Online Frequent Episode Mining

Xiang Ao [#†1], Ping Luo [#2], Chengkai Li [*3] , Fuzhen Zhuang [#1], Qing He [#1]

[#] Key Lab of Intelligent Information Processing of Chinese Academy of Sciences (CAS),
Institute of Computing Technology, CAS, Beijing, China
[1] {aox,zhuangfz,heq}@ics.ict.ac.cn [2] luop@ict.ac.cn

[*] *University of Texas at Arlington, USA*
[3] cli@uta.edu

[†] *University of Chinese Academy of Sciences, Beijing, China*

***Abstract*—Frequent episode mining is a popular framework for discovering sequential patterns from sequence data. Previous studies on this topic usually process data offline in a batch mode. However, for fast-growing sequence data, old episodes may become obsolete while new useful episodes keep emerging. More importantly, in time-critical applications we need a fast solution to discovering the latest frequent episodes from growing data. To this end, we formulate the problem of Online Frequent Episode Mining (OFEM). By introducing the concept of *last episode occurrence* within a time window, our solution can detect new *minimal episode occurrences* efficiently, based on which all recent frequent episodes can be discovered directly. Additionally, a trie-based data structure, *episode trie*, is developed to store minimal episode occurrences in a compact way. We also formally prove the soundness and completeness of our solution and analyze its time as well as space complexity. Experiment results of both online and offline FEM on real data sets show the superiority of our solution.**

I. Introduction

Frequent episode mining (FEM) techniques are broadly conducted to analyze data sequences in the domains of telecommunication [29], [30], manufacturing [20], [21], finance [33], [18], biology [5], [18], system log analysis [44], [18], and news analysis [3]. An episode (also known as *serial episode*) is usually defined as a totally ordered set of events, and the frequency of an episode is the measure of how often it occurs in a sequence. FEM aims at identifying all the frequent episodes whose frequencies are larger than a user-specified threshold.

There are many ways to define the frequency of an episode. Existing measures include window-based frequency [30], [33], minimal occurrence [29], [41], [18], [27], non-overlapped occurence [22], [1], [44], non-interleaved occurrence [19] and total frequency [1]. Since minimal occurrence can capture the most intense correlation between events, we adopt this frequency measure for episode mining.

Previous studies on FEM mostly process data *offline* in a batch mode. Usually, masses of historical data are provided, and the mining process may last for hours or even days. Two characteristics make most existing FEM solutions time-consuming: 1) The anti-monotonicity property may fail to hold for episode frequency [2]. For instance, the frequency of a sub-episode may be less than that of the super-episode if *minimal occurrence* is used to measure the episode frequency. 2) Testing whether an episode occurs in a sequence is an NP-complete problem [38].

In this paper, we study *online* frequent episode mining where the sequence of events continuously grows. In such a scenario, old episodes may become obsolete while newly-emerging episodes may become valuable. More importantly, for time-critical applications we need efficient methods to find recent, frequent episodes from the growing sequence. This online mining problem is motivated by some real-world applications, and we describe two of such applications below.

The first application is High Frequency Trading (HFT) in quantitative finance. HFT is a type of algorithmic trading that uses sophisticated models to rapidly trade securities [14]. After transforming price and volume series into a sequence of events, episode mining can be applied in guiding trading decisions. It is also worth emphasizing that data-processing speed and the ability of adapting to market variations are key factors to the success of HFT. A recent report shows that a high-frequency trader holds stocks for only 22 seconds in average [9] since only the swiftest HFT operations can benefit from existing opportunities [23]. Therefore, episode mining for HFT needs an efficient online method to handle fast-growing data and identify the freshest patterns for prediction.

Another application is predictive maintenance of data centers. Predictive maintenance strives to realize equipment failures beforehand, for preventing unanticipated equipment downtime and promoting service quality for operations staff [36]. By mining equipment event logs which contain rich operational information, we can generate the prediction model to report risky alarms or repair possible malfunctions automatically. In most cases the log data arrive within seconds. Also, with the growth of log data and the variations of tasks and users, meaningful patterns may change dramatically. Hence, we also need an fast episode mining method to discover the latest patterns for predictive maintenance.

In summary, real-world applications present the following three key challenges in online frequent episode mining:

- *Fast-growing data.* A long sequence of events is fast-growing, and new events often arrive at an interval of seconds.
- *Recency effect.* Only the freshest patterns from recent events are of interests.
- *Time-critical analysis*. The mining process is required to be fast and responsive. Data-processing speed is critical in these

applications since a delay may lead to drastic loss or even disaster.

A naive solution to the online episode mining problem is to continuously perform a batch-mode episode mining algorithm over the sequences in current time windows. Clearly, this method allows much room for improvement in time efficiency since it incurs lots of repeated computations in consecutive rounds of mining and fails to reuse the results across different rounds. Therefore, we aim to develop an efficient method that fully leverages the immediate results from the last round. It must tackle the following challenges.

- Infrequent events at the current moment may become frequent in the future. Therefore, we cannot simply discard infrequent events. Instead, we must keep all events at all time. This requirement drastically increases the complexity of the problem.
- Since we have to retain all events in the mining process, the intensive computation will generate lots of episode occurrences. Thus, we are in need of a compact and effective data structure for storing all such episode occurrences.
- Efficiently mining all minimal occurrences of episodes also becomes a big challenge over the growing sequence.

In this paper, we propose an algorithm named MESELO (Mining frEquent Serial Episode via Last Occurrence) for online frequent episode mining. We design a sophisticated data structure, *episode trie*, to compactly store all minimal occurrences of episodes. We also introduce the concept of *last episode occurrence*. Given a time-window, there is no occurrence of the same episode after its last occurrence. An important property based on this concept is that appending any new event (at a new time point) to a last episode occurrence can generate a new minimal episode occurrence. In other words, we can directly generate all the new minimal episode occurrences for the coming events based on the last episode occurrences. Thereby, it enables an efficient online method to identify each minimal episode occurrence, based on which we count the frequency of each episode and find the frequent ones. In this study, we also formally prove the correctness and completeness of the proposed method, and we analyze its time and space complexity. Experiment results of both online and offline batch modes on ten real-world data sets demonstrate significant superior time efficiency of the proposed method over the baselines. In addition, we compare our method and some state-of-the-art batch mode FEM methods based on minimal-occurrence. They have not been comprehensively compared in prior studies.

