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:

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.

To tackle the problem, researchers and businesses have been heavily investigated in mining various system events from logs, such as, log patterns [5, 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)\(^\dagger\). 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

\(\dagger\)https://blog.springboardretail.com/what-is-a-retail-management-system/
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 $\rightarrow$ display $\odot$ marked by the red dashed arrows (a barcode scan followed by display of multiple characters), and ii) key pressed $\Rightarrow$ 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 $\rightarrow$ display $\rightarrow$ 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 approach 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 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 pre-generated 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, \( M = \{m_1, m_2, ..., m_n\} \), the first step converts raw messages into a stream of log patterns \( S = \{p(s_1), p(s_2), ..., p(s_m)\} \), where \( p(s_i) \) represents the pattern id of message \( s_i \). We denote the set of all log patterns as \( \mathcal{P} = \{p_1, p_2, ..., p_t\} \), where \( t \) 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 : s*" (s* 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 \( G^* = (V^*, E^*) \) from the log pattern stream, where each node \( v \in V^* \) represents a log pattern (i.e., a cluster of messages), and each \( e \in E^* \subset V^* \times V^* \) 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:

\[
G = \text{arg min}_{G' \subseteq G^*} E(G'),
\]

where \( G' \) is a subgraph of the initial event graph \( G^* \), and function \( E(\cdot) \) measures the quality of the summarized graph. We will discuss the detail of \( E(\cdot) \) 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.,

\[
m(i) = \frac{\left| \{p(s) = p_i \mid i \in [1, \ldots, n]\} \right|}{n},
\]

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.,

\[
\exists \tau \text{ s.t. } p = p_i \land \tau_{i+1} = p_j, \text{ where } p \in \mathcal{P} = \{p_1, p_2, ..., p_t\} \text{ is the set of all log patterns.}
\]

The edges are weighted according to a quality measure \( q(\cdot, \cdot) \) quantifying the strength of the relation. Formally, we use the forwarding transitional probability as our edge quality measure \( q(\cdot, \cdot) \), 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., \(m(i, j) = \{s, r, s_{i+1}, p) : s, r, p = p_i \land s_{i+1}, p = p_j\}\). 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 \(G^* = (V^*, E^*)\), where \(E^*\) denotes a set of mixed pairwise relations, the structural event detection is a graph editing process. The goal is to return a graph \(G = (V, E)\) (possibly disconnected) that represents important structural events of the system, where \(V \subseteq V^*\), and \(E \subseteq E^*\), and \(E^* \subseteq V^* \times V^*\). Intuitively, the resulting graph \(G\) 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:

\[
E = E_E + E_V + E_G.
\]

where \(E_V\) is a measure for the cost of including node set \(V\), \(E_E\) measures the cost of including set of edges \(E\), and \(E_G\) is a graph regularization term. We first give our complete energy function as:

\[
E(G) = \lambda_e \sum_{e \in E} (1 - q(e)) + \lambda_r \sum_{e \in E^* \setminus E} q(e) + \lambda_\lambda \sum_{i \in V} \left| E_i \right|\]  

\[+ \lambda_d \sum_{i \in V} -m(i) + \lambda_c |G|_d, \quad (1)\]

where \(\lambda_e, \lambda_r, \lambda_\lambda, \lambda_d, \lambda_c\) 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:

\[
E_E = \lambda_e \sum_{e \in E} (1 - q(e)) + \lambda_r \sum_{e \in E^* \setminus E} q(e),
\]

where \(E\) is the set of edges in \(G\), \(E^* \setminus E\) 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** \(\rightarrow\) **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** \(\rightarrow\) **display** or the edge **key pressed** \(\rightarrow\) **display**.

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

\[
E_V = \lambda_d \sum_{i \in V} m(i).
\]

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

\[
E_G = |G|_d,
\]

where \(|G|_d\) 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 breadth-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 \(G'\) is proposed by the function \(Q\). Given the newly proposed configuration, the algorithm decides whether to accept the new configuration with a probability \(\gamma\) defined as follows:

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

(2)

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

\[
f(G') = \frac{\exp^{-E(G')/T}}{Z}.
\]
\textbf{Algorithm 1} SED($E^*$, $Q$, $E$)

\textbf{Input:} Mined relation set $E^*$, proposal function $Q$, energy function $E$.

