Large-scale Frequent Episode Mining from Complex Event Sequences with Hierarchies

XIANG AO, Institute of Computing Technology, Chinese Academy of Sciences, China and University of Chinese Academy of Sciences, China
HAORAN SHI, University of California, Irvine, USA
JIN WANG, University of California, Los Angeles, USA
LUO ZUO, HONGWEI LI, and QING HE, Institute of Computing Technology, Chinese Academy of Sciences, China and University of Chinese Academy of Sciences, China

Frequent Episode Mining (FEM), which aims at mining frequent sub-sequences from single long event sequence, is one of the essential building blocks for sequence mining research field. Existing studies about FEM suffer from unsatisfied scalability when facing with complex sequences as it is an NP-complete problem for testing whether an episode occurs in a sequence. In this paper, we propose a scalable, distributed framework to support FEM on "big" event sequences. As a rule of thumb, "big" illustrates an event sequence is either very long or with masses of simultaneous events. Meanwhile, the events in this paper are arranged in a predefined hierarchy. It derives some abstractive events which can form episodes may not directly appear in the input sequence. Specifically, we devise an event-centered and hierarchy-aware partitioning strategy to allocate events from different levels of the hierarchy into local processes. We then present an efficient special-purpose algorithm to improve the local mining performance. We also extend our framework to support maximal and closed episode mining in the context of event hierarchy, and to the best of our knowledge, we are the first attempt to define and discover hierarchy-aware maximal and closed episodes. We implement the proposed framework on Apache Spark and conduct experiments on both synthetic and real-world datasets. Experimental results demonstrate the efficiency and scalability of the proposed approach and show that we can find practical patterns when taking event hierarchies into account.

Additional Key Words and Phrases: Frequent episode mining, peak episode miner, large-scale sequence mining, hierarchy-aware maximal/closed episode

ACM Reference Format:
Xiang Ao, Haoran Shi, Jin Wang, Luo Zuo, Hongwei Li, and Qing He. 2018. Large-scale Frequent Episode Mining from Complex Event Sequences with Hierarchies. *ACM Trans. Intell. Syst. Technol.* xx, xx, Article xx (July 2018), 24 pages. https://doi.org/0000001.0000001

Authors' addresses: Xiang Ao, Institute of Computing Technology, Chinese Academy of Sciences, Key Lab. of Intelligent Information Processing of Chinese Academy of Sciences(CAS), No.6 Kexueyuan South Rd, Haidian Dist. Beijing, 100190, China, University of Chinese Academy of Sciences, Beijing, 100049, China, aoxiang@ict.ac.cn; Haoran Shi, University of California, Irvine, Department of Computer Science, USA, haoras4@uci.edu; Jin Wang, University of California, Los Angeles, Computer Science Department, USA, jinwang@cs.ucla.edu; Luo Zuo; Hongwei Li; Qing He, Institute of Computing Technology, Chinese Academy of Sciences, Key Lab. of Intelligent Information Processing of Chinese Academy of Sciences(CAS), Beijing, China, University of Chinese Academy of Sciences, Beijing, 100049, China.

2157-6904/2018/7-ARTxx $15.00
https://doi.org/0000001.0000001

1 INTRODUCTION

Mining frequent sub-sequences from sequential data is helpful for researchers to perceive inherent correlations implied in data. *Frequent episode mining* (FEM) [31], as one of the fundamental tasks in sequence mining, has emerged as a popular research topic in recent years [4, 5, 7, 16, 27, 39, 40]. Here an *episode* (also known as serial episode [31]) refers to a totally ordered set of events. FEM then aims at identifying all frequent episodes with the frequency larger than a given threshold when facing with a single long event sequence. It has been successfully applied to various applications, e.g. mining financial correlations [4, 5, 7, 21, 27, 34], fault diagnostics for manufacturing [43], user behavior analysis [21, 44], intrusion detection [22], transportation management [7, 41], protein sequence classification [11], and text mining [6, 16, 40].

Several characteristics make most existing FEM solutions time-consuming. First, episode frequencies may fail to hold the anti-monotonicity property. For instance, *all occurrence*, *minimal occurrence* and *head frequency* may lead to the frequency of a sub-episode less than that of its super-episode [2]. Second, testing whether an episode occurs in a sequence is an NP-complete problem [41]. As a result, most of existing exact FEM algorithms adopt intricate and computing-intensive searching and enumerations. In addition, given the number of event sets in the sequence as N and the average size of each event set as M, the estimated possible search space for episode occurrences without any constraint could be $O(\sum_{i=1}^{N} M^i)$, which suffers from an exponential increase. Therefore, most current FEM solutions may encounter efficiency problems when scaling them to big sequence. As a rule of thumb, a *big* sequence is either with masses of sequentially occurred event sets, i.e. with a large N, or every event set involves lots of simultaneous events, i.e. with a large M.

Nowadays, there is great requirement in mining such big sequences in real world scenarios. For instance, high-frequency Forex price sequences, take minute price as an example, could exceed 10 million because Forex performs 24 hours trading since 1971[1]. If we study the NASDAQ-listed stocks and construct an event sequence with their daily closing prices, we may get a sequence having more than 3000 simultaneous events[2]. Sequences with similar characteristics can be found in other fields such as biology, human behavior, natural language and etc. The chromosome of human consists of a sequence of three billion base pairs of nucleotides[3], in which every nucleotide can be regarded as an event. Charles Hamilton, the most prolific author in the world, is estimated to have written about 100 million words in his lifetime, and we could obtain a rather long text (event) sequence once we connect every sentence of his output one by one[4].

In most applications mentioned above, the individual events could be naturally arranged in pre-defined hierarchies, and the hierarchies may demonstrate different properties due to various applications. For instance, stocks can form an industry and sector hierarchy: the stock "BIDU" in NASDAQ generalizes to its industry of "Internet Software & Services", which in turn generalizes to its sector of "Technology". The individual words in text can be organized in a syntactic hierarchy. For example, the word "learns" can first generalize to its lemma "learn", which further generalizes to its corresponding part-of-speech tag "VERB". Another example can be found in custom behavior sequence in which products can be arranged by a product hierarchy such as "iPhone X" belongs to "cell phones" subsequently generalizing to "telecommunication facilities".

In this paper, we are interested in an extensive form of frequent episode mining, in which the event hierarchies are taken into consideration. In particular, we allow the events in episodes belong to different levels of the event hierarchy. Though the episodes containing abstractive concepts may

[1] https://admiralmarkets.com/education/articles/forex-strategy/how-does-the-forex-market-trade-24-hours-a-day
[2] http://www.nasdaq.com/screening/companies-by-industry.aspx?exchange=NASDAQ
[3] https://en.wikipedia.org/wiki/Human_genome
[4] https://en.wikipedia.org/wiki/Charles_Hamilton_(writer)

not explicitly appear in the original input data, they are essential for different real world scenarios. For example, we might unearth potential correlations between multiple industry sectors in stock market [7, 17]. Here a sector consists of a group of individual stocks. If we only mine frequent episodes on the level of individual stocks we may fail to discover enough occurrences to make some episodes pass the frequency threshold. However, with the help of event hierarchies, we may derive frequent and meaningful patterns. In the context of text mining, such episodes include relational patterns, e.g., "NOUN is the capital of NOUN", or generalized n-grams such as "one of the ADJ NOUN" and "a ADJ woman", etc. In such cases, the episodes do not actually occur in the input data, but are useful for corresponding NLP tasks, e.g. information extraction [33] and language modeling [26]. Meanwhile, we are also aware of the successes of exploiting hierarchies in mining market-basket data [38] and web-usage logs [25].

While mining transactional or temporal databases with hierarchies has been well-studied [9, 10, 19, 38], it is still underexplored for episode mining in event sequences. The most straightforward approach to solve such problem is to rewrite the input data by replacing every event with an event set consisting of such event and its ancestors. For instance, replace "learns" by {"learns", "learn", "VERB"} according to the syntactic hierarchy. Then, conventional FEM algorithms can be adopted and pruning or post-processing techniques are utilized to provide consistent output. Interesting enough, such data augment might challenge the efficiency of processing significantly. Recall that the estimated possible search space for episode occurrences could be $O(\sum_{i=1}^{N} M^i)$, where M indicates the average size of each event set in the sequence, and such rewrite would increase M clearly. As a consequence, traditional FEM algorithms might be inefficient or even infeasible to deal with such complex sequences with event hierarchies. Scalable FEM algorithms towards on big sequence with event hierarchies are in urgent demand.

In this paper, we propose LA-FEMH (**LA**rge-scale **F**requent **E**pisode **M**ining with **H**ierarchies), a scalable distributed framework for frequent episode mining from complex sequence with event hierarchies. To improve the performance, we first propose an event-centered and hierarchy-aware strategy to partition the sequence into pieces so as to evenly allocate it into local processes and guarantee the scalability. In such strategy, we adopt optimized rewrite skills to reduce duplicate information both within one partition and among different partitions. Then, we devise an efficient local mining algorithm *peak episode miner* (PEM for short) to further avoid redundant computing. We also make an extension of our framework and propose LA-FEMH+ to support other episode mining tasks such as maximal and closed episode mining in the context of event hierarchies. To demonstrate the effectiveness of our approach, we implemented LA-FEMH and LA-FEMH+ with MapReduce on Apache Spark and performed experimental studies on both synthetic and real-world datasets including financial sequences and natural language text. The experimental results show that the proposed approach has significant performance gain in efficiency and scalability. At the same time, our results also suggest the effectiveness of episode with hierarchies.

The main contributions of this paper are summarized as following:

- We propose a scalable distributed framework LA-FEMH, with an event-hierarchy-aware partition strategy, which divides the input sequence to produce a balanced workload partition.
- We propose a specialized local miner named PEM which performs efficient specialized episode mining in local process with the help of the proposed tree-like structure and concise scannings.
- We propose the concept of maximal and closed episode in the context of event hierarchies and extend our LA-FEMH to support hierarchy-aware maximal and closed episode mining.
- We conduct extensive experiments on both synthetic and real word datasets. The experimental results demonstrate that our proposed method can clearly outperform existing methods and find more meaningful patterns by involving the event hierarchies.

The remainder of this paper is organized as follows. We discuss the related work in literature in Section 2. We present preliminary concepts and the problem statement in Section 3. Then we give an overview of LA-FEMH and alternative baseline algorithms in Section 4. In Section 5 we introduce the partition strategy of LA-FEMH. In Section 6, we present the local mining algorithm. We extend our framework to support maximal and closed episode mining in Section 7. We report the results of experimental study in Section 8. Finally we conclude the paper in Section 9.

2 RELATED WORK

The related researches in literature are broadly categorized as follows.

Frequent episode mining. FEM is an essential operation in the area of complex event processing [18]. The problem of frequent episode mining was first introduced by Mannila et al. [31], which was used to mine frequent sub-sequences from alarm sequences in telecom networks. Then it was widely used to analyze various forms of data like web navigation logs [13, 31], time-stamped fault reports in manufacturing plants [24], sales transactions [8, 44], stock data [5, 21, 28, 34], news [6], and so on [35, 42, 47]. Depending on different applications, distinct definitions of episode frequency were proposed to unearth different types of frequent episodes. Achar et al. [2] reviewed a variety of frequency definitions, among which *minimal occurrence* is one of the most widely used frequency measures. Existing FEM algorithms usually fall into two categories: breadth-first enumeration (BFS) and depth-first enumeration (DFS). The BFS algorithms mainly involve two steps: candidate generation and frequency counting. Among them, MINEPI [30] is the first minimal-occurrence based BFS algorithm. Unlike BFS approaches, the DFS algorithms discover frequent episodes without candidate generation but by expanding prefix in the sequence, e.g. DFS [1], UP-Span[44], EPT [29], etc. There are also efforts that focus on mining subclasses of episode, such as maximal episode [29], closed episode [41] and episode with unique labels [3]. However, few of them take event hierarchies into account. To the best of our knowledge, we are the first attempt to define hierarchy-aware maximal and closed episode in the context of event hierarchies.