The remainder of this paper is organized as follows. Section II discusses related work. Section III presents concept definitions and problem statement. In Section IV, we overview the framework for online frequent episode mining. In Section V, we describe the details of the MESELO algorithm and prove its soundness and completeness. In Section VI, we analyze the time and space complexity of the proposed algorithm. Section VII presents experiment results. Section VIII concludes the paper and discuss the future work.

II. Related Work

We are not aware of prior work on the problem of online frequent episode mining. However, there are several studies related to this task, including frequent episode mining and online frequent pattern mining.

Frequent episode mining [1]−[4], [6], [11], [18], [22], [26], [27], [29]−[31], [33]−[35], [38]−[41], [43], [44] on event sequences is an important data mining problem for various forms of data, such as alarm sequences in telecom networks [30], [35], web navigation logs [6], [30], time-stamped fault reports in car manufacturing plants [22], sales transactions [4], [41], stock data [18], [33], news [3], and so on [34], [39]. Depending on different applications, various 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 definitions.

Given a particular frequency definition, frequent episode mining algorithms fall into two categories: breadth-first enumeration (also known as apriori-based) methods and depth-first enumeration (also known as pattern-growth) methods. The breadth-first algorithms involve two main steps: candidate generation and frequency counting. Candidate generation is usually improved based on anti-monotonicity or some more restricted anti-monotone properties of frequency definitions. However, anti-monotonicity fails to hold for minimal occurrences [2]. Particularly, the frequency of a sub-episode may be less than that of its super-episode if minimal occurrence is used in defining episode frequency. We provide an example to explain such an observation in Section III. The depth-first enumeration algorithms discover frequent episodes without candidate generation but by expanding prefix in the sequence. They are fit to use minimal occurrence as frequency definition. However, most descriptions of such algorithms in the literature lack the details of how to detect a minimal occurrence of an expanded episode. To our best understanding, these algorithms usually consider the occurrences of a prefix as independent to each other while expanding to longer episodes. As a consequence, it requires a post-processing step for ensuring a detected occurrence is truly a minimal occurrence. Recently, Achar et al. [1] solved this problem by introducing a detailed implementation to compute a minimal occurrence list of an expanded episode.

Patnaik et al. [35] considered episode mining on dynamic event streams. However, their work is fundamentally different from ours. In their work, event sequence grows with a batch of data, including events happening on a set of consecutive time stamps. For a new batch of data, any batch-mode episode mining algorithm can be used to detect the candidate episodes. The main contribution of their work is to identify a frequency lower-bound such that only the episodes with frequency higher than this bound are likely the top-k frequent ones with a high probability in a time-window. Since the episode mining process is only applied on the batch of growing events locally, episodes spanning over two consecutive batches cannot be identified. In an extreme case, when each batch of data contains only the events from a single time stamp, their method fails to produce correct results. However, our work actually considers this extreme case where data arrive tick by tick. In this sense we call our problem *online frequent episode mining*.

New problems extended from frequent episode mining have also been studied in [41] and [40] recently. Wu et al. [41] combine frequent episode mining and utility pattern

mining to discover high utility episodes in complex event sequence. In [40], the authors focus on mining probabilistic frequent serial episodes over a sequence of uncertain events. In these problems, minimum support is no longer taken as the interesting measure. Instead, they consider other domain-specific metrics.

Online mining of different kinds of patterns, such as mining frequent pattern on stream data, has been extensively researched. The proposed algorithms in such studies can be grouped into two classes, namely approximate methods and exact methods. Approximate algorithms, such as Carma [12], LCA [28], estDec [7], FP-Stream [10] and FDPM [42], discover frequent patterns by approximate support counting. Exact algorithms, including DSTree [24], SWIM [32], CanTree [25], CPS-tree [37] and MOMENT [8], usually utilize a prefix-tree structure to store itemsets. They mainly focus on maintaining prefix-trees upon incoming new data.

The main difference between online itemset mining and online episode mining is that, in online itemset mining, we do not need to use the time information of each transaction. However, in online episode mining, every event has an occurring time. It is hard to use only one node to represent the same event occurring at different times. Thus, in this paper, we have to use a group of episode tries to store all episode occurrences. This requirement makes the updating the structure much harder. Thus, as mentioned earlier, we propose the concept of *last episode occurrence* to reduce the execution time of trie updating.

III. Frequent Episode Mining

This section provides an overview of the definitions and properties used in this paper. For details about the framework of episode mining, readers could refer to [30], [2], [41]. We also formulate the problem of online frequent episode mining in this section.

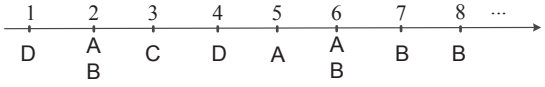


Fig. 1. The running example of event sequence.

Definition 1 (**Event sequence**). Let $\mathcal{E}=\{e_1, e_2, \cdots, e_m\}$ be a finite set of events. An *event sequence*, denoted $\vec{S}=\langle(E_1, t_1), (E_2, t_2), \cdots, (E_n, t_n)\rangle$, is an ordered sequence of events, where each $E_i \subseteq \mathcal{E}$ consists of all events associated with time stamp t_i, and $t_j < t_k$ for any $1 \leq j < k \leq n$. For example, Figure 1 shows an event sequence $\vec{S}=\langle$ ($\{$D$\}$, 1), ($\{$A, B$\}$, 2), ($\{$C$\}$, 3), ($\{$D$\}$, 4), ($\{$A$\}$, 5), ($\{$A, B$\}$, 6), ($\{$B$\}$, 7), ($\{$B$\}$, 8)$\rangle$.

Definition 2 (**Episode**). An episode (also known as a serial episode) α is defined as a non-empty totally ordered set of events of the form $e_1 \rightarrow \cdots e_j \rightarrow \cdots e_k$ where $e_i \in \mathcal{E}$ for all $i \in [1, k]$ and the event e_i occurs before the event e_j for all $1 \leq i < j \leq k$. The *length* of an episode is defined as the number of events in the episode. An episode α of length k is called a k-episode. For example, α = D$\rightarrow$A$\rightarrow$C is a 3-episode. An event A can also be viewed as a 1-episode.