\textbf{Output:} Structural event graph $G$

\begin{verbatim}
1: while Stopping criteria not met do
2:     G ← G
3:     Propose $G' ← Q(G'; G)$
4:     Compute $y(i)$ (Eq.6) with $E(G)$ and $E(G')$.
5:     if $U[0, 1] < y(i)$ then
6:         $G^{i+1} ← G'$
7:     else
8:         $G^{i+1} ← G$
9:     end if
10: end while
11: return G
\end{verbatim}

where $Z = \sum_{G'} e^{-E(G')/T}$ is the partition function (i.e., normalizing constant), and $T$ is the temperature parameter. Note that since $y$ is a ratio, we only need $f(G)$ up to 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:

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

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

$$p_d(e) = \frac{\exp^{-q(e)}}{\sum_{e' \in E \setminus E^*} \exp^{-q(e')}}$$ \quad (4)

and $E^* \setminus E$ 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 $G$ with a probability $p_d(e)$ defined as follows:

$$p_d(e) = \frac{\exp^{-q(e)}}{\sum_{e' \in E} \exp^{-q(e')}}$$ \quad (5)

where $E$ 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 $y$.

Usually, the algorithm starts at a high temperature (a large $T$), where the distribution of $f(G)$ 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]:

$$T(i) = T_0 \exp(-\alpha i/|N|),$$

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 $y(i)$ varies over iterations as follows:

$$y(i) = \min \left[ 1, \frac{\exp\frac{-E(G')}{T(i)} Q(G, G')}{\exp\frac{-E(G)}{T(i)} Q(G, G')} \right]$$ \quad (6)

The optimization process is presented in Algorithm 1. The algorithm takes an edge set $E^*$, 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, $E(G)$, 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, $O((|V| + |E^*|) \times N_{max})$.

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 $E^*_k$ to denote a set of high-order relations of length $k$, e.g., we have $E^*_1 = E^*_2$, and $E^*_k = \{(i, j, k)\}$. 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 $E_k \subseteq E^*_k$ to enrich the structural event graph. We can similarly define an energy term that measures the precision and coverage of included relations,

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

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

$$E_{V_k} = -\lambda_m \sum_{i \in V(E_k)} m(i),$$

where $V(E_k)$ 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., $E_3$. The generalization to a larger $k$ is straightforward. Here, we define weights for high-order relations (of order 2) as:
Algorithm 2 BlockSED(E\textsuperscript{′}, E\textsubscript{k}\textsuperscript{′})

Input: Mined relation set E\textsuperscript{′}, E\textsubscript{k}\textsuperscript{′}
Output: Structural event graph G
1. G(V, E) ← SED(E\textsuperscript{′}, Q, E)
2. E\textsubscript{filtered} ← \{(i, j, k) : (i, j) ∈ E ∧ (j, k) ∈ E(i, j, k) ∈ E\textsubscript{k}′\}
3. G(V, E, E\textsubscript{k}) ← SED(E\textsubscript{filtered}, H, E)
4. return G(V, E, E\textsubscript{k})

\[ q(i, j, k) = q((i, j), k) \times q(i, (j, k)) = \frac{m(i, j, k) \times m(i, j)}{m(i, j)} \]

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

\[ E(G_k) = E_{E_k} + E_{V_k}. \] (8)

Accordingly, the joint energy function is given by:

\[ E = E(G) + E(G_k), \] (9)

where \( E(G) \) 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:

\[ H(G'; G) = \begin{cases} H_a(G'; G), & p = 0.5 \\ H_d(G'; G), & p = 0.5 \end{cases} \] (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 \( E' \) and \( E \) with high-order set \( E_k' \) and \( E_k \) 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 \( G \) with selected edges \( E \) and the higher-order sequences \( E_k \) 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.