Pattern mining over hierarchies. Hierarchy based (sequential) pattern mining was first introduced by Srikant and Agrawal [37, 38]. In their work, the use of extended transactions are considered to incorporate hierarchies into the mining procedure. Similar idea can be found in multiple-level association rule mining [19]. In Han et al.'s work [19], transaction table was firstly encoded by extended items from hierarchies and a top-down progressive deepening approach was proposed for mining strong association rules from sales transactions. Except for general patterns, new types of pattern particularly based on hierarchies was also proposed. Barsky et al. [9], based on item hierarchies, proposed new type pattern called flipping patterns, which are specific for a given abstraction level, and flip from positive to negative and vice versa when items are generalized to a higher level. Zhang et al. [46] utilized semantic hierarchies to discover fine-grained sequential patterns in trajectories of social media users.

Distributed sequence mining. Parallel sequence mining has emerged to support large scale sequence mining tasks. Generally, parallel sequence mining algorithms were mainly proposed by extending the existing serial algorithms. For example, Zaki [45] extended his serial frequent sequential pattern mining algorithm SPADE to the shared memory parallel architecture, creating pSPADE. Similar idea also can be found in Par-ASP [14] and Par-CSP [15].

Recent representative studies for parallel sequence mining are MG-FSM [32] and LASH [10] algorithms. Their major contributions were proposing several ways to compress projected temporal databases for gap-constrained sequential pattern mining. LASH, as a successor of MG-FSM, can handle item hierarchies in advance. Our work is inspired by MG-FSM and LASH. First, we devise hierarchy-aware variant of event-based partitioning to divide the whole input sequence into work-load balanced pieces. A novel specialized algorithm subsequently mines each partition in parallel and

independently. The major differences between our method and LASH (MG-FSM) include: 1) the input data of LASH is temporal database which consists of many independent short sequences. Hence there is no need to divide the input data into pieces. However, in our problem how to derive work-load balanced sequence partitions remains a challenge problem. 2) the maximal interval threshold between any two *consecutive* items in subsequences is specified in LASH, which significantly reduces the search space for the process of local mining. While in our scenario, we free such parameter and discover minimal occurrences for episodes, which may challenge the local mining process since minimal occurrence is a kind of frequency measure without anti-monotonicity property. Therefore, we argue that both MG-FSM and LASH cannot adapt to our mining problem in this paper directly.

There are also other parallel sequence mining algorithms, e.g. PSmile [12] and ACME [36], which search motif from biology sequence with small alphabets, typically four symbols for DNA sequence. However, as they cannot be implemented in the MapReduce programming framework, we cannot compare with them in a fair environment with the same degree of fault tolerance and scalability. Thus we do not regard them as baselines in this paper.

3 PRELIMINARIES AND PROBLEM STATEMENT

In this section, we provide preliminaries, related concepts and problem statement of our mining problem.

Event hierarchy. Suppose we have a finite event hierarchy $\mathcal{E}$ in which each event has at most one parent[5]. The most specific nodes with no descendants are known as *leaf* events, the most general nodes with no ancestors are known as *root* events, and other nodes are known as *intermediate* events. For two events $p, q \in \mathcal{E}$, if p is a child of q in the hierarchy, we call p *directly generalizes* to q, denoted by $p \Rightarrow q$. And we use $\Rightarrow^*$ to denote the transitive closure of $\Rightarrow$. Figure 1 gives an example of event hierarchy that will be used throughout the paper. In this figure, b_{12} refers to a leaf event, b_1 is an intermediate event and B refers to a root event. We also have $\mathsf{b}_{12} \Rightarrow \mathsf{b}_1$ and $\mathsf{b}_{12} \Rightarrow^* \mathsf{B}$.

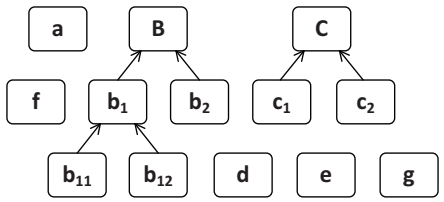


Fig. 1. Event hierarchy.

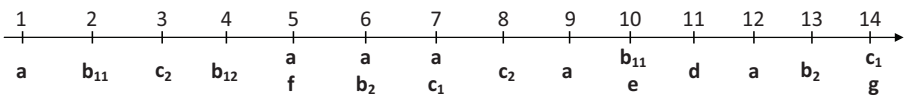


Fig. 2. The running example of event sequence.

Event sequence. An event sequence, $\vec{S} = \langle (E_1, t_1), (E_2, t_2), \cdots, (E_n, t_n) \rangle$, is an ordered sequence of events. It consists of all events associated with time stamp t_i, where $t_j < t_k$, $\forall 1 \leq j < k \leq n$. Each event $e \in E_i$, where $1 \leq i \leq n$, is a *leaf event* in the event hierarchy $\mathcal{E}$. Figure 2 shows an event sequence in which every event is a leaf event in the event hierarchy shown in Figure 1. This event sequence will be used throughout the paper as the running example.

Episode. An episode α (also known as serial episode in this paper) is a non-empty *totally ordered* event set in the form of $e_1 \rightarrow \cdots \rightarrow e_i \rightarrow \cdots \rightarrow e_k$ where $e_j \in \mathcal{E}, \forall j \in [1, k]$ and for $1 \leq i < j \leq k$, the

[5]Here event hierarchy is modeled as a forest. Our approaches also support directed acyclic graph style hierarchies.

event e_i occurs before e_j. Without ambiguity, we use the word "episode" to denote serial episode in this paper. The *length* of an episode α, denoted by $|\alpha|$, indicates the number of events it has. And an episode α of length k is called a k-episode. Note that in this paper we allow all kinds of events in the hierarchy to form an episode. For example, $\alpha = \mathsf{a}{\rightarrow}\mathsf{B}{\rightarrow}\mathsf{b}_1$ is a 3-episode where a is a *leaf event*, B is a *root event* and b_1 is an intermediate event in the example event hierarchy. An event d can also be viewed as a 1-episode.

Subepisode. Consider two episodes $\alpha = e_1{\rightarrow}\cdots{\rightarrow}e_k$ and $\beta = e'_1{\rightarrow}\cdots{\rightarrow}e'_m$ where $m \leq k$, the episode β is a *subepisode* of α, denoted by $\beta \sqsubseteq \alpha$, iff there exist m integers $1 \leq i_1 \leq i_2 \cdots \leq i_m \leq k$ such that $e'_j = e_{i_j}$ for every $j \in [1, m]$. For example, $\mathsf{a} \rightarrow \mathsf{b}_{11}$ and $\mathsf{a} \rightarrow \mathsf{d}$ are subepisodes of the episode $\mathsf{a} \rightarrow \mathsf{b}_{11} \rightarrow \mathsf{d}$, however $\mathsf{d} \rightarrow \mathsf{b}_{11}$ is not.

Episode occurrence. Given an episode $\alpha = e_1 \rightarrow \cdots \rightarrow e_j \rightarrow \cdots \rightarrow e_k$, $[t_{i_1}, \cdots, t_{i_j}, \cdots, t_{i_k}]$ is an *occurrence* of α iff: (1) there exists $e'_j \Rightarrow^* e_j$ (recall that $e'_j \Rightarrow^* e_j$ includes the case $e'_j = e_j$) s.t. $e'_j \in E_{i_j}$ for all $j \in [1, k]$, where E_{i_j} is the event set associated with the time stamp t_{i_j}; (2) $t_{i_1} < t_{i_2} < \cdots < t_{i_k}$, and $t_{i_k} - t_{i_1} < \delta$ where δ is the *maximum occurrence window threshold*. t_{i_1} and t_{i_k} form the *occurrence time interval* of an episode occurrence. Here we consider the episode occurrence as a kind of *generalized* form, i.e., events in an episode are regarded to implicitly occurring on the sequence if we could find some leaf events on sequence that are generalized to them.

For example, $[4, 5]$, $[4, 6]$ and $[10, 12]$ are occurrences of the episode $\mathsf{b}_1{\rightarrow}\mathsf{a}$ if $\delta = 3$. Since $\mathsf{b}_{12} \Rightarrow^* \mathsf{b}_1$ and $\mathsf{b}_{12} \in E_4$, we can derive that b_1 also occurs at the time stamp 4 implicitly. Similarly, b_1 occurs at 10 since $\mathsf{b}_{11} \in E_{10}$ and $\mathsf{b}_{11} \Rightarrow^* \mathsf{b}_1$. $[4, 7]$ is not a valid occurrence of $\mathsf{b}_1{\rightarrow}\mathsf{a}$ since it exceeds the boundary of maximum occurrence window threshold.

Minimal occurrence. Given an occurrence time interval $[t_i, t_j]$ of an episode α, we call $[t_i, t_j]$ is *minimal* if α does not occur at any proper subinterval $[t'_i, t'_j] \subset [t_i, t_j]$. t_i and t_j are called *start time* and *end time*, respectively. The set of all distinct minimal occurrences of α is denoted $\mathrm{moSet}(\alpha)$. For example, $\mathrm{moSet}(\mathsf{b}_1{\rightarrow}\mathsf{a}) = \{[4, 5], [10, 12]\}$ in Figure 2 if $\delta = 3$. $[4, 6]$ fails to be minimal since it subsumes another occurrence of $\mathsf{b}_1{\rightarrow}\mathsf{a}$, namely $[4, 5]$.

Frequent episode. We denote the *support* of an episode α by $\mathrm{sp}(\alpha)$, which is the number of its distinct minimal occurrences, i.e., $\mathrm{sp}(\alpha) = |\mathrm{moSet}(\alpha)|$. We say an episode α is *frequent* in $\vec{S}$ if its support value exceeds a user-specific *minimum support threshold* $min\_sup \geq 0$, i.e. $\mathrm{sp}(\alpha) \geq min\_sup$.

Problem Statement. Given an event sequence $\vec{S}$ with simultaneous events consisting of leaf events from an event hierarchy $\mathcal{E}$, a minimum support threshold min_sup and a maximum occurrence window threshold δ, the mining problem (FEMH) aims at finding all frequent episodes in $\vec{S}$.

4 FRAMEWORK OVERVIEW

In this section, we propose the framework of LA-FEMH. We implement our algorithms on Apache Spark [6] with its high-level operators (e.g. *flatMap*, *groupByKey*, *filter*, etc.) which enable the expression of programs as data flows of transformations on Resilient Distributed Datasets (RDD). The proposed algorithms can also be implemented on other distributed system supporting MapReduce programming framework. We first introduce two alternative partition strategies for implementing distributed FEMH (Section 4.1), then we present the overall framework of LA-FEMH (Section 4.2).

4.1 Partition Approaches

4.1.1 Timestamp-based Approach. An intuitive approach for distributed FEMH is to split the sequence according to timestamps. Event hierarchies, on the other hand, can be processed by imitating similar way used in association rule [37], i.e., extending event set on every time stamp with

[6] http://spark.apache.org/

an extensive event set consisting of all original events and their ancestors in the hierarchy. After that we can adopt conventional mining algorithms on each partition independently to solve the problem. Following this route, we give a timestamp-based approach with the following steps:

- extend the input sequence with the event hierarchies and split it into several approximate equal-length parts by timestamp;
- conduct any custom minimal-occurrence based FEM algorithm on each part to mine minimal episode occurrences;
- count the global frequency of every discovered episode.

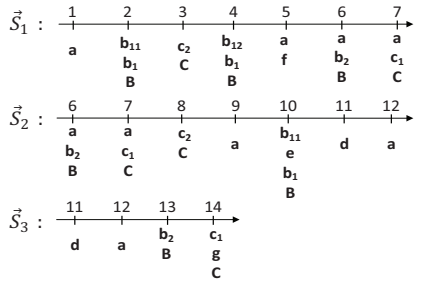


Fig. 3. Example of Timestamp-based Approach for Partition ($\delta = 3$).

Figure 3 demonstrates the results of partitioning the example sequence shown in Figure 2 into three pieces using the timestamp-based approach. In this process, event sets are extended in advance. Every partition is expected to contain equal number of event sets. In this example, as the input sequence contains 14 event sets, there are 5 event sets in each partition. Partitions may have overlaps for the purpose of completeness.