Definition 3 (**Sub-episode and super-episode**). Consider two episodes $\alpha = e_1 \rightarrow \cdots e_i \rightarrow \cdots \rightarrow e_n$ and $\beta = e'_1 \rightarrow \cdots e'_j \rightarrow \cdots \rightarrow e'_k$ where $k \leq n$. The episode β is a *sub-episode* of α (correspondingly α is a *super-episode* of β), denoted as $\beta \preceq \alpha$, if and only if there exists k integers $1 \leq i_1 < i_2 < \cdots < i_k \leq n$ such that $e_{i_j} = e'_j$ for every $j \in [1, k]$. For example, 2-episode β = D$\rightarrow$C is a sub-episode of 3-episode D$\rightarrow$A$\rightarrow$C, but β' = A$\rightarrow$D is not.

Definition 4 (**Occurrence**). Given an episode $\alpha = e_1 \rightarrow \cdots e_i \rightarrow \cdots \rightarrow e_k$, $[t_1, \cdots, t_i, \cdots, t_k]$ is an occurrence of α if and only if (1) e_i occurs at t_i for all $i \in [1, k]$; (2) $t_1 < t_2 < \cdots < t_k$; and (3) t_k - $t_1 < \delta$ where δ is a user-specified threshold called the *maximum occurrence window*. t_1 is the *start time* and t_k is the *end time* of the occurrence. For example, in the running example sequence shown in Figure 1, $[1, 2, 3]$ constitutes an occurrence of episode D$\rightarrow$A$\rightarrow$C if δ is set to 3 while $[2, 3, 4]$ does not.

Definition 5 (**Equivalence of occurrences**). Consider an episode $\alpha = e_1 \rightarrow \cdots e_i \rightarrow \cdots \rightarrow e_k$ and its two different occurrences $[t_1, \cdots, t_i, \cdots, t_k]$ and $[t'_1, \cdots, t'_i, \cdots, t'_k]$. The two occurrences are considered *equivalent* if and only if $t_1 = t'_1$ and $t_k = t'_k$. In other words, they are considered the same occurrence and we use only the start time and the end time to represent an occurrence. Based on this concept, we denote an occurrence of episode α as (α, $[t_1, t_k]$) hereafter. $[t_1, t_k]$ is called an *occurrence window* of α. For instance, consider episode D$\rightarrow$A$\rightarrow$B in the running example, its two occurrences $[4, 5, 7]$ and $[4, 6, 7]$ are equivalent. They are both instances of occurrence (D$\rightarrow$A$\rightarrow$B, $[4, 7]$).

Definition 6 (**Minimal occurrence**). Consider two time windows $[t_1, t_2]$ and $[t'_1, t'_2]$. $[t'_1, t'_2]$ is *subsumed* by $[t_1, t_2]$ if $t_1 \leq t'_1$ and $t'_2 \leq t_2$. An occurrence window of episode α, $[t_1, t_2]$, is a *minimal occurrence window* of α if no other occurrence window $[t'_1, t'_2]$ of α is subsumed by $[t_1, t_2]$. The occurrence of α in an minimal occurrence window $[t_1, t_2]$ of α is defined as a minimal occurrence of α, and we denote it by (α, $[t_1, t_2]$). The set of all distinct minimal occurrences of α is denoted $\mathrm{moSet}(\alpha)$. For example, in Figure 1, moSet(A$\rightarrow$B) = $\{[5, 6], [6, 7]\}$ if $\delta = 3$.

Definition 7 (**Support of an episode**). The *support* of an episode α, denoted as $\mathrm{sp}(\alpha)$, is defined as the number of distinct minimal occurrences, i.e., $\mathrm{sp}(\alpha) = |\mathrm{moSet}(\alpha)|$. Thus, the support of A$\rightarrow$B is 2 in the running example.

Definition 8 (**Frequent episode**). An episode is called *frequent*, if and only if its support is no less than min_sup—a user-specified *minimum support threshold*. Otherwise, the episode is infrequent.

Minimal occurrence as a frequency measure does not satisfy anti-monotonicity. It means the supports of sub-episodes may be less than that of their super-episodes if minimal occurrence is used to define episode frequency. For instance, if δ is set to 3, the support of A$\rightarrow$B$\rightarrow$C in Figure 2 is 2 (moSet(A$\rightarrow$B$\rightarrow$C) = $\{[1, 3], [2, 4]\}$), while the support of its sub-episode A$\rightarrow$C is only 1 (moSet(A$\rightarrow$C) = $\{[2, 3]\}$). Neither $[1, 3]$ nor $[2, 4]$ is a minimal occurrence window of A$\rightarrow$C because $[2, 3]$ is subsumed by the two time windows.

Problem statement of frequent episode mining (batch mode): Given an event sequence $\vec{S}$, a minimum support threshold min_sup and a maximum occurrence window threshold δ,

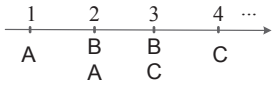


Fig. 2. A toy event sequence.

the frequent episode mining problem in batch mode is to find all frequent episodes in $\vec{S}$.

Problem statement of online frequent episode mining: Consider a dynamic, ever-growing event sequence $\vec{S}$ with the *current* time stamp t_k. We define the *valid sequence* as $\vec{S}^k_\Delta = \langle (E_i, t_i), \cdots, (E_k, t_k) \rangle$, where $i = \max(1, k - \Delta + 1)$ and $\Delta \in \mathcal{N}^+$ is the time window size within which the users would like to find the frequent episodes. $\vec{S}^k_\Delta$ thus contains the events in the latest Δ time stamps. The problem of online frequent episode mining is to find all the frequent episodes within the current valid sequence $\vec{S}^k_\Delta$.

Here, the last time stamp t_k may grow along the time. When the current time goes to t_{k+1}, a brute-force and thus clearly inefficient solution is to find the frequent episodes within $\vec{S}^{k+1}_\Delta$ from the scratch. This paper proposes an online algorithm to solve the problem based on immediate results from the last time stamp.