**Windows Server**\textsuperscript{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)**\textsuperscript{3}. 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 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\textsuperscript{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 \( G = (V, E) \). 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 \( G_T = (V_T, E_T) \), precision

\footnotesize

<table>
<thead>
<tr>
<th>Log Source</th>
<th># messages</th>
<th># patterns</th>
<th># labels</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows Server</td>
<td>61,190</td>
<td>140</td>
<td>12</td>
</tr>
<tr>
<td>RMS</td>
<td>21,736</td>
<td>106</td>
<td>10</td>
</tr>
<tr>
<td>Web Browser</td>
<td>997,176</td>
<td>26</td>
<td>11</td>
</tr>
</tbody>
</table>

\textsuperscript{2}Names are not revealed due to non-disclosure issues.

\textsuperscript{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., \( P = \frac{|E \cap E_G|}{|E_G|} \). Recall measures the fraction of edges in the ground truth graph that are recovered in the result graph G, i.e., \( R = \frac{|E \cap E_G|}{|E|} \). We also report \( F_1 \) score that considers both precision and recall, i.e., \( F_1 = \frac{2PR}{P+R} \). 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 \( q(e) < \theta \), \( \theta \) 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 \( \mathcal{G} \). 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.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Threshold ( \theta = 0.1 )</th>
<th>Threshold ( \theta = 0.5 )</th>
<th>StoryLine</th>
<th>K-cores</th>
<th>ESRE</th>
<th>SED</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server</td>
<td>P 0.76</td>
<td>R 0.62</td>
<td>0.33</td>
<td>0.86</td>
<td>1</td>
<td>0.87</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.28</td>
<td>0.93</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.61</td>
<td>0.67</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RMS</td>
<td>P 0.67</td>
<td>R 0.77</td>
<td>0.31</td>
<td>0.18</td>
<td>0.83</td>
<td>0.75</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.84</td>
<td>0.84</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Browser</td>
<td>P 0.88</td>
<td>R 0.83</td>
<td>0.27</td>
<td>0.31</td>
<td>0.56</td>
<td>0.86</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0.4</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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 strutures 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
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 reflect 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 reflect 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.

7.7 Parameter Study

In this section, we study the effect of the four parameters: $\lambda_e$, $\lambda_r$, $\lambda_n$ and $\lambda_c$ 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 $\lambda_e$ and $\lambda_r$:** We first derive a condition under which include an edge, $e$, when minimizing the graph energy. From
with respect to different values of \( P \).

Moreover, the selection of nodes is not affect the result for our datasets, as the selection of nodes is implicit in the workflow graph using undirected edges. Relations among logs, such as running in parallel. Those relations among log patterns in this paper. There are other useful relations in log messages. The structural events are useful for status monitoring 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.

Figure 4: The number of components in the resulting graph with respect to different values of \( \lambda_c \).

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

\[
\delta(e) = \lambda_e \times (1 - q(e)) - \lambda_r \times q(e)
\]

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

**Node parameter \( \lambda_e \):** We found that the values of \( \lambda_e \in [0,1] \) do not affect the result for our datasets, as the selection of nodes is also implicitly considered in Eq. 11.

**Connectivity parameter \( \lambda_c \):** 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 \( \lambda_c \). 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 \( \lambda_c \) increases from 0.1 to 1. These results suggests that the detected events are not sensitive to the value of parameter \( \lambda_c \).

### 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 \( \to \) P2, the transition P3 \( \to \) P4 happens far less frequently, as multiple keys need to be pressed to input an item. Threshold based method can easily transition P3 \( \to \) 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 \( \to \) P80 \( \to \) 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 \( \to \) P80 \( \to \) P83. If we only consider transitions between two patterns, P80 \( \to \) P81 and P80 \( \to \) 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 opposed 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.

ACKNOWLEDGEMENTS

Zhenhui Li and Fei Wu were supported in part by NSF awards #1618448, #1652525, #1639150, and #1514445. The views and conclusions contained in this paper are those of the authors and should not be interpreted as representing any funding agencies.
REFERENCES