4.1.2 Event-projected Approach. Another alternative can be intuitively extended from the notion of item-based partitioning e.g. FP-growth [20] which underlines a number of (sequential) pattern mining algorithms. Specifically, we can partition the single sequence according to every occurrence of an event independently, and then process the derived projected subsequences of each event in parallel. Similar with the timestamp-based approach, any conventional FEM algorithm can be adopted to detect episode minimal occurrences on each individual projected sequence, and it also requires a global frequency counting to produce the final results.

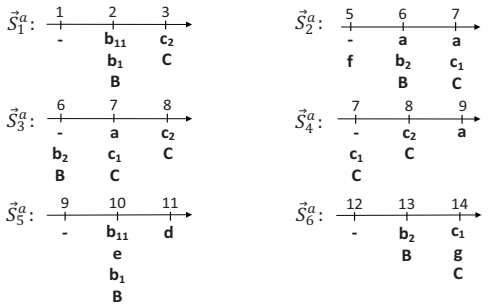


Fig. 4. Partition the event sequence in Fig. 2 by event-projected approach in which “a” is considered as the projected event ($\delta = 3$).

Figure 4 displays the partition results of the event-projected approach when a is considered as the projected event. We use “-” to denote the projected event occurrence. Here we also extend the event sets by the hierarchy in advance. The approach processes every occurrence of a independently.

There are totally six occurrences of a in the running example, thus the partition results contain six sub-sequences. Each of them starts from the timestamp where a happens. Its continuous consequent time intervals are determined by the parameter of δ, which is 3 in this example.

4.2 Framework of LA-FEMH

Though the timestamp-based and the event-projected approaches can be easily implemented, we argue that these alternatives could be generally inefficient and not scalable when the event sequence is *skewed*. There are two real world scenarios in which skewness can happen. The first one is with skewed time periods. In some sequences, most of simultaneous events are generated within a few time periods. For example, the stock price event sequence will generate much more simultaneous events in both bull and bear market than the fluctuation market. Similar scenario could also happen on the event sequence of electricity consumption or traffic condition, in which masses of simultaneous events are generated in rush hours. If we partition the sequence by time stamp, we may derive irregular work assignments that some of partitions have heavily computational burden while the others not. Remember that overall processing time is determined by the last finished process, which deteriorates the performance of timestamp-based approach in such cases. The second kind of skewness might occur on the event frequency, and it makes event-projected approaches derive to irregular partitions across different processes. For instance, as is shown in Fig. 2, the partition of event a contains 6 sub-sequences, while that of event b_2 only has 2 sub-sequences, which results in a 3×-skewed workload assignments.

Hence, we propose a novel framework LA-FEMH to provide balanced workload assignments among processes. Specifically, LA-FEMH is expected to generate more regular partitions with more balanced sub-sequences for discovering minimal episode occurrences. It could significantly improve the overall performance in the scenarios when event sequences are skewed, e.g., financial correlation discovery, electricity usage management, traffic management etc.

The LA-FEMH framework consists of two parts: a work partitioning strategy that assigns balanced workload to different processes and an efficient local mining algorithm on each individual process. Accordingly, we apply an event-centered and hierarchy-aware sequence partitioning including flexible sequence rewrites. LA-FEMH alleviates the impacts from both the amount of event types and the number of sub-sequences. To this end, for every frequent event e, it constructs a partition $\mathcal{P}^e$ which consists of a group of rewritten sub-sequences of $\vec{S}$. Such $e \in \mathcal{E}$ can explicitly or implicitly occurs on $\vec{S}$. Then LA-FEMH mines episodes on each partition $\mathcal{P}^e$ in parallel with a special-purpose local mining algorithm which only detects episodes containing the corresponding event e. Finally, local output are collected to obtain the global results.

Following this routine, LA-FEMH consists of three stages: a pre-scanning stage, a sequence partition stage and a local mining stage. The details are shown in Algorithm 1.

Pre-scanning In this stage, LA-FEMH performs a linear scan on the event sequence $\vec{S}$ to discover frequent events and derives a *generalized event list* (g-list) with a total order $\prec$ as defined in Definition 4.1.

Definition 4.1. **The total order $\prec$ on g-list.** The total order $\prec$ is denoted as $e_1 \prec e_2$. Denoted by $\mathrm{sp}(e)$ as the frequency of an event e, $e_1 \prec e_2$ holds if either $\mathrm{sp}(e_1) > \mathrm{sp}(e_2)$ or e_1 is located at a higher level in the event hierarchy when $\mathrm{sp}(e_1) = \mathrm{sp}(e_2)$. The remaining ties are broken by lexicographical order. We call the event e_1 is *relevant* to e_2 and e_2 is *irrelevant* to e_1 if $e_1 \prec e_2$.

Figure 6 (a) shows the g-list of the example sequence when $min\_sup = 3$, and we have a $\prec$ B $\prec$ C $\prec$ b_1 on it. d, e, f and g are excluded from g-list because they are infrequent.

Sequence Partition The sequence partition stage of LA-FEMH is performed via a *Map* job. In this process, events in the g-list are mapped over to different processes. For each e in the g-list, we

ALGORITHM 1: LA-FEMH Framework

Input: $\vec{S}$: the event sequence;
min_sup: threshold of minimum support;
δ: threshold of maximum occurrence window;

1 g-list $\leftarrow$ Perform pre-scanning on $\vec{S}$ and get the generalized event-list
2 Map(g-list)
3 **begin**
4 **forall the** $\langle e, \mathrm{moSet}(e)\rangle \in g$-*list s.t.* $\mathrm{sp}(e) \geqslant min\_sup$ **do**
5 **foreach** *occurrence time interval* $[t_i, t_i] \in \mathrm{moSet}(e)$ **do**
6 Construct $\mathcal{P}^e_{t_i}(\vec{S})$ //Sec. 5
7 Emit $(e, \mathcal{P}^e_{t_i}(\vec{S}))$
8 Map($\mathcal{P}$)
9 **begin**
10 **forall the** $(e, \mathcal{P}^e_{t_i}(\vec{S})) \in \mathcal{P}$ **do**
11 Mine peak episode minimal occurrences in $\mathcal{P}^e_{t_i}(\vec{S})$ //Sec. 6
12 **forall the** *peak episode* $\alpha \in \mathcal{P}^e_{t_i}(\vec{S})$ **do**
13 Emit $(\alpha, \mathrm{sp}(\alpha))$
14 Reduce(α)
15 **begin**
16 Compute global frequency of α
17 Filter infrequent episodes and collect results

iterate over each occurrence time interval $[t_i, t_i]$ in $\mathrm{moSet}(e)$ and generate a sub-sequence $\mathcal{P}^e_{t_i}(\vec{S})$ of $\vec{S}$ based on both $[t_i, t_i]$ and δ. We further adopt optimized rewrite skills to simplify the sub-sequences. All the generated sub-sequences are emitted for further using in the mining stage (Line 4–7). We will introduce the details of sequence partition stage in Section 5.

Local Mining The local mining stage is carried out by a consequent *MapReduce* job. First, we directly *Map* the partitioned sub-sequences $\mathcal{P}$ with Equation 1 to different processes (Line 10).

$$\mathcal{P} = \bigcup_{\substack{e\in\mathcal{E},\\ [t_i,t_i]\in\mathrm{moSet}(e)}} \mathcal{P}^e_{t_i}(\vec{S}) \tag{1}$$

Then we mine every subsequence $\mathcal{P}^e_{t_i}(\vec{S})$ in parallel and output a special kind of episodes named *peak episode* defined in Definition 4.2.

Definition 4.2. **Peak episode.** Given a partition $\mathcal{P}^e$, we call the episode containing the event e but no irrelevant event according to $\prec$ on g-list as *peak episode*. e is referred to peak event to $\mathcal{P}^e$.

For example, episodes including B and optionally a will be peak episodes of $\mathcal{P}^{\mathsf{B}}$ according to the g-list in Figure 6 (a).

Instead of using conventional mining algorithm with an additional post-processing steps, we propose an efficient approach called *peak episode miner*, which directly produces the set of peak episode minimal occurrences for each partition (Line 11). Details of such local mining algorithm will be shown in Section 6. Finally, we *Reduce* the emitted episode occurrences to compute global frequencies, filter out the false positive and generate the final results (Line 16–17).

5 SEQUENCE PARTITIONING

In this section, we introduce the sequence partitioning approach. The basic idea is to adopt an event occurrence centered partitioning strategy to evenly split the sequence. Consider an occurrence time interval $[t_i, t_i]$ for event e, we construct $\mathcal{P}^e_{t_i}(\vec{S})$ as $\vec{S}^{t_i+\delta-1}_{t_i-\delta+1}$, where δ is the threshold of maximum

occurrence window, and $\vec{S}_j^k$ denotes the sub-sequence of the event sequence $\vec{S}$ starting from time stamp j and ending with time stamp k ($1 \leq j < k \leq t_n$). In this way we can get the partition of B, i.e. $\mathcal{P}^{\mathsf{B}}$, when $\delta = 3$, as shown in Figure 5.

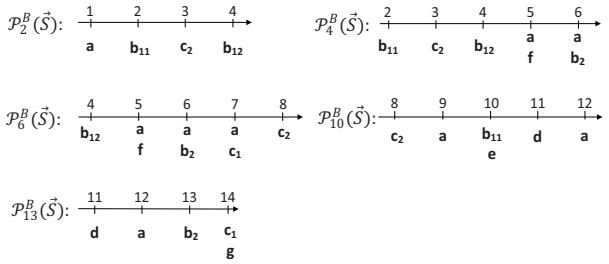


Fig. 5. Partition $\mathcal{P}^{\mathsf{B}}$ by event occurrence centered partitioning strategy without optimized rewrites.

Although such partitioning is effective for discovering minimal occurrence of episode, it is inefficient in our task for the following reasons. First, duplicated episode minimal occurrences may appear in different sub-sequences within a same partition. So we need extra checking to guarantee the correctness. For instance, $[4, 5]$ as the minimal occurrence of episode $\mathsf{B}\rightarrow\mathsf{a}$ will be discovered in both $\mathcal{P}_4^{\mathsf{B}}(\vec{S})$ and $\mathcal{P}_6^{\mathsf{B}}(\vec{S})$. So does the minimal occurrence $[5, 6]$ of $\mathsf{a}\rightarrow\mathsf{B}$. Such minimal occurrences appear only once in the original sequence. Second, duplicated episodes may exist on multiple partitions but output only in one partition. For example, episode $\mathsf{a}\rightarrow\mathsf{b}_1$ will be mined in $\mathcal{P}^{\mathsf{a}}$, $\mathcal{P}^{\mathsf{B}}$ and $\mathcal{P}^{\mathsf{b}_1}$ but outputted only in partition $\mathcal{P}^{\mathsf{b}_1}$ since $\mathsf{a}\rightarrow\mathsf{b}_1$ is a peak episode of the partition $\mathcal{P}^{\mathsf{b}_1}$. Third, partitions with high frequency events will still contain more sub-sequences than those with low frequency events, which could result in skewed workload assignments just as the event-projected approaches.

Next, we propose several optimized rewrite principles for constructing partition $\mathcal{P}^e$ to overcome the aforementioned shortcomings.

5.1 Merge according to neighborhood information

We first try to eliminate the redundancy within a same partition. To this end, we establish the notion of *δ-neighborhood* of a time interval, which is an essential concept for sequence partitioning in LA-FEMH.

Given a time window $[t_i, t_j], i, j \in [1, n]$ and the threshold of maximum occurrence window δ, we define its δ-neighborhood as the time interval$[\max(t_1, t_i - \delta + 1), \min(t_j + \delta - 1, t_n)]$, where t_1 and t_n indicate the first and the last timestamp of the input event sequence. With such concept, we observe that the first aforementioned shortcoming will happen if there exists other peak event occurrences in the δ-neighborhood for an occurrence $[t_i, t_i]$ of peak event e. Hence, if $j > i$ and t_j is in the δ-neighborhood of $[t_i, t_i]$, we can merge the sub-sequences $\mathcal{P}_{t_i}^e(\vec{S})$ and $\mathcal{P}_{t_j}^e(\vec{S})$ to generate a longer sub-sequence, denoted as $\mathcal{P}_{\langle t_i, t_j\rangle}^e(\vec{S})$. Here, t_i and t_j indicate the first and last occurrence time of the peak event in $\mathcal{P}_{\langle t_i, t_j\rangle}^e(\vec{S})$, respectively. We can further merge the generated sub-sequence $\mathcal{P}_{\langle t_i, t_j\rangle}^e(\vec{S})$ following the similar method until no more merge operations could be conducted. In particular, the time interval $[t_i, t_j]$ will be taken into account when computing the δ-neighborhood for the sub-sequence $\mathcal{P}_{\langle t_i, t_j\rangle}^e(\vec{S})$.