In summary, we have three user-specified parameters:

- δ, the maximum occurrence window threshold. An episode occurrence must be within the window size of δ.
- min_sup, the minimum support threshold. An episode is frequent only if its support is not smaller than min_sup.
- Δ, the window size for the current valid sequence. Only the events from the last Δ time stamps are considered for frequent episode mining.

Usually, we have $\Delta \gg \delta$, indicating that we consider the sequence in a relatively long time period and each occurrence of an episode must be within a much shorter time window δ.

For convenience of exposition, in the ensuing discussion we first consider the task when $\Delta = +\infty$, which means that we consider the whole event sequence so far. Then, we consider the general case when Δ is set to any positive integer. When discussing this case and its examples, we always set $\delta = 4$, $min\_sup = 2$ and $\Delta = 7$ for the event sequence in Figure 1.

IV. Online Frequent Episode Mining Framework

Based on the definitions and problem statements in Section III, we introduce the framework of online frequent episode mining in this section. Prior to that, we first introduce an important concept used in this framework.

Definition 9 (Minimal episode occurrences starting at t_i and ending no later than t_j). Given a time window $[t_i, t_j]$, we use M_i^j to denote the set of all minimal episode occurrences for which the start time is *equal* to t_i and the end time is *not larger than* t_j.

For example, in Figure 1, $M_5^7 = \{(\mathsf{A}, [5,5]), (\mathsf{A} \rightarrow \mathsf{A}, [5,6]), (\mathsf{A} \rightarrow \mathsf{B}, [5,6]), (\mathsf{A} \rightarrow \mathsf{B} \rightarrow \mathsf{B}, [5,7]), (\mathsf{A} \rightarrow \mathsf{A} \rightarrow \mathsf{B}, [5,7])\}$.

Theorem 1: Given a sequence $\vec{S}$ with the time stamps starting from 1 to k, $\mathcal{M} = \{M_1^\delta \cup M_2^{\delta+1} \cup \cdots \cup M_{k-\delta}^{k-1} \cup M_{k+1-\delta}^k \cup M_{k+2-\delta}^k \cup \cdots M_{k-1}^k \cup M_k^k\}$ contains all the minimal episode occurrences in this sequence.

Proof: Since M_i^j contains all minimal occurrences of episodes starting at time stamp t_i for each $i \in [1, k]$, the set $\mathcal{M}$ does not omit any minimal occurrence in $\vec{S}$. Since the time interval between the start time and the end time of each M_i^j is always smaller than δ, i.e., $t_j - t_i < \delta$, for all $1 \leq i \leq k$ and $\delta \leq j \leq k$, every minimal occurrence in $\mathcal{M}$ always satisfies the constrains in our formulation (See Definition 4). Hence, $\mathcal{M}$ always contains all valid minimal occurrences of all episodes in $\vec{S}$. ■

Based on the set of all minimal episode occurrences in $\mathcal{M}$, we can count the support of each episode and then identify all the frequent episodes whose frequencies exceed min_sup.

Theorem 1 provides a natural way to divide the elements in $\mathcal{M}$ into k separate subsets, namely $M_1^\delta, M_2^{\delta+1}, \cdots, M_{k-\delta}^{k-1}, M_{k-\delta+1}^k, M_{k-\delta+2}^k, \cdots, M_{k-1}^k, M_k^k$. These subsets can be further divided into the following two groups:

$$\mathcal{M}_{ex} = \{M_1^\delta, M_2^{\delta+1}, \cdots, M_{k-\delta}^{k-1}, M_{k-\delta+1}^k\} \quad (1)$$

$$\mathcal{M}_{in} = \{M_{k-\delta+2}^k, M_{k-\delta+3}^k, \cdots, M_{k-1}^k, M_k^k\} \quad (2)$$

Clearly, $\mathcal{M}_{ex}$ contains the first $k - \delta + 1$ components. For any $M_i^j \in \mathcal{M}_{ex}$, we have $j - i = \delta - 1$. It means that M_i^j includes all possible minimal episode occurrences starting from time stamp t_i and having minimal occurrence window sizes at most δ. On the other hand, $\mathcal{M}_{in}$ contains the last $\delta - 1$ components. For any $M_i^j \in \mathcal{M}_{in}$, $j - i < \delta - 1$.

Consider that the event set E_{k+1} at the time stamp t_{k+1} comes to the sequence of $\vec{S}$. Since, for every component $M_i^j \in \mathcal{M}_{ex}$, $j - i = \delta - 1$, the combination of any occurrence in M_i^j and any event in E_{k+1} results in an occurrence whose occurrence window size is bigger than δ. Thus, the new events in E_{k+1} do not affect the components in $\mathcal{M}_{ex}$. However, for any component $M_i^j \in \mathcal{M}_{in}$, since $j - i < \delta - 1$, any such combination leads to an occurrence whose occurrence window size is at most δ. Thus, the new events in E_{k+1} do affect the components in $\mathcal{M}_{in}$. Then, it comes to the crux of the proposed method, namely how to update $\mathcal{M}_{ex}$ and $\mathcal{M}_{in}$ with the new events E_{k+1} appended.

Next, we will first address the question of how to store the components in $\mathcal{M}_{ex}$ and $\mathcal{M}_{in}$, and then discuss the updating process when new events E_{k+1} are appended to the end of $\vec{S}$.

A. The Storage Framework

For the traditional batch mode problem of frequent episode mining, all solutions remove infrequent events by scanning the whole sequence in the first round, and then the mining process performs in the space of frequent events. This pre-processing step greatly reduces the event space and thus saves much memory consumption. However, in the online mode of frequent episode mining we cannot apply such pre-processing since an infrequent event in the current sequence may become frequent in the future. Thus, the mining process can only be conducted in the original event space, which may lead to a sharp increase of memory consumption for storing all minimal

episode occurrences. To tackle this challenge, we propose a storage management framework, as shown in Figure 3(a).

