactively exploring the system behaviors with different focuses (parameter settings) on coverage, quality or connectivity. This requires our framework to respond to different parameter settings in a timely manner. We plan to investigate how to effectively support the interactive setting with online inference in future work.

ACKNOWLEDGEMENTS

Zhenhui Li and Fei Wu were supported in part by NSF awards #1618448, #1652525, #1639150, and #1544455. The views and conclusions contained in this paper are those of the authors and should not be interpreted as representing any funding agencies.

REFERENCES

[1] E. Aarts and J. Korst. Simulated annealing and boltzmann machines. 1988.

[2] C. C. Aggarwal and J. Han. *Frequent pattern mining*. Springer, 2014.

[3] M. Aharon, G. Barash, I. Cohen, and E. Mordechai. One graph is worth a thousand logs: Uncovering hidden structures in massive system event logs. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 227–243. Springer, 2009.

[4] I. Beschastnikh, J. Abrahamson, Y. Brun, and M. D. Ernst. Synoptic: Studying logged behavior with inferred models. In *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, pages 448–451. ACM, 2011.

[5] I. Beschastnikh, Y. Brun, S. Schneider, M. Sloan, and M. D. Ernst. Leveraging existing instrumentation to automatically infer invariant-constrained models. In *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, pages 267–277. ACM, 2011.

[6] I. Cohen, M. Aharon, E. Mordechai, and G. Barash. Message clustering of system event logs, June 26 2012. US Patent 8,209,567.

[7] J. Fowkes and C. Sutton. A subsequence interleaving model for sequential pattern mining. *arXiv preprint arXiv:1602.05012*, 2016.

[8] J. Han, J. Pei, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, and M. Hsu. Prefixspan: Mining sequential patterns efficiently by prefix-projected pattern growth. In *proceedings of the 17th international conference on data engineering*, pages 215–224, 2001.

[9] W. K. Hastings. Monte carlo sampling methods using markov chains and their applications. *Biometrika*, 57(1):97–109, 1970.

[10] Y. Jiang, C.-S. Perng, and T. Li. Natural event summarization. In *Proceedings of the 20th ACM international conference on Information and knowledge management*, pages 765–774. ACM, 2011.

[11] J. Kiernan and E. Terzi. Constructing comprehensive summaries of large event sequences. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 3(4):21, 2009.

[12] J. Kwon and K. M. Lee. A unified framework for event summarization and rare event detection. In *CVPR*, pages 1266–1273, 2012.

[13] J. Kwon and K. M. Lee. A unified framework for event summarization and rare event detection from multiple views. *IEEE transactions on pattern analysis and machine intelligence*, 37(9):1737–1750, 2015.

[14] H. T. Lam, F. Mörchen, D. Fradkin, and T. Calders. Mining compressing sequential patterns. *Statistical Analysis and Data Mining*, 7(1):34–52, 2014.

[15] T. Li, F. Liang, S. Ma, and W. Peng. 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*, pages 776–781. ACM, 2005.

[16] C. Lin, C. Lin, J. Li, D. Wang, Y. Chen, and T. Li. Generating event storylines from microblogs. In *Proceedings of the 21st ACM international conference on Information and knowledge management*, pages 175–184. ACM, 2012.

[17] H. Mannila and M. Salmenkivi. Finding simple intensity descriptions from event sequence data. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 341–346. ACM, 2001.

[18] H. Mannila, H. Toivonen, and A. I. Verkamo. Discovery of frequent episodes in event sequences. *Data mining and knowledge discovery*, 1(3):259–289, 1997.

[19] A. Nandi, A. Mandal, S. Atreja, Dasgupta, G. B., and S. Bhattacharya. Anomaly detection using program control flow graph mining from execution logs. In *SIGKDD'16*.

[20] W. Peng, C. Perng, T. Li, and H. Wang. Event summarization for system management. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1028–1032. ACM, 2007.

[21] 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.

[22] L. Tang and T. Li. Logtree: A framework for generating system events from raw textual logs. In *2010 IEEE International Conference on Data Mining*, pages 491–500. IEEE, 2010.

[23] N. Tatti and J. Vreeken. 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*, pages 462–470. ACM, 2012.

[24] R. Vaarandi et al. A data clustering algorithm for mining patterns from event logs. In *Proceedings of the 2003 IEEE Workshop on IP Operations and Management (IPOM)*, pages 119–126, 2003.

[25] D. Wang, T. Li, and M. Ogihara. Generating pictorial storylines via minimum-weight connected dominating set approximation in multi-view graphs. In *AAAI*. Citeseer, 2012.

[26] P. Wang, H. Wang, M. Liu, and W. Wang. An algorithmic approach to event summarization. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 183–194. ACM, 2010.

[27] G. Xiao, J. Wang, P. Liu, J. Ming, and D. Wu. Program-object level data flow analysis with applications to data leakage and contamination forensics. In *Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy*, pages 277–284. ACM, 2016.

[28] X. Yu, P. Joshi, J. Xu, G. Jin, H. Zhang, and G. Jiang. Cloudseer: Workflow monitoring of cloud infrastructures via interleaved logs. In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 489–502. ACM, 2016.