For example, $\mathcal{P}_2^{\mathsf{B}}(\vec{S})$ and $\mathcal{P}_4^{\mathsf{B}}(\vec{S})$ can be merged to $\mathcal{P}_{\langle 2,4\rangle}^{\mathsf{B}}(\vec{S})$ and further to $\mathcal{P}_{\langle 2,6\rangle}^{\mathsf{B}}(\vec{S})$ when $\delta = 3$ in Figure 5.

5.2 Simplifying by the relevance

By looking at the δ-neighborhood, we expect to alleviate redundancy within a same partition. But the generated $\mathcal{P}^e_{\langle t_i,t_j\rangle}(\vec{S})$ might have the same length with the original sequence $\vec{S}$ and bring irregular workload assignments in the worst case. Such worst case will happen when there always exists new peak event occurrence in the δ-neighborhood of the merged sub-sequence. To further balance the skewness among multiple partitions, we perform another rewrite that only *relevant* events are kept in every sub-sequence $\mathcal{P}^e_{\langle t_i,t_j\rangle}(\vec{S})$, which is called simplifying by the relevance.

Directly drop all irrelevant events from a partitioned sub-sequence $\mathcal{P}^e_{\langle t_i,t_j\rangle}(\vec{S})$ may lead to false negative since generalizations of irrelevant events may become relevant. For example, b_1 is irrelevant to C but B is C-relevant. Hence, we may need to rewrite the irrelevant events by other relevant events based on the total order on g-list (Definition. 4.1). That is, if an event h is irrelevant to a given peak event e and does not have any ancestor $h' \prec e$, it can be dropped safely. Otherwise, let H denote the set of ancestors of h in which each element is e-relevant. We then replace the occurrence of h by the elements in H. After we finish rewriting every $\mathcal{P}^e_{t_i}(\vec{S})$, we further remove the prefix/suffix of the sequence if it does not contain relevant events.

For example, the results of $\mathcal{P}^{\mathsf{B}}$ after simplification are shown as Figure 6 (c). From the g-list, we know the sub-sequences in $\mathcal{P}^{\mathsf{B}}$ can only contain event B and a. As a result, events c_1, c_2, d–g are dropped from the sequences. While events b_1, b_2, b_{11} and b_{12} are generalized to B. Compared with Figure 5, we observe that $\mathcal{P}^{\mathsf{B}}$ in Figure 6 (c) is more concise. The final partition for B contains only 3 sub-sequences with 12 event occurrences versus 5 sub-sequences with 30 event occurrences.

Finally, we summarize the correctness of our rewrites in Theorem 5.1.

THEOREM 5.1. *Given an event e, the peak episodes in the sequence partition $\mathcal{P}^e$ after two rewrites in Section 5.1 and 5.2 are the same with those in the original event sequence $\vec{S}$.*

PROOF. Denoted by $\mathcal{G}_e(\mathcal{P}^e)$ the set of peak episodes in $\mathcal{P}^e$ and $\mathcal{L}_e(\vec{S})$ the set of episodes whose largest event on the total order of g-list is e in $\vec{S}$. Hence we have to prove $\mathcal{G}_e(\mathcal{P}^e) = \mathcal{L}_e(\vec{S})$. Let $\alpha = e_{\alpha_1} \rightarrow \cdots \rightarrow e_{\alpha_k} \in \mathcal{L}_e(\vec{S})$. By its definition, there exist some events $e_{\alpha_j} = e$ where $j \in [1,k]$ and other events $e_{\alpha_i} \prec e$ for $1 \le i \le k$. Thus there exist k event sets $E_{i_1}, \cdots, E_{i_k}$ in $\vec{S}$ where $1 \le i_1 < \cdots < i_k \le n$ and $t_{i_k} - t_{i_1} < \delta$, in which we can find an event $e' \in E_{i_w}$ having $e' \Rightarrow^* e_{\alpha_w}$ for $w \in [1,k]$. Event sets $E_{i_1}, \cdots, E_{i_k}$ could have their corresponding rewrites in $\mathcal{P}^e$ since there exists $e_{\alpha_j} = e$. We denote them by $E'_{i_1}, \cdots, E'_{i_k}$, respectively. If $e' \in E_{i_w} \prec e$, it does not modify by our rewrites so that we can find $e'' \in E'_{i_w}$ having $e'' = e' \Rightarrow^* e_{\alpha_w}$. Otherwise, we can find $e'' \in E'_{i_w}$ having $e' \Rightarrow^* e'' \Rightarrow^* e_{\alpha_w}$ since we replace e' by all the ancestors that are relevant to e, including e''. Hence we obtain $\mathcal{L}_e(\vec{S}) \subseteq \mathcal{G}_e(\mathcal{P}^e)$. $\mathcal{L}_e(\vec{S}) \supseteq \mathcal{G}_e(\mathcal{P}^e)$ can be proved by the similar idea. □

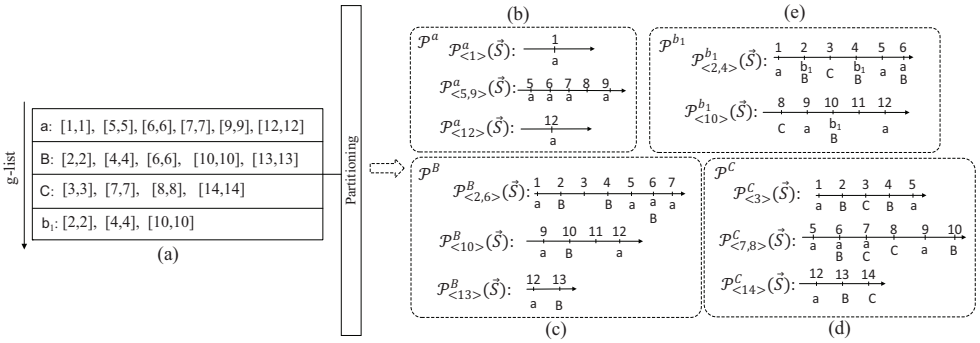


Fig. 6. Sequence partitioning of LA-FEMH, for $\delta = 3$ and $min\_sup = 3$.

5.3 Discussions on Sequence Partitioning of LA-FEMH

The partition approach in LA-FEMH generates more balanced workloads in terms of the g-list. Recall that each partition $\mathcal{P}^e$ is processed in parallel in the local mining stage. Take Figure 6 as an example, though the partition for top events on g-list, e.g. $\mathcal{P}^{\mathsf{a}}$, contains more sub-sequences, each of them is more concise compared with those on lower rankings, e.g. $\mathcal{P}^{\mathsf{b}_1}$.

Furthermore, the partition approach in LA-FEMH derives more evenly workload distribution compared with other alternatives. By taking the event occurrence centered partition strategy and the merging skill with neighborhood information (Section 5.1), LA-FEMH can reduce the number of generated sub-sequences and bound the length of them. After simplifying by relevance (Section 5.2), the size of event alphabet for each partition is further pruned. Generally speaking, with partitioning strategy in LA-FEMH, the processing time for each partition will be closer to each other and the overall mining time will constantly be reduced.

6 PEAK EPISODE MINER

In this section, we introduce the details of Peak Episode Miner (PEM). The goal of PEM is to optimize the local mining procedure on each individual process.

6.1 The Overview and Concepts of PEM

The input of PEM is a sub-sequence $\mathcal{P}^e_{\langle t_i,t_j\rangle}(\vec{S})$ generated from a specific partition $\mathcal{P}^e$, and its output is all the peak episodes in $\mathcal{P}^e_{\langle t_i,t_j\rangle}(\vec{S})$ with their local frequencies. In this process, PEM performs linear scans twice, namely a *backward scan* and a subsequential *forward scan*, over $\mathcal{P}^e_{\langle t_i,t_j\rangle}(\vec{S})$ for mining minimal occurrences of peak episodes.

Given an event e in $\mathcal{P}^e$ with its occurrence interval $[t_i, t_i]$, we first define a peak episode tree denoted by $\tau^e_{t_i}$, to store peak episode minimal occurrences involving such occurrence of e. It is a tree-like structure defined below.

1) The root node r, denoted by $(r.event{:}r.time)$, consists two fields: the corresponding event e, and the occurrence time of such event, i.e., $[t_i, t_i]$.
2) Each non-root node n, denoted by $(n.episode{:}n.mo)$, consists of two fields: a peak episode involving e, and a minimal occurrence of such episode in the δ-neighborhood of $[t_i, t_i]$.

For example, Figure 8 demonstrates the example of proposed peak episode trees for $\mathcal{P}^{\mathsf{C}}$, namely τ^{C}_7 and τ^{C}_8. Take τ^{C}_7 shown in Figure 8 (a) as an example. The root of such tree, denoted as (C:[7,7]), indicates the occurrence of C, i.e. [7, 7] in $\mathcal{P}^{\mathsf{C}}_{\langle 7,8\rangle} \in \mathcal{P}^{\mathsf{C}}$. The left-most leaf node, which is (a→a→C:$[5, 7]$), represents a minimal occurrence of peak episode a→a→C in $\mathcal{P}^{\mathsf{C}}_{\langle 7,8\rangle}$.

6.2 The PEM Algorithm

With the peak episode tree, we could represent all the corresponding peak episode minimal occurrences associated with a peak event occurrence into a single tree. Hence the purpose of PEM is then to construct and maintain peak episode trees when it meets a sub-sequence $\mathcal{P}^e_{\langle t_i,t_j\rangle}(\vec{S})$. PEM accomplishes such task by sequentially invoking two internal procedures called Backward_Scan and Forward_Scan. During each scan, PEM discovers peak episode minimal occurrences by constructing peak episode trees, expanding tree nodes and management the status of tree nodes. Finally, PEM produces a group of peak episode trees, which together store all the corresponding peak episode minimal occurrences in $\mathcal{P}^e_{\langle t_i,t_j\rangle}(\vec{S})$.

These two scan procedures can expand the peak episode trees through an analogous manner. Due to the space limitation, here we only introduce the detailed process of Backward_Scan. The Forward_Scan can be done in a similar way and with the same peak episode trees.

We give the pseudo-code of Backward_Scan as Algorithm 2. Here $\mathcal{P}^e_{\langle t_i,t_j\rangle}(\vec{S})$ is scanned by a reverse order. In particular, we obtain the event sets on such sequence starting from the time stamp t_j and stopping at $t_i-\delta+1$ (Line 2). Remember that $t_i-\delta+1$ is the start time of $\mathcal{P}^e_{\langle t_i,t_j\rangle}(\vec{S})$. For every obtained event set E'_k, if it contains an event of e, we construct a new peak episode tree $\tau^e_{t_k}$ and add it to the peak episode tree list $\mathcal{T}$ (Line 3–5). Then, we begin to update other existing peak episode trees in $\mathcal{T}$. All the peak episode trees $\tau^e_{t_p}$ s.t. $0<t_p-t_k<\delta$ will be considered (Line 6). For the update order, specifically, we require that $\tau^e_{t'_p}$ should be updated *earlier* than $\tau^e_{t''_p}$ if $t'_p<t''_p$.

ALGORITHM 2: Backward_Scan

Input: $\mathcal{P}^e_{\langle t_i,t_j\rangle}(\vec{S})$: a partitioned sub-sequence for peak event e;
δ: threshold of maximum occurrence window;
$\mathcal{T}$: peak episode tree list;

```
1  begin
2      for t_k = t_j to t_i − δ + 1 do
3          if e ∈ E'_k in P^e_⟨t_i,t_j⟩(S⃗) then
4              build peak episode tree τ^e_{t_k}
5              add τ^e_{t_k} to T
6          for τ^e_{t_p} ∈ T s.t. 0 < t_p − t_k < δ do
7              forall the q ∈ τ^e_{t_p} do
8                  for e' ∈ E'_k do
9                      if q satisfies constrains in Lemma 6.2 then
10                         extend a new node q' ← (e'→q.episode:[t_k, t_p]) as the child node of q
11                         Emit (q'.episode, 1)
12                 check and update the status of node q
13     return T
```

In this process, we do not need to traverse every node q on a given τ^e_p. According to the definition of episode minimal occurrence, for an event $e'\in E'_k$, we can avoid visiting a node q if its corresponding time intervals do not fall into a particular range. To represent such relations, we propose the definition of *general last minimal occurrence* shown as in Definition 6.1.