In Figure 3(a), all components in $\mathcal{M}_{in}$ and $\mathcal{M}_{ex}$ are stored by chronological orders. Since the $\delta - 1$ components in $\mathcal{M}_{in}$ will be updated with the new coming events, $\mathcal{M}_{in}$ can always stay in the main memory. (Note that δ usually is a small number.) For the other part $\mathcal{M}_{ex}$, since the new coming events do not affect any components in $\mathcal{M}_{ex}$, we can save the $k-\delta+1$ components into the external storage especially when k is very large. There are still two simple structures stored in this framework, namely *frequent episode set* and *infrequent episode set*. Both of them are tables of which the key records the name of episodes and the value records the support count of the corresponding episode. Same with $\mathcal{M}_{ex}$, the two structures can also be stored in external storage when the number of discovered episodes is very large.

B. The Solution Framework

Based on the storage framework, Figure 3(b) shows the solution framework of the proposed method. Specifically, with E_{k+1} coming this updating process can be detailed into the following two steps:

1) We first add a new component M_{k+1}^{k+1} into $\mathcal{M}_{in}$ of Equation (2). Namely, we have

$$\mathcal{M}_{in} \leftarrow \mathcal{M}_{in} \cup \{M_{k+1}^{k+1}\} \tag{3}$$

Then, for each component M_i^k in $\mathcal{M}_{in}$ of Equation (3), update its upper index from k to $(k+1)$. Finally, we have

$$\mathcal{M}_{in}^{'} = \{M_{k-\delta+2}^{k+1}, M_{k-\delta+3}^{k+1}, \cdots, M_k^{k+1}, M_{k+1}^{k+1}\} \tag{4}$$

2) For the first component in $\mathcal{M}_{in}^{'}$, namely $M_{k-\delta+2}^{k+1}$, it reaches the maximal time range size of $(\delta - 1)$. Thus, we remove it from $\mathcal{M}_{in}^{'}$ and then put it into $\mathcal{M}_{ex}$ of Equation (1). Namely, we have

$$\mathcal{M}_{in} \leftarrow \mathcal{M}_{in}^{'} - \{M_{k-\delta+2}^{k+1}\} \tag{5}$$

$$\mathcal{M}_{ex} \leftarrow \mathcal{M}_{ex} \cup \{M_{k-\delta+2}^{k+1}\} \tag{6}$$

In short, this updating process can be summarized as follows,

$$\mathcal{M}_{ex} \leftarrow \mathcal{M}_{ex} \cup \{M_{k-\delta+2}^{k+1}\} \tag{7}$$

$$\mathcal{M}_{in} \leftarrow \{M_{k-\delta+3}^{k+1}, \cdots, M_{k-1}^{k+1}, M_k^{k+1}, M_{k+1}^{k+1}\} \tag{8}$$

After these two steps, we update the current set of frequent episodes. After E_{k+1} comes, the solution finds all the new minimal episode occurrences, denoted by

$$\begin{aligned}\mathcal{Q} \leftarrow & (M_{k-\delta+2}^{k+1} - M_{k-\delta+2}^{k}) \cup (M_{k-\delta+3}^{k+1} - M_{k-\delta+3}^{k}) \\ & \cup \cdots \cup (M_k^{k+1} - M_k^k) \cup M_{k+1}^{k+1}\end{aligned} \tag{9}$$

Thus, the count of each episode in $\mathcal{Q}$ will be added 1. Thus, when $\Delta = +\infty$ (meaning that the valid event sequence covers the full size of the sequence until now), some infrequent episodes will change to be frequent with the coming of E_{k+1}.

C. The Solution When $\Delta \neq +\infty$

When $\Delta \neq +\infty$, the episodes with the starting time outside the valid event sequence will expire. Thus, the count of each expired episode should minus 1. When E_{k+1} comes, the valid event sequence changes to $\vec{S}_{\Delta}^{k+1}$, meaning that all the events in E_i $(i = \max(1, k - \Delta + 1))$ expire. Therefore, the count of each episode in $\mathcal{M}_i^{i+\delta-1}$ should minus 1. Some previous frequent episodes may become infrequent after this step.

V. MESELO ALGORITHM

Next, we detail the method of identifying all the new minimal episode occurrences, denoted by $\mathcal{Q}$ in Equation (9), with the coming of E_{k+1}. In this section, we will give its efficient solution.

A. Episode Trie

First, we give the description of the data structure, which stores all minimal occurrences in M_i^j. Remind that M_i^j is the set of all minimal episode occurrences, starting at time t_i and ending no later than t_j. In our solution, we use the *trie* structure to store all the elements in M_i^j in a compact way.

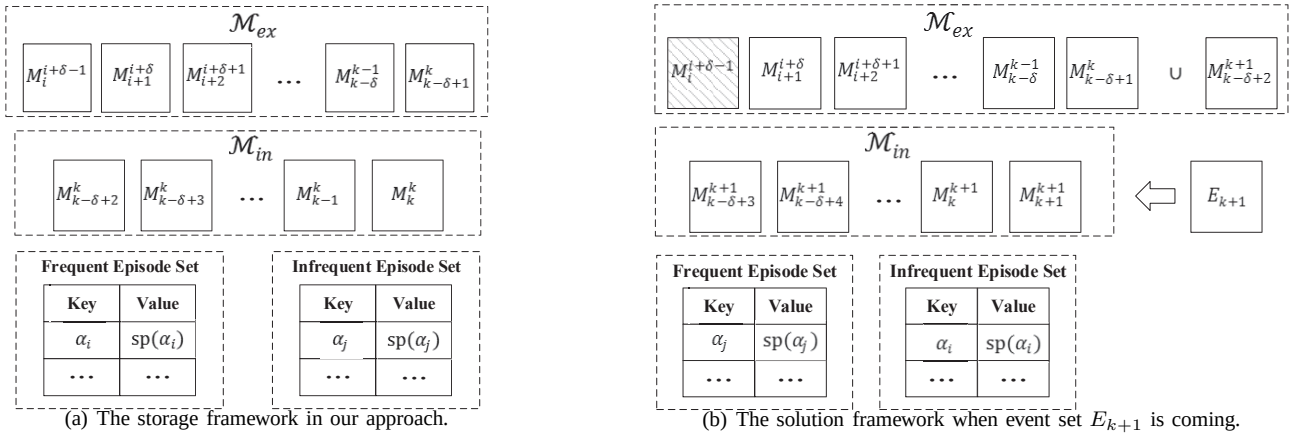
A trie [15], also called prefix tree, is an ordered tree data structure that is used to store a dynamic set where the keys are usually strings. In a trie, all the descendants of a node have a common prefix of the string associated with that node, and the root is associated with the empty string. In this study we utilize the trie structure to record all the minimal episode occurrences in M_i^j associated with the starting time t_i and the end time t_j. We call this data structure *episode trie*.