Definition 6.1. **General last minimal occurrence.** Given a time interval $[t_i,t_j]$, a minimal occurrence of α, denoted by $[t_1,t_2]$, is called a *general last minimal occurrence* of α within $[t_i,t_j]$ iff there is no other minimal occurrence of α, denoted by $[t'_1,t'_2]$, where $(t_i\le t'_1\le t'_2\le t_j)$ such that

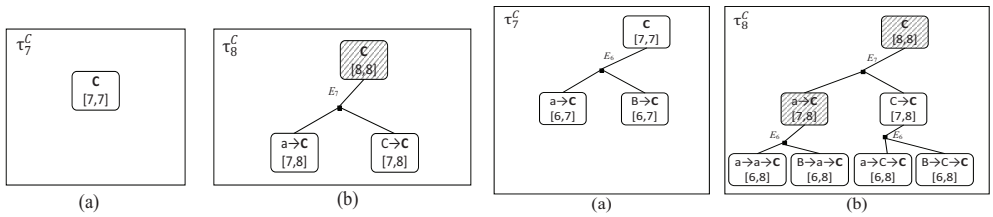
$$\begin{cases} t_1<t'_1, \text{ for Forward\_Scan;} \\ t'_2<t_2, \text{ for Backward\_Scan;} \end{cases} \tag{2}$$

Then with the help of general last minimal occurrence, we can design pruning techniques in Lemma 6.2 and avoid post-processing.

LEMMA 6.2. *Given the scanning time interval $[t_k,t_j]$, an existing node q in a peak episode tree $\tau^e_{t_p}$, where $k\le p\le j$, can be expanded to generate a new minimal occurrence of an episode α if:*

1: q stores a general last minimal occurrence within $[t_k,t_j]$;

2: q has no child node that represents other occurrence of α.



(a) (b) (a) (b)

(a) $\mathcal{T}$ after scanning E_7. (b) $\mathcal{T}$ after scanning E_6.

Fig. 7. Results of $\mathcal{T}$ when performing Backward_Scan for $\mathcal{P}^{\mathsf{C}}_{\langle 7,8\rangle}$.

With Lemma 6.2, only partial nodes can be further expanded to generate valid peak episode minimal occurrences. After the node expansion, the generated peak episode minimal occurrence is emitted for the subsequent computations (Line 10 and 11). After we traverse a node q, we need to check whether $q.mo$ is still a general last minimal occurrence according to current scanning direction, and change its status if it is not (Line 12).

Take the sub-sequence $\mathcal{P}^{\mathsf{C}}_{\langle 7,8\rangle}$ in $\mathcal{P}^{\mathsf{C}}$ in Figure 6 (d) as an example. Recall that the original event sequence is partitioned by setting $min\_sup = 3$ and $\delta = 3$. We perform Backward_Scan of PEM on such sub-sequence by scanning from E_8 to E_5 as the rightmost occurrence of C is $[8, 8]$ and the start point of $\mathcal{P}^{\mathsf{C}}_{\langle 7,8\rangle}$ is 5. Since there is an occurrence of C on E_8, we construct a new peak episode tree of τ^{C}_8 and add it to the tree list $\mathcal{T}$.

Next we scan E_7 and derive the updated $\mathcal{T}$ in Figure 7(a). There are two peak episodes trees after scanning E_7. Meanwhile, for occurrences of C, $[7, 7]$ is its general last minimal occurrence in the time interval $[7, 8]$ while $[8, 8]$ is not according to Definition 6.1. Hence we can avoid traversing the root of τ^{C}_8, (C:$[8, 8]$), which is shaded in the figure, during the following scanning in Backward_Scan.

Further, we continue to scan E_6 and maintain τ^{C}_7 and τ^{C}_8, respectively. Figure 7(b) demonstrates the results. After scanning E_6, since $[7, 8]$ is no longer the general last minimal occurrence of A→C in the time interval of $[6, 8]$, the node (A → C:$[7, 8]$) on τ^{C}_8 thus can be safely pruned.

Forward_Scan is similar as a mirror reflection of Backward_Scan. When we perform Forward_Scan, the event sets starting from the time stamp t_i and stopping at $t_j + \delta - 1$ are scanned sequentially, and the peak episode tree with *larger subscript* should be processed first. The processing for node expansion as well as management are similar with that in Backward_Scan. Remember that we do not construct new tree in Forward_Scan. Instead, we need to compare with the minimal occurrences generated by Backward_Scan to avoid duplication.

Figure 8 shows the results when finishing PEM over $\mathcal{P}^{\mathsf{C}}_{\langle 7,8\rangle}$ when setting $\delta = 3$. The two peak episode trees in such figure store all the peak episode minimal occurrences in $\mathcal{P}^{\mathsf{C}}_{\langle 7,8\rangle}$.

6.3 Discussions on PEM

Unlike traditional FEM algorithms which discover all episodes in sequence, PEM directly mines the defined peak episodes. Suppose a sub-sequence $\mathcal{P}^{e}_{\langle t_i,t_j\rangle}(\vec{S})$ has length of L and the average size of each event set in such sequence is M, and there are k occurrences of e in such sequence. Then the search space for traditional FEM algorithm is about $O(L\sum_{i=1}^{\delta} M^i)$. While the search space for PEM is about $O(2k\sum_{i=1}^{\delta-1} M^i)$. PEM explores a fraction of $\frac{2k}{L\cdot M}$ of the episodes produced by traditional FEM methods. For example, if $k = 1$, $L = 10$ and $M = 5$, PEM explores 4% of the search space of traditional FEM methods. As a consequence, such specialized miner might be more efficient compared to traditional methods. In addition, we can prune the search space during both of the scans based on Lemma 6.2, which further facilitates the efficiency of PEM.

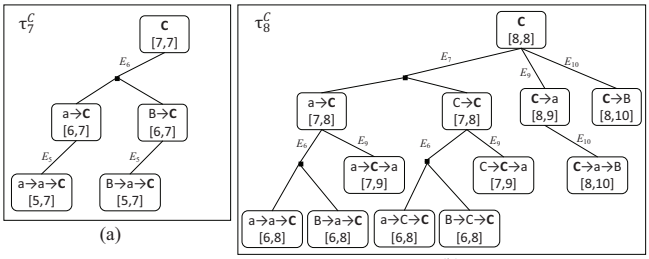


Fig. 8. The two peak episode trees after performing PEM on $\mathcal{P}^{\mathsf{C}}_{\langle 7,8\rangle}$. They all together store all the peak episode minimal occurrences on $\mathcal{P}^{\mathsf{C}}_{\langle 7,8\rangle}$.

7 MINING HIERARCHY-AWARE MAXIMAL AND CLOSED EPISODES

Despite the effectiveness of Frequent Episode Mining, it may discover potentially large number of redundant episodes which satisfy the thresholds. A standard approach to alleviate this problem is to produce only episodes are maximal [29] and closed [41]. In this section, we introduce how to extend LA-FEMH to support mining maximal and closed episodes in the context of event hierarchy.

7.1 Definitions

Given a maximal or closed pattern from itemsets, we can derive the information about whether its certain sub-patterns are frequent or not. That is based on the monotonicity of support: the support of a sub-pattern is not less than that of its super pattern.

However, as a frequency measure, minimal occurrence of episode is not monotonically decreasing in the sequence with simultaneous events [5, 41]. It means the supports of subepisodes may be smaller than that of their super episodes. As a consequence, we cannot directly adopt the concept of maximality and closedness in pattern mining of itemsets in our problem. Fortunately, for an episode in sequence with simultaneous events, the anti-monotonicity of frequency based minimal occurrence still holds with its corresponding consecutive subepisode as defined in Definition 7.1.

Definition 7.1. **Consecutive Subepisode.** Given an episode $\alpha = e_1 \rightarrow \cdots \rightarrow e_k$, a subepisode β of α is a consecutive subepisode, denoted by $\beta \tilde{\sqsubseteq} \alpha$, if $|\beta| = m$ and $\beta = e_i \rightarrow e_{i+1} \cdots \rightarrow e_{i+m-1}$ for some $i = 1, 2, \cdots, k - m + 1$.

For example, a→b and b→c are consecutive subepisodes of a→b→c, while a→c is not.

With the help of consecutive subepisode, we can use the property of *Support monotonicity on subepisode* [3] to deduce the relationship between their supports: let α and β be two episodes such that $\beta \tilde{\sqsubseteq} \alpha$, then we have $\mathrm{sp}(\beta) \geq \mathrm{sp}(\alpha)$.

Since we consider event hierarchy in this paper, we can find another support monotonicity between an episode and its *generalized episodes* defined below.

Definition 7.2. **Generalized episode.** We say an episode $\alpha = e_{\alpha_1} \rightarrow \cdots \rightarrow e_{\alpha_k}$ *directly generalizes* to an episode $\beta = e_{\beta_1} \rightarrow \cdots \rightarrow e_{\beta_k}$ (denoted by $\alpha \Rightarrow \beta$), if α and β have same length and there exists *at least* one subscript index j such that $e_{\alpha_j} \Rightarrow e_{\beta_j}$ and $e_{\alpha_i} = e_{\beta_i}$ for other $i \neq j$ and $1 \leq i \leq k$. Similar with the event hierarchy, we use $\Rightarrow^*$ to denote the transitive closure of $\Rightarrow$.

For example, $\mathsf{a}\rightarrow\mathsf{b}_{11}\rightarrow\mathsf{c}_2 \Rightarrow \mathsf{a}\rightarrow\mathsf{b}_1\rightarrow\mathsf{C}$, and $\mathsf{a}\rightarrow\mathsf{b}_{11}\rightarrow\mathsf{c}_2 \Rightarrow^* \mathsf{a}\rightarrow\mathsf{B}\rightarrow\mathsf{C}$.

THEOREM 7.3. *[Support monotonicity on hierarchy]. Let α and β be two episodes such that $\alpha \Rightarrow^* \beta$, then we have* $\mathrm{sp}(\beta) \geq \mathrm{sp}(\alpha)$.

PROOF. Since there may exist an episode α' such that $\alpha' \neq \alpha$ that can generalize to β, $\mathrm{sp}(\beta)$ will not be less than $\mathrm{sp}(\alpha)$ because $\mathrm{sp}(\beta) = |\mathrm{moSet}(\beta)| = |\mathrm{moSet}(\alpha) \cup \mathrm{moSet}(\alpha')|$. □

Based on the above observations, we give the formal definitions of hierarchy-aware maximal and closed episode.

Definition 7.4. **Hierarchy-aware maximal episode.** An episode α is *maximal* if α is frequent and there is no episode α' such that $\alpha \tilde{\sqsubseteq} \alpha'$ and α'' such that $\alpha'' \Rightarrow^* \alpha$ which are also frequent.

Definition 7.5. **Hierarchy-aware closed episode.** An episode α is *closed* if α is frequent and there is no episode α' such that $\alpha \tilde{\sqsubseteq} \alpha'$ and α'' such that $\alpha'' \Rightarrow^* \alpha$ of the same frequency.

With such definitions, we can recover all the frequent episodes in the event sequence by generating the corresponding consecutive subepisodes and generalized episodes. However, with only the maximal episode, we will lose the exact frequency information.

7.2 Extend LA-FEMH to Maximality and Closedness

Now we propose a reasonable extension on our framework to support mining both maximal and closed generalized episode. We refer to such extension as LA-FEMH+.

7.2.1 Preliminaries of LA-FEMH+. First we introduce the preliminaries used in LA-FEMH+. With the sequence partition strategy in LA-FEMH, we cannot test the maximality or closedness locally in each partition. For instance, in the running example, the episode $\mathsf{a}\rightarrow\mathsf{B}$ in the partition $\mathcal{P}^{\mathsf{B}}$ is frequent and local maximal when $min\_sup = 2$ and $\delta = 3$. But the global maximal episode should be $\mathsf{a}\rightarrow\mathsf{B}\rightarrow\mathsf{C}$ which is also frequent but in another partition $\mathcal{P}^{\mathsf{C}}$.

Hence we need to determine whether a local maximal (closed) episode is still global maximal (closed) or not. The episodes that are local maximal (closed) but not global maximal (closed) are called *specious* episodes. Suppose α is a specious episode, there could be some episodes which testify α is specious, and they are called *spectator* episodes. Thus $\mathsf{a}\rightarrow\mathsf{B}$ is a specious episode and $\mathsf{a}\rightarrow\mathsf{B}\rightarrow\mathsf{C}$ is its spectator in the previous example. We restrict the search to the set of *elementary spectators* as defined in Definition 7.6.

Definition 7.6. **Elementary spectator.** Let α be a specious episode, and its peak event is e. Then there is an elementary spectator episode of α' with peak event e' ($e \prec e'$) which satisfies:

(1) α' is frequent and local maximal (closed),

(2) $\alpha \tilde{\sqsubseteq} \alpha'$ and there is no intermediate episode α_1 with the peak event as e_1 that $\alpha \tilde{\sqsubseteq} \alpha_1 \tilde{\sqsubseteq} \alpha'$ and $e \prec e_1 \prec e'$ or,

(3) $\alpha' \Rightarrow^* \alpha$ and there is no intermediate episode α_2 with the peak event as e_2 that $\alpha' \Rightarrow^* \alpha_2 \Rightarrow^* \alpha$ and $e \prec e_2 \prec e'$.

For example, $\mathsf{a}\rightarrow\mathsf{B}\rightarrow\mathsf{C}$ is an elementary spectator of $\mathsf{a}\rightarrow\mathsf{B}$ (refer to (2) in Definition 7.6) and $\mathsf{b}_1\rightarrow\mathsf{a}$ is an elementary spectator of $\mathsf{B}\rightarrow\mathsf{a}$ (refer to (3) in Definition 7.6) in our running example.

Hence the key phase for LA-FEMH+ is to match the specious episodes with their elementary spectators to filter false positives. In more detail, for each local maximal (closed) episode α', we generate the set for which α' can potentially be an elementary spectator.

Specifically, by splitting at every peak event we have:

$$\alpha' = (e')^*\alpha_1(e')^+\alpha_2(e')^+ \cdots (e')^+\alpha_n(e')^* \tag{3}$$

where e' is the peak event of α' and every α_i whose peak event is relevant to e' is a consecutive subepisode of α'. $(e')^*$ ($(e')^+$) denotes 0 or more (1 or more) of occurrences of the peak event e'. We denote $\mathbb{C}_{\alpha'} = \{\alpha_1, \alpha_2, \cdots, \alpha_n\}$ in which α' might be the elementary spectator of every

contained α_i. In addition we have $\mathbb{G}_{\alpha'} = \{\beta \mid \alpha' \Rightarrow \beta\}$, where $\mathbb{G}_{\alpha'}$ is the episodes that α' can directly generalize to. Here α' could be the elementary spectator to every β in $\mathbb{G}_{\alpha'}$.

Finally, we can find

$$\mathbb{S}_{\alpha'} = \{\alpha'\} \cup \{e'\} \cup \mathbb{C}_{\alpha'} \cup \mathbb{G}_{\alpha'} \tag{4}$$

as the set of episodes for which α' can be an elementary spectator, as well as e' and α' itself. The episodes in $\mathbb{S}_{\alpha'}$ will be further used in our LA-FEMH+, and we will detail it later.

For example, $\mathbb{S}_{\mathsf{a}\to\mathsf{B}\to\mathsf{C}} = \{\mathsf{a}\to\mathsf{B}\to\mathsf{C},\ \mathsf{C},\ \mathsf{a}\to\mathsf{B}\}$. For another episode $\mathsf{b}_1\to\mathsf{a}$ in $\mathcal{P}^{\mathsf{b}_1}$, $\mathbb{S}_{\mathsf{b}_1\to\mathsf{a}} = \{\mathsf{b}_1\to\mathsf{a}, \mathsf{b}_1, \mathsf{a}, \mathsf{B}\to\mathsf{a}\}$.

7.3 Mining Hierarchy-aware Maximal/Closed Episode with LA-FEMH+

Next we detail the mining process of LA-FEMH+ on discovering hierarchy-aware maximal and closed episodes. Its pseudo-code is shown as Algorithm 3. It can be performed by an additional single MapReduce job after the mining procedure of LA-FEMH.

First in the local mining stage of LA-FEMH, we can alternate existing maximal episode miners such as PPS [29] or closed episode miners [41] to detect local frequent maximal/closed episodes in every partition. We can also derive the local frequent maximal/closed episodes by extending our PEM by postprocessings (Line 1). Next we *Map* over those local maximal/closed episodes as well as all frequent events in the g-list and compute the episode set $\mathbb{S}_{\alpha'}$ for every processing α' according to Eq. 4. We then form a key-value pair for every episode $\beta \in \mathbb{S}_{\alpha'}$: the key is β and the value, denoted by $\langle l, s\rangle$, is $\langle |\alpha'|, \mathrm{sp}(\alpha')\rangle$, where $|\alpha'|$ denotes the length of α' and $\mathrm{sp}(\alpha')$ denotes the support of α' (Line 5–7).

For example, since $\mathbb{S}_{\mathsf{a}\to\mathsf{B}\to\mathsf{C}} = \{\mathsf{a}\to\mathsf{B}\to\mathsf{C},\ \mathsf{C},\ \mathsf{a}\to\mathsf{B}\}$, the value of the three key-value pairs produced by $\mathbb{S}_{\mathsf{a}\to\mathsf{B}\to\mathsf{C}}$ is $\langle 3, 3\rangle$ since $\mathsf{a}\to\mathsf{B}\to\mathsf{C}$ is 3-episode and $\mathrm{sp}(\mathsf{a}\to\mathsf{B}\to\mathsf{C}) = 3$. Similarly, all the key-value pairs in $\mathbb{S}_{\mathsf{b}_1\to\mathsf{a}}$ is $\langle 2, 2\rangle$. Figure 9 (b) shows the results of such operation.

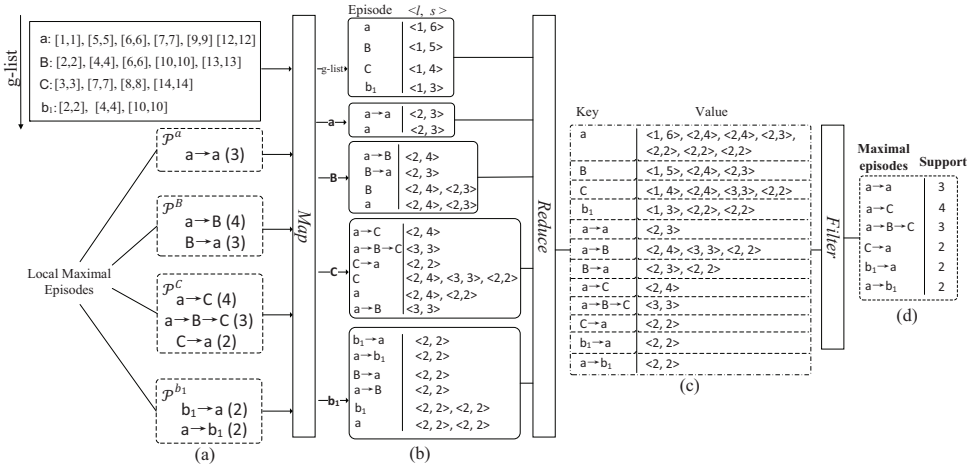


Fig. 9. Mining maximal generalized episode with LA-FEMH+ by setting $min\_sup = 2$ and $\delta = 3$.

Next we perform the *Reduce* on every key-value pair by the key β and derive a corresponding *attestant set* $\mathbb{A}_\beta = \{\langle l, s\rangle\}$ of (length-support)-pairs as input (Line 10). Then β will be a maximal generalized episode if $|\mathbb{A}_\beta| = 1$. In this case, there is no spectator episode testifying β is specious. Based on this, we can obtain the global hierarchy-aware maximal episodes (Line 13 and 14).

For example, Figure 9 (c) and (d) demonstrate the results of corresponding attestant sets and global hierarchy-aware maximal episodes in the running example, respectively. There are all together 6 global maximal episodes since their attestant sets only contain a single value.

ALGORITHM 3: LA-FEMH+ for Maximality and Closedness

Input: $\vec{S}$: the event sequence;
min_sup: threshold of minimum support;
δ: threshold of maximum occurrence window;
$type \in \{\text{maximal, closed}\}$;

$\mathcal{F}^{type} \leftarrow$ mining local frequent $type$ episodes in each partition $\mathcal{P}^e$
Map($\{\mathcal{F}^{type} \cup g\text{-list}\}$)
begin
  foreach $\alpha' \in \{\mathcal{F}^{type} \cup g\text{-}list\}$ **do**
    compute $\mathbb{S}_{\alpha'}$ according to Eq. 4
    foreach $\beta \in \mathbb{S}_{\alpha'}$ **do**
      Emit $(\beta, \langle|\alpha'|, \text{sp}(\alpha')\rangle)$
Reduce($\beta, \langle l, s\rangle$)
begin
  $\mathbb{A}_\beta \leftarrow$ the attestant set of β as $\{\langle l, s\rangle\}$
  switch $type$ **do**
    case *maximal:*
      if $|\mathbb{A}_\beta|$=*1* **then**
        Emit (β, s)
    case *closed:*
      $\langle l^+, s^+\rangle \leftarrow$ pair in $\mathbb{A}_\beta$ with largest support s; resolve tie by taking maximum length l
      if $|\beta| = l^+$ *and* $\langle l^+, s^+\rangle$ *is unique* **then**
        Emit (β, s^+)

To discover global hierarchy-aware closed episodes, we sort the elements, namely $\langle l, s\rangle$, in $\mathbb{A}_\beta$ and look for the element with the *largest* support, denoted by $\langle l^+, s^+\rangle$. We break ties by selecting the pair of maximal length (Line 16). Then β can be regarded as a hierarchy-aware closed episode if $\langle l^+, s^+\rangle$ is *unique* and $|\beta| = l^+$, and we emit such episode if it satisfies the criterions (Line 17 and 18). If there are more than one $\langle l^+, s^+\rangle$ in $\mathbb{A}_\beta$, it means there exists some episode α' such that $\alpha' \Rightarrow \beta$ and $\text{sp}(\alpha') = \text{sp}(\beta)$. $|\beta| \neq l^+$ only happens as $|\beta| < l^+$. It means there exists other episode α' such that $\beta \tilde{\sqsubseteq} \alpha'$, $|\alpha'| = l^+$ and $\text{sp}(\alpha') = \text{sp}(\beta)$.

For example, A→B and B→A become closed though they are not maximal in the running example. For A→B, $\langle l^+, s^+\rangle = \langle 2, 4\rangle$ and the length of A→B is equal to l^+ which is 2. For B→A, we have $\langle l^+, s^+\rangle = \langle 2, 3\rangle$ and the length of B→A is 2. While C is not closed since we find $\langle l^+, s^+\rangle$=$\langle 2, 4\rangle$ with $|\text{C}| < l^+$. The detailed information can be referred to Fig. 9 (c).

8 EXPERIMENTS

In this section, we present the results of our experimental studies on both synthetic and real-world datasets. We verify the efficiency and effectiveness of the proposed techniques in LA-FEMH including the sequence partitioning strategy as well as the local miner PEM. Besides, we also investigate the effect of event hierarchy and LA-FEMH+. We finally study the scalability of LA-FEMH.

8.1 Experimental Settings

Environment All the experiments are performed on Apache Spark cluster consisting of ten HP ProLiant DL-580 servers, each with 4 Intel Xeon E5-2620 CPUs and 128G gigabytes memory, running on Linux CentOS 6.4. The nodes in the cluster are connected via a 1000Mb Ethernet. The version of Apache Spark is 1.6.0 which runs on JDK 1.7. In the cluster, one node acted as master node and the others acted as worker nodes. The number of concurrent executors was set to 36 and the

executor memory was set to 16GB. All of the compared algorithms are implemented in Scala (2.10.6). We run every experiment 3 times and report the average result.