Definition 10 (**Episode Trie**). Given a time-window $[t_i, t_j]$, an *episode trie* (E-trie for short, denoted $\mathcal{T}_i^j$) is a trie-like structure defined below.

1) Each node p, denoted by $p.event$:$p.time$, consists of two fields: $p.event$ and $p.time$. Here, $p.event$ registers which event this node represents, and $p.time$ registers the occurrence time stamp of such event.
2) The *event* field of the root is associated with the empty string (labeled as "root"), and the *time* field of the root is equal to t_i, the starting time of all the episode occurrences in this set.

With this data structure, each node p (except the root) in an episode trie actually represents an episode occurrence. The event sequence along the path from the root to p corresponds to its content, and its occurrence window is $[t_i, p.time]$. For clarity and convenience, we use $\mathrm{ep}(p)$ to represent the episode associated with a node p in an episode trie, and then the episode occurrence can be denoted as $(\mathrm{ep}(p), [t_i, p.time])$.

For example, Figure 4 shows the episode trie $\mathcal{T}_5^7$ to store all the elements in M_5^7 for the running example in Figure 1. Here, $M_5^7 = \{(\mathsf{A}, [5,5]), (\mathsf{A} \rightarrow \mathsf{A}, [5,6]), (\mathsf{A} \rightarrow \mathsf{B}, [5,6]), (\mathsf{A} \rightarrow \mathsf{B} \rightarrow \mathsf{B}, [5,7]), (\mathsf{A} \rightarrow \mathsf{A} \rightarrow \mathsf{B}, [5,7])\}$. It contains 5 minimal episode occurrences. The starting time of all these occurrences is equal to 5, and their end time is not later than 7. More importantly, each element in M_5^7 corresponds to a node (except the root) in $\mathcal{T}_5^7$. For example, the occurrence of $(\mathsf{A} \rightarrow \mathsf{A} \rightarrow \mathsf{B}, [5,7])\}$ corresponds to the leftmost leaf node in the trie of Figure 4. $\mathsf{A} \rightarrow \mathsf{A} \rightarrow \mathsf{B}$ is actually the event sequence along the root to



(a) The storage framework in our approach. (b) The solution framework when event set E_{k+1} is coming.

Fig. 3. The whole framework in our approach, where $i = \max(1, k - \Delta + 1)$.

this leftmost leaf node. In this way we call $\mathcal{T}_i^j$ is equivalent to M_i^j, denoted as $\mathcal{T}_i^j \Leftrightarrow M_i^j$.

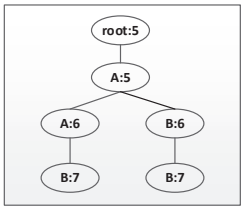


Fig. 4. The episode trie $\mathcal{T}_5^7$.

B. The Last Episode Occurrence

Next, we introduce another important definition, *the last episode occurrence*, which is the key concept to the proposed online algorithm.

Definition 11 (**The last episode occurrence**). Given a time window $[t_i, t_j]$, an episode occurrence $(\alpha, [t_1, t_2])$ ($t_i \le t_1 \le t_2 \le t_j$) is the *last episode occurrence* of α within this time window if and only if there does not exist another occurrence of $(\alpha, [t_1^{'}, t_2^{'}])$ ($t_i \le t_1^{'} \le t_2^{'} \le t_j$) such that $t_1^{'} > t_1$. The set of all the last episode occurrences within the time window of $[t_i, t_j]$ is denoted by L_i^j.

See the running example in Figure 1. Consider the time window of $[4, 7]$. (A→B, $[6, 7]$) is the last occurrence of episode A→B within this time window. However, (A→B, $[5, 6]$) is not the last occurrence because of the existence of (A→B, $[6, 7]$).

With the definition of the last episode occurrence, the set of M_j^k can be divided into two parts. Namely, considering the time window of $[k - \delta + 1, k]$, the two disjoint parts of M_j^k can be represented as

$$M_j^k = \begin{cases} S_j^k = M_j^k \cap L_{k-\delta+1}^k \\ \overline{S_j^k} = M_j^k - (M_j^k \cap L_{k-\delta+1}^k) \end{cases} \tag{10}$$

Clearly, all the elements in S_j^k are the *last minimal* occurrences within the time window of $[k-\delta+1, k]$. However, the elements in $\overline{S_j^k}$ are just the *minimal* occurrences, but not the *last* ones.

Check the running example in Figure 1 again. When $t_k = 7$, with time window $[4, 7]$, M_5^7 can be divided into two parts. Namely, $S_5^7 = \{(\mathsf{A} \to \mathsf{A}, [5, 6]), (\mathsf{A} \to \mathsf{A} \to \mathsf{B}, [5, 7]), (\mathsf{A} \to \mathsf{B} \to \mathsf{B}, [5, 7])\}$, and $\overline{S_5^7} = \{(\mathsf{A}, [5, 5]), (\mathsf{A} \to \mathsf{B}, [5, 6])\}$.

Remind that M_i^j is equivalent to its corresponding episode trie $\mathcal{T}_i^j$. Thus, the nodes in $\mathcal{T}_i^j$ can then be divided into two groups. If the corresponding episode of a node p in $\mathcal{T}_i^j$ belongs to $\overline{S_i^j}$ we call that p is a *non-last-occurrence node* (*nlo-node* for short); Otherwise, p is a *last-occurrence node* (*lo-node* for short). As shown in Figure 5 (a) for $\mathcal{T}_5^7$, all the nlo-nodes are shaded while the lo-nodes remain blank.

It is clear that every lo-node in $\mathcal{T}_i^j$ corresponds to a last minimal episode occurrence. Instead, each nlo-node is only associated with a minimal occurrence, but not a last one. The lo-nodes and nlo-nodes have different properties in generating the minimal episode occurrences when new events coming. Thus, we clearly distinguish these types of nodes. In the following we will detail the proposed algorithm for online frequent episode mining.