Datasets. We adopt one synthetic dataset and two real-world datasets to perform experimental studies. Table 1 shows the their statistics.

Table 1. Statistical information of datasets. The second to the fifth columns, namely "Seq. Len." to "Total event", describe the statistics of the event sequence; the rest, namely "Leaf event" to "Max fan-out" describe the information about the event hierarchy.

Dataset	Seq. Len.	Max. events per ev. set	Avg. events per ev. set	Total events	Leaf events	Root events	Hie. levels	Avg. fan-out	Max fan-out
SYN	5,000	50	4.1	20,114	50	NA	NA	NA	NA
STK	9,334	2,598	600.3	5,603,200	11,443	116	2	95.9	271
FIC	9,600,259	1	1	9,600,259	75,286	34	4	1.65	17518

The **SYN** dataset refers to a synthetic dataset which contains an event vocabulary, denoted as $\mathcal{E}$, whose size is 50 with no event hierarchy. The sequence length is fixed as 5000. The size of event set is determined by a function of $\sin(t \mod 90) \times |\mathcal{E}|$ where t refers to the time index and $|\mathcal{E}|$ denotes the size of event vocabulary. For the events in each event set, we extracted them following a uniform distribution. The dataset is skewed and thereby challenges partitioning strategies in algorithms.

The **STK** dataset consists of sequences from China stock market. It includes 2,780 stocks with their price information from December 19th, 1990 to July 14th, 2016. The events for a stock were generated based on its daily closing price followed by preprocessing method in [5]. In particular, the increase ratio of price between two consecutive trading days is calculated and discretized into four levels, namely UH (up high), UL (up low), DL (down low) and DH (down high), by thresholds. Thus each stock can generate one of the above events every day unless suspending trading. For event hierarchy, we took the industry sector taxonomy from finance organization to construct a flat hierarchy of stocks in which all the stocks are grouped into 29 industry sectors. Combining with event types we generated 116 root events.

The **FIC** dataset is a subset of Gutenberg Dataset [23], which contains 3,036 English books written by 142 authors. From this dataset, we selected all books written by George Alfred Henty, who is the most prolific author with 89 listed fictions in the dataset. Among these fictions, we treated each word as an event and built a single sequence by concatenating each sentence of every his fiction one by one. For hierarchies, we adopted a syntactic hierarchy in which a word first generalizes to its lowercase form, which generalizes to its lemma, which in turn generalizes to its POS tag derived by Stanford CoreNLP parser [7]. For example, the word "Learning"⇒"learning"⇒"learn"⇒"VERB".

8.2 Overall Runtime

To begin with, we evaluate the overall performance of LA-FEMH and compare it with the timestamp-based approach (denoted by **TBA** hereafter) and event-projected approach (denoted by **EPA** hereafter) discussed in Section 4. Since all the compared methods will always have the same results given the same input sequences, we thus focus on the evaluate their efficiency. As MESELO [5] is a competitive method among the local miners we compared (later shown in Section 8.4), here we adopt it as the local mining algorithm of the baselines, i.e. TBA and EPA. We tune both min_sup and δ, and the overall execution time are exhibited in Figure 10. Note that TBA failed to finish running within 12 hours, so we leave these cases as "NA" in the figure. We observe that LA-FEMH significantly outperforms the baselines on both synthetic and real-world datasets. It achieves a speedup more than an order of magnitude compared with TBA when $min\_sup = 0$ and $\delta = 5$ on SYN dataset as shown

[7] http://nlp.stanford.edu/software/corenlp.shtml

in Figure 10 (a). We argue that mining such sequence with lenient parameter settings is challenging even for short length and small event alphabet. For real world datasets, LA-FEMH outperforms other methods by at least three times.

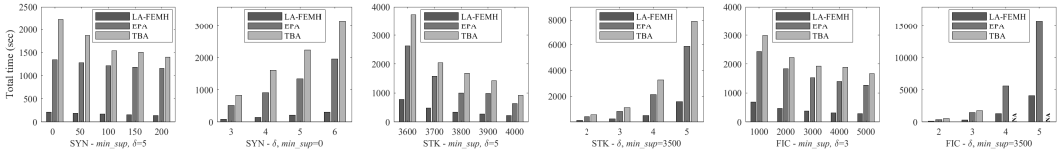


Fig. 10. Performance of overall runtime.

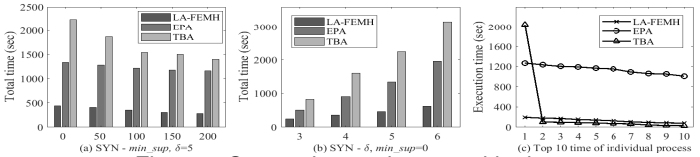


Fig. 11. Comparisons about partitioning.

8.3 Effectiveness of Partitioning

Next we evaluate the effectiveness of sequence partitioning in LA-FEMH.

Firstly, we use the same local miner for all compared methods and conduct the same experiments on the SYN dataset to demonstrate the effectiveness of partitioning. From the results shown in Figure 11 (a) and (b), we can see that even with the same local mining algorithm, our proposed partitioning method still has clearly advantages compared with the competitors. For example, when $min\_sup = 0$ and $\delta = 5$, the execution time of LA-FEMH, EPA and TPA is 440, 1340 and 2240 seconds, respectively. Recall that the optimized rewrites in sequence partitioning of LA-FEMH compress both the length and event set size in each sub-sequence, it derives clearly time saving.

Secondly, we evaluate the consistency in sequence partitioning by looking at individual process of the compared methods. In particular, we investigate the case of $min\_sup = 0$ and $\delta = 5$ on SYN dataset in the overall running time evaluation. We rank the execution time of each process in a descending order and visualize the execution time of the top-10-slow processes in Figure 11 (c). From such figure, we can clearly see a sharp value in TBA, i.e., 2051 seconds for the slowest process. EPA is more consistent than TBA with the slowest individual process time of 1272 seconds. However, all individual processes of EPA are much slower than that of LA-FEMH due to the lack of optimized rewrites. The trend of execution time of each process in LA-FEMH, on the other hand, is much flatter, and the elapsed time of its slowest individual process is 195 seconds. There is no doubt the overall runtime of a MapReduce algorithm is highly related to its slowest individual process. For instance, in our case, LA-FEMH terminates in 210 seconds, while the running time of EPA and TPA are 1340 and 2240 seconds, respectively. Another interesting finding is the other processes of TBA, except the slowest one, are even finished faster than the processes of EPA and LA-FEMH. That is because both EPA and LA-FEMH would copy event occurrences into different sub-sequences while TPA does not. It might indicate a potential insight that a balanced workload is more essential than the average data amount for a distributed episode mining algorithm.

8.4 Efficiency of PEM

In our next set of experiments, we investigate the efficiency of the proposed local miner PEM. We replace the local miner of the LA-FEMH framework by several state-of-the-art methods including MESELO [5], UP-Span [44] and DFS [1]. All of them compute minimal occurrence of episodes for purpose or at least as one of their key phases. We extend some compared methods to support our problem and guarantee a fair comparison. All compared methods take δ as a time constraint

parameter and all the intermediate computational results are stored in main memory. Figure 12 shows the results. We observe that PEM hold a clear advantage compared with others. PEM obtains at least $2\times$ speedup compared with the strongest competitor MESELO among all compared approaches. Hence MESELO was taken as the local mining approach of the baselines in Section 8.2. For DFS and UP-Span, DFS has advantages in sequence with much simultaneous events while UP-Span performs better in flatter but long sequence. We also investigate the number of discovered episode occurrences generated by PEM and other local miners on FIC dataset by fixing min_sup=1000 and δ=3. We observe that PEM explores only 30.66% and 26.88% of searching space compared with MESELO and UP-Span, respectively.

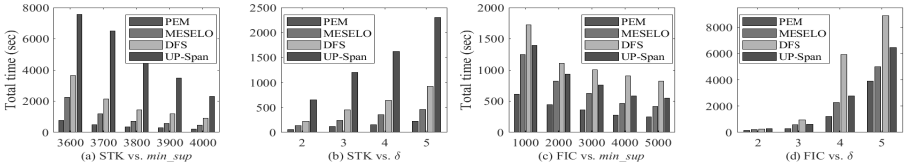


Fig. 12. Execution time with different local miners.

8.5 Efficiency over Hierarchy

Next we examine the impact of event hierarchies on efficiency. In particular, we conceal the event hierarchy and repeat the experiments in Section 8.2 on both two real datasets. We observe that LA-FEMH is more scalable to the hierarchies as shown in Figure 13. Both TBA and EPA surprisingly have rather conspicuous increase in execution time when taking the hierarchy structures into account. For example, by adding the lexical hierarchy in the FIC dataset, the elapsed time of TBA increases from 100 to 1762 seconds, which is by 17.6 times. EPA, which has more than 21.5 times increase, terminates in 1464 seconds with hierarchy. While its running time without hierarchy is 68 seconds. For our LA-FEMH, its execution time with/without hierarchy is 294 and 42 seconds, respectively. From this comparison, we can observe that our proposed LA-FEMH is much more scalable to the event hierarchy, and its advantages become more obvious as the event hierarchy goes deeper.

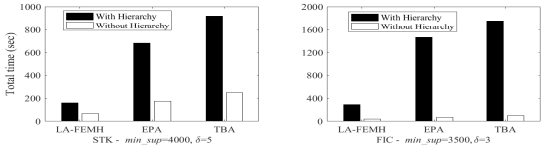


Fig. 13. Execution time with/without event hierarchy.

8.6 Effectiveness of Hierarchy

We next investigate the effectiveness of discovered episodes over event hierarchies. We report the percentage of *non-trivial* output episodes to judge the benefit from event hierarchy and demonstrate examples of interesting patterns with considering event hierarchies. An output episode is *trivial*, if it can be generated by traditional FEM algorithms (which ignores event hierarchies). We demonstrate the percentage of non-trivial episodes mined from FIC dataset on different minimum support threshold settings in the third column of Table 2. Some non-trivial episodes include: "**one**→**of**→**the**→**ADJ**→**NOUN**", "**the**→**NOUN**→**had**→**been**", "**be**→**able**→**to**→**VERB**", and etc. No specializations of these episodes were frequent in the input data. We argue these non-trivial patterns are useful for understanding the writing style of the author.

For the STK dataset, we observe that more than 94.08% (setting $min\_sup = 500$ and $\delta = 3$) of the output episodes are non-trivial. Some non-trivial episodes on STK dataset suggest "**Real Estate-DH**" → "**SZ:000012-DH**" (a well-known building materials company); "**Oil and Gas-UH**"

→ "**SZ:000550-DH**" (an automobile manufacturer); "**SH:600809-UH**" (a leading liquor-making enterprise) → "**Food and Beverage-UH**". We believe these episodes have practical meanings as we observe that real estate industry has some positive impacts for building materials companies from the first non-trivial episode. As is shown in the second pattern, the correlation between oil, gas industry and automobile companies is negative. What is more, as shown in the third example, we could find some leading stocks for an industry sector. They are in accordance with events in the real world. Without the consideration of hierarchies, it is difficult to perceive such correlation and meanwhile constructing concrete trading strategies. This serves an essential evidence to defense our motivation of involving hierarchy information in this work.

8.7 Effectiveness of LA-FEMH+

Then, we study the effect of LA-FEMH+ for mining hierarchy-aware maximal and closed episodes. In more detail, we first run both LA-FEMH and LA-FEMH+ to mine all, maximal and closed episodes on different parameter settings, respectively. For mining local maximal/closed episodes, we extend our PEM by postprocessings. Then we compute the percentage of closed and maximal episodes compared with all frequent episodes to demonstrate the effect of LA-FEMH+. We show the results in Table 2. We also conceal the event hierarchy of FIC and perform the same investigation. The results are shown in the same table. From the table we can observe the effectiveness of LA-FEMH+ for reducing redundancies, and it benefits more from deeper hierarchies or lower support. To the best of our knowledge, we are the first attempt for direct mining of hierarchy-aware maximal and closed episodes in the context of event hierarchies. We meanwhile visualize the corresponding running time of LA-FEMH and LA-FEMH+ on the same table. From the table we can clear see LA-FEMH+ is comparable to LA-FEMH in efficiency, however, it produces non-redundant output.