C. MESELO Algorithm

In this subsection, we formally introduce the algorithm MESELO. According to the solution framework, there are all together two steps for updating process in the algorithm with E_{k+1} coming, 1) add a new component $\mathcal{T}_{k+1}^{k+1}$ to $\mathcal{M}_{in}$ and update the upper index of each component $\mathcal{T}_i^k$ in $\mathcal{M}_{in}$ from k to $k + 1$; 2) transfer the component that reach the maximal time range size of $\delta - 1$, i.e. $\mathcal{T}_{k-\delta+2}^{k+1}$ to $\mathcal{M}_{ex}$. Since, the second step is relatively simple, we merely focus on the first step. In order to explain our algorithm, we will consider the situation that the time stamp $t_k = 7$, and E_8 is following to arrive as an example in this subsection.

For the first step, add $\mathcal{T}_{k+1}^{k+1}$ can be completed by build an E-trie via E_{k+1}. Algorithm 1 shows its pseudo code. For each event $e \in E_{k+1}$, we insert a node associated with e as the child of the root node. For example, suppose the event set occurring on time stamp 8 is arriving as the latest event set, now the maximum occurrence window we consider is $[5, 8]$, and $\mathcal{T}_8^8$ is shown as Figure 5 (g).

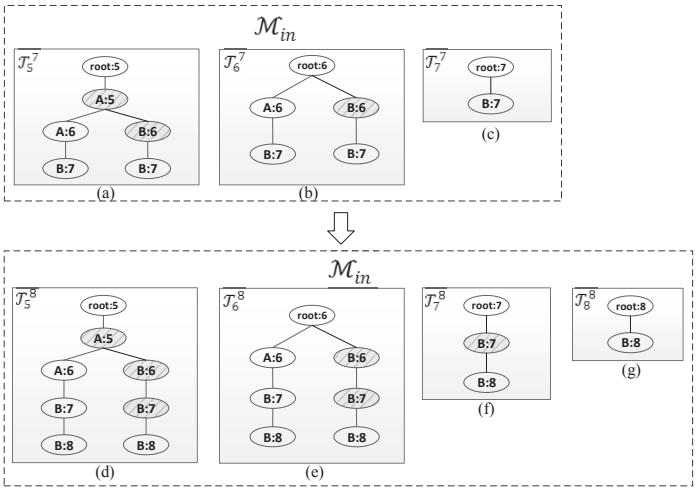


Fig. 5. The results of update process in $\mathcal{M}_{in}$ from time stamp = 7 to time stamp = 8. All the nlo-nodes are marked as shaded, and lo-nodes leave blank.

Algorithm 1: BuildTrie(E_{k+1}): Build Episode Trie via E_{k+1}

Input: E_{k+1}: the new event set
Output: $\mathcal{T}_{k+1}^{k+1}$: the new E-trie

1 create a root node $root$ = root:t_{k+1};
2 **foreach** *event* $e \in E_{k+1}$ **do**
3 create a child node $p = e{:}t_{k+1}$ of $root$;
4 return $\mathcal{T}_{k+1}^{k+1}$;

Then, MESELO begins to update all upper index of $\mathcal{T}_i^k$ from k to $k+1$. We perform such a process by a *reverse chronological order* that is we update from $\mathcal{T}_k^k$ to $\mathcal{T}_{k-\delta+2}^k$. The updating order is really important since we can use the concept of last minimal occurrence and minimal occurrence to improve the efficiency of algorithm. Briefly speaking, we only expand lo-nodes in each $\mathcal{T}_i^k$, where $k-\delta+2 \leq i \leq k$, with a simple additional constraint in this process. Meanwhile, the property of lo-node may change as new minimal episode occurrence is discovered.

Algorithm 2 shows the pseudo code of this step. First, we initialize the new discovered minimal episode occurrence set $\mathcal{Q}$ as an empty set. Since $\mathcal{T}_{k+1}^{k+1}$ contains all new discovered minimal occurrences of 1-episodes, we add each of them to $\mathcal{Q}$ (Line 2–3). Then, MESELO performs a sequential updating from $\mathcal{T}_k^k$ to $\mathcal{T}_{k-\delta+2}^k$. For updating every $\mathcal{T}_i^k$ ($k-\delta+2 \leq i \leq k$), the operations indeed can be separated into two parts. The first part is to expand nodes for adding minimal occurrences of episodes, and the second part is to update the type of nodes in the trie. For the first step, we have two essential constraints: 1) only lo-nodes can be expanded, and 2) when preparing to add a node associated with event e, only the nodes which do not contain child node associated with e can be expanded. Specifically, for any lo-node p in $\mathcal{T}_i^k$ ($k-\delta+2 \leq i \leq k$), we first get an episode α associated with p as a prefix ready to be expanded (Line 7). Then, for each event e belonging to E_{k+1}, we can always add a child node q of p which is associated with e if there is no child node of p associated with the same event e (Line 9 and 10). Note that it is an important constraint for both our frequent episode mining problem and the trie data structure. Then, a new expanded node can refer to a new episode with a new minimal occurrence. We also add such a minimal occurrence to the set $\mathcal{Q}$ ready for the later use (Line 11 and 12).

Take $\mathcal{T}_6^7$ shown in Figure 5 (b) as an example. Note that the only event in E_8 is B. As shown in the figure, since the node B:6 is a nlo-node, MESELO does not expand it. For the lo-node A:6, since there is already a child node which is associated with event B, the algorithm does not to expand it. For another two lo-nodes B:7 on the leaf, MESELO adds a child node B:8 for both of them as they satisfy all the constraints in this part. Finally, two new discovered minimal episode occurrences (A→B→B, $[6,8]$) and (B→B→B, $[6,8]$) are added into $\mathcal{Q}$.