Table 2. Output Statistics.

Dataset	min_sup	Non-trivial (%)	Closed (%)	Maximal (%)	Original (Sec)	Closed (Sec)	Maximal (Sec)
FIC, δ=3 (with Hierarchy)	5000	92.25	95.02	54.85	158.2	162.2	162
	1500	91.99	93.61	51.80	370	377.8	375.4
	150	90.08	90.71	44.60	1540.6	1564.8	1562.6
FIC, δ=3 (without Hierarchy)	5000	N/A	100	84.94	23.4	24.8	24.4
	1500	N/A	100	82.65	47.6	50.8	50.6
	150	N/A	99.99	71.30	165.6	170.8	169

8.8 Scalability

Eventually we study the scalability of LA-FEMH on the SYN dataset as we add more worker nodes and/or increased the input data size. The sequence is simply duplicated to generate different size of data. The number of event types and the event hierarchy (remember this is no event hierarchy in the SYN dataset) are fixed in this evaluation. For the parameter settings, we set $min\_sup = 0$ and $\delta = 3$ for this investigation.

We first investigate the performance of LA-FEMH by varying the input data size. To this end, we use 100%, 200% and 300% of the dataset as the input. Figure 14 (a) demonstrates the results, and we observe LA-FEMH is robust in dealing with increasing amounts of data.We validate strong scalability by fixing data size and varying computing node numbers. In particular, we fix the data size (100% of SYN) and vary the number of nodes from 3 to 6 and 9. Figure 14 (b) shows that LA-FEMH exhibits a good speedup as we increase the number of nodes. Finally we study weak scalability that we increase the input data size and simultaneously add more computing nodes. Specifically, we increase the size of input data (100%, 200% and 300%) and the number of computing nodes (3, 6 and 9). The result is demonstrated as Fig. 14 (c). As we double both data and computing resources, the total time ideally remains constant. Therefore, we can conclude that LA-FEMH achieves a good scaleup.

To sum up, based on the evaluation results, our proposed LA-FEMH has a competitive scalability in terms of the number of simultaneous event sets in the sequence.

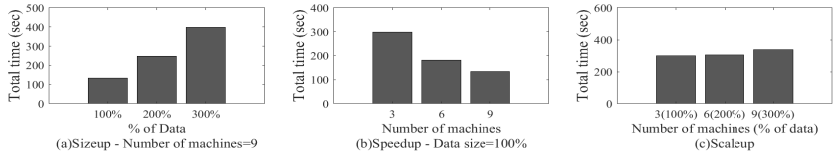


Fig. 14. Scalability of LA-FEMH.

9 CONCLUSIONS

In this paper, we investigated the problem of mining episodes from complex event sequence with hierarchies and proposed a distributed and scalable algorithm LA-FEMH. We first devised an effective partition strategy with optimized rewrites to decompose the large sequence into balanced slices. Then a specialized local mining algorithm called PEM was designed to efficiently mine every partition in parallel. LA-FEMH was then extended to mine both hierarchy-aware maximal and closed episodes in the context of event hierarchy for producing non-redundant output. Experimental results on both synthetic and real-world datasets showed that the proposed approaches demonstrated significantly performance gain in both efficiency and scalability and unearthed non-trivial patterns.

ACKNOWLEDGEMENTS

The research work is partially supported by the National Key Research and Development Program of China under Grant No. 2017YFB1002104, the National Natural Science Foundation of China under Grant No. U1811461, 61602438, 91846113, the CCF-Tencent Rhino-Bird Young Faculty Open Research Fund No. RAGR20180111. This work is also funded in part by Ant Financial through the Ant Financial Science Funds for Security Research. Xiang is also supported by Youth Innovation Promotion Association CAS. Jin Wang is the corresponding author.

REFERENCES

[1] Avinash Achar, A Ibrahim, and PS Sastry. 2013. Pattern-growth based frequent serial episode discovery. *DKE* (2013).
[2] Avinash Achar, Srivatsan Laxman, and PS Sastry. 2012. A unified view of the apriori-based algorithms for frequent episode discovery. *KAIS* (2012).
[3] Avinash Achar, Srivatsan Laxman, Raajay Viswanathan, and PS Sastry. 2012. Discovering injective episodes with general partial orders. *DMKD* (2012).
[4] Xiang Ao, Yang Liu, Zhen Huang, Luo Zuo, and Qing He. 2018. Free-Rider Episode Screening via Dual Partition Model. In *DASFAA*.
[5] Xiang Ao, Ping Luo, Chengkai Li, Fuzhen Zhuang, and Qing He. 2015. Online Frequent Episode Mining. In *ICDE*.
[6] Xiang Ao, Ping Luo, Chengkai Li, Fuzhen Zhuang, Qing He, and Zhongzhi Shi. 2018. Discovering and Learning Sensational Episodes of News Events. *Information Systems* (2018).
[7] Xiang Ao, Ping Luo, Jin Wang, Fuzhen Zhuang, and Qing He. 2018. Mining precise-positioning episode rules from event sequences. *IEEE TKDE* (2018).
[8] Mikhail Atallah, Wojciech Szpankowski, and R Gwadera. 2004. Detection of significant sets of episodes in event sequences. In *ICDM*.
[9] Marina Barsky, Sangkyum Kim, Tim Weninger, and Jiawei Han. 2011. Mining flipping correlations from large datasets with taxonomies. *VLDB* (2011).
[10] Kaustubh Beedkar and Rainer Gemulla. 2015. LASH:Large-Scale Sequence Mining with Hierarchies. In *SIGMOD*.
[11] Bouchra Bouqata, Christopher D Carothers, Boleslaw K Szymanski, and Mohammed J Zaki. 2006. Vogue: a novel variable order-gap state machine for modeling sequences. In *PKDD*.
[12] Alexandra M Carvalho, Arlindo L Oliveira, Ana T Freitas, and Marie-France Sagot. 2004. A parallel algorithm for the extraction of structured motifs. In *SAC*.
[13] Gemma Casas-Garriga. 2003. Discovering unbounded episodes in sequential data. In *PKDD*.
[14] Shengnan Cong, Jiawei Han, Jay Hoeflinger, and David Padua. 2005. A sampling-based framework for parallel data mining. In *PPoPP*.
[15] Shengnan Cong, Jiawei Han, and David Padua. 2005. Parallel mining of closed sequential patterns. In *KDD*.
[16] Lina Fahed, Armelle Brun, and Anne Boyer. 2018. DEER: Distant and Essential Episode Rules for early prediction. *ESWA* (2018).

[17] Robert R Grauer, Nils H Hakansson, and Frederick C Shen. 1990. Industry rotation in the US stock market: 1934–1986 returns on passive, semi-passive, and active strategies. *Journal of Banking & Finance* (1990).
[18] Jiaqi Gu, Jin Wang, and Carlo Zaniolo. 2016. Ranking support for matched patterns over complex event streams: The CEPR system. In *ICDE*. 1354–1357.
[19] Jiawei Han and Yongjian Fu. 1995. Discovery of multiple-level association rules from large databases. In *VLDB*.
[20] Jiawei Han, Jian Pei, Yiwen Yin, and Runying Mao. 2004. Mining frequent patterns without candidate generation: A frequent-pattern tree approach. *DMKD* (2004).
[21] Kuo-Yu Huang and Chia-Hui Chang. 2008. Efficient mining of frequent episodes from complex sequences. *Information Systems* (2008).
[22] Klaus Julisch. 2002. Data mining for intrusion detection. In *Applications of data mining in computer security*.
[23] Shibamouli Lahiri. 2014. Complexity of Word Collocation Networks: A Preliminary Structural Analysis. In *EACL Workshop*.
[24] Srivatsan Laxman, PS Sastry, and KP Unnikrishnan. 2007. A fast algorithm for finding frequent episodes in event streams. In *KDD*.
[25] Zhen Liao, Daxin Jiang, Enhong Chen, Jian Pei, Huanhuan Cao, and Hang Li. 2011. Mining concept sequences from large-scale search logs for context-aware query suggestion. *ACM TIST* (2011).
[26] Yuri Lin, Jean-Baptiste Michel, Erez Lieberman Aiden, Jon Orwant, Will Brockman, and Slav Petrov. 2012. Syntactic annotations for the google books ngram corpus. In *ACL*. 169–174.
[27] Yu Feng Lin, Cheng Wei Wu, Chien Feng Huang, and Vincent S. Tseng. 2015. Discovering utility-based episode rules in complex event sequences. *ESWA* (2015).
[28] Ling Luo, Xiang Ao, Feiyang Pan, Jin Wang, Tong Zhao, Ningzi Yu, and Qing He. 2018. Beyond Polarity: Interpretable Financial Sentiment Analysis with Hierarchical Query-driven Attention. In *IJCAI*. 4244–4250.
[29] Xi Ma, HweeHwa Pang, and Kian-Lee Tan. 2004. Finding constrained frequent episodes using minimal occurrences. In *ICDM*.
[30] Heikki Mannila and Hannu Toivonen. 1996. Discovering Generalized Episodes Using Minimal Occurrences.. In *KDD*.
[31] Heikki Mannila, Hannu Toivonen, and A Inkeri Verkamo. 1997. Discovery of frequent episodes in event sequences. *DMKD* (1997).
[32] Iris Miliaraki, Klaus Berberich, Rainer Gemulla, and Spyros Zoupanos. 2013. Mind the Gap: Large-Scale Frequent Sequence Mining. In *SIGMOD*.
[33] Ndapandula Nakashole, Martin Theobald, and Gerhard Weikum. 2011. Scalable knowledge harvesting with high precision and high recall. In *WSDM*. 227–236.
[34] Anny Ng and Ada Wai-Chee Fu. 2003. Mining Frequent Episodes for Relating Financial Events and Stock Trends. In *PAKDD*.
[35] Debprakash Patnaik, Patrick Butler, Naren Ramakrishnan, Laxmi Parida, Benjamin J Keller, and David A Hanauer. 2011. Experiences with mining temporal event sequences from electronic medical records: initial successes and some challenges. In *KDD*.
[36] Majed Sahli, Essam Mansour, and Panos Kalnis. 2013. Parallel motif extraction from very long sequences. In *CIKM*.
[37] Ramakrishnan Srikant and Rakesh Agrawal. 1995. Mining generalized association rules. In *VLDB*.
[38] Ramakrishnan Srikant and Rakesh Agrawal. 1996. Mining sequential patterns: Generalizations and performance improvements. In *EDBT*.
[39] Nikolaj Tatti. 2014. Discovering episodes with compact minimal windows. *DMKD* (2014).
[40] Nikolaj Tatti. 2015. Ranking episodes using a partition model. *DMKD* (2015).
[41] Nikolaj Tatti and Boris Cule. 2011. Mining closed episodes with simultaneous events. In *KDD*.
[42] Nikolaj Tatti and Jilles Vreeken. 2012. The long and the short of it: Summarising event sequences with serial episodes. In *KDD*.
[43] KP Unnikrishnan, Basel Q Shadid, PS Sastry, and Srivatsan Laxman. 2009. Root cause diagnostics using temporal data mining. US Patent 7,509,234.
[44] Cheng-Wei Wu, Yu-Feng Lin, S Yu Philip, and Vincent S Tseng. 2013. Mining High Utility Episodes in Complex Event Sequences. In *KDD*.
[45] Mohammed J. Zaki. 2001. Parallel Sequence Mining on Shared-Memory Machines. *JPDC* (2001).
[46] Chao Zhang, Jiawei Han, Lidan Shou, Jiajun Lu, and Thomas La Porta. 2014. Splitter: Mining fine-grained sequential patterns in semantic trajectories. In *VLDB*.
[47] Yong Zhang, Jiacheng Wu, Jin Wang, and Chunxiao Xing. 2019. A Transformation-based Framework for KNN Set Similarity Search. *IEEE Trans. Knowl. Data Eng.* (2019).