Algorithm 2: UpdateTries: Mining New Minimal Episode Occurrences

Input: $\mathcal{M}_{in}$: the set of minimal occurrence still varies
E_{k+1}: the set of events occurring at time t_{k+1}
Output: $\mathcal{Q}$: the new discovered minimal episode occurrence set

1 $\mathcal{Q} \leftarrow \emptyset$;
2 **foreach** *event* $e \in E_{k+1}$ **do**
3 $\mathcal{Q} \leftarrow \mathcal{Q} \cup (e, [t_{k+1}, t_{k+1}])$;
4 $i \leftarrow k$;
5 **while** $i \geq k-\delta+2$ **do**
6 **foreach** *lo-node* $p \in \mathcal{T}_i^k$ **do**
7 $\alpha \leftarrow \mathrm{ep}(p)$;
8 **foreach** *event* $e \in E_{k+1}$ **do**
9 **if** *there is no child node of* p *associated with* e **then**
10 create a child node $q = e{:}t_{k+1}$ of p;
11 $\alpha' \leftarrow \mathrm{ep}(q)$;
12 $\mathcal{Q} \leftarrow \mathcal{Q} \cup (\alpha', [t_i, t_{k+1}])$;
13 **if** α *is contained by* $\mathcal{Q}$ **then**
14 set p to a nlo-node;
15 $i \leftarrow i-1$;
16 return $\mathcal{Q}$;

After adding nodes to an episode trie, there is another important step left which is to check whether the node p should be change into a nlo-node. The transformation is triggered when there is a same $\alpha = \mathrm{ep}(p)$ in $\mathcal{Q}$, which means there is some other minimal occurrence of α is behind this one (Line 13–14). Continue taking the update of $\mathcal{T}_6^7$ as an example. Before updating $\mathcal{T}_6^7$, the $\mathcal{Q} = \{$(B, $[8,8]$), (B→B, $[7,8]$)$\}$. When the expanding of the node B:7 on the rightmost leaf of $\mathcal{T}_6^7$ is finished, we check the prefix episode α = B→B. Since $\mathcal{Q}$ contains an element associated with episode B→B, MESELO change such node into a nlo-node, and it will not be expanded in the future. The result of $\mathcal{T}_6^8$ is shown as Figure 5 (e). The whole results of update process of this step, i.e. update $\mathcal{T}_5^7$, $\mathcal{T}_6^7$ and $\mathcal{T}_7^7$ to $\mathcal{T}_5^8$, $\mathcal{T}_6^8$ and $\mathcal{T}_7^8$, are shown as Figure 5 (a)–(f), respectively. After updating $\mathcal{M}_{in}$, the next step is to move $\mathcal{T}_{k-\delta+2}^{k+1}$ into $\mathcal{M}_{ex}$ as its time range reaches $\delta-1$. We leave its pseudo code for the space limitation.

After updating episode tries, MESELO performs the final step which is to find frequent episodes in $\vec{S}_{\Delta}^{k+1}$. As previously mentioned, we need to load the expire episode trie $\mathcal{T}_{k-\Delta+1}^{k-\Delta+\delta}$

from $\mathcal{M}_{ex}$ and minus support count of all episodes in such E-trie by 1 and add support count of all episodes in $\mathcal{Q}$ by 1. Then, episodes whose support count is no less than min_sup are output. For example, under the parameter settings in the running example, the set of frequent episodes in $\vec{S}_7^8$ is {A, B, A→B, A→B}. D, D→A and D→B, which are frequent in $\vec{S}_7^7$, become infrequent in $\vec{S}_7^8$ since $E_1 = \{D\}$ expires when E_8 arrives. Algorithm 3 gives the pseudo code of this step.

With all aforementioned algorithms, we can give the pseudo code of the whole solution framework in Algorithm 4. All algorithms are called as sub-procedures in our approach.

Algorithm 3: OutputF: Output Frequent Episodes

Input: $\mathcal{C}$: the current set of infrequent episodes
$\mathcal{F}$: the current set of frequent episodes
$\mathcal{Q}$: the new discovered minimal episode occurrence set
t_{k+1}: current time stamp
Δ: the window size of valid sequence
min_sup: minimal support threshold
Output: $\mathcal{F}$: the current set of frequent episodes

load $\mathcal{T}_{k-\Delta+1}^{k-\Delta+\delta}$ from $\mathcal{M}_{ex}$;
foreach *node* $p \in \mathcal{T}_{k-\Delta+1}^{k-\Delta+\delta}$ **do**
  $\alpha \leftarrow \mathrm{ep}(p)$;
  $\mathrm{sp}(\alpha) \leftarrow \mathrm{sp}(\alpha) - 1$;
foreach $(\alpha, [t_i, t_j]) \in \mathcal{Q}$ **do**
  $\mathrm{sp}(\alpha) \leftarrow \mathrm{sp}(\alpha) + 1$;
update $\mathcal{F}$ and $\mathcal{C}$ by min_sup;
return $\mathcal{F}$;

Algorithm 4: MESELO Algorithm

Input: E_{k+1}: the new event set
δ: maximum occurrence window
min_sup: minimum support
Δ: window size
Output: $\mathcal{F}$: frequent episodes

$\mathcal{C} \leftarrow$ current set of infrequent episodes; $\mathcal{F} \leftarrow$ current set of frequent episodes;
$\mathcal{T}_{k+1}^{k+1} \leftarrow \mathrm{BuildTrie}(E_{k+1})$;
$\mathcal{Q} \leftarrow \mathrm{UpdateTries}(\mathcal{M}_{in}, E_{k+1})$;
update $\mathcal{M}_{in}$ and $\mathcal{M}_{ex}$ by Eq. (5) and (6);
$\mathcal{F} \leftarrow \mathrm{OutputF}(\mathcal{C}, \mathcal{F}, \mathcal{Q}, t_{k+1}, \Delta, min\_sup)$;

D. Correctness and Completeness of MESELO

In this subsection, we prove the correctness and the completeness of our approach. Not that the essential task of our problem is to find minimal episode occurrences in event sequence, and our algorithm can detect them efficiently. By our algorithm, an episode α associated with a lo-node p in an E-trie in $\mathcal{M}_{in}$ can directly be expanded with an event e if there is no child node of p which is associated with e, then we can get a new episode α'. We first prove such a kind of expansion can derive a minimal occurrence of α'.

Theorem 2: (Correctness). Given an E-trie $\mathcal{T}_i^k$, where $k - \delta + 2 \leq i \leq k$ and an event set E_{k+1}, for a lo-node p of $\mathcal{T}_i^k$ with $\mathrm{ep}(p) = \alpha$ and an event $e \in E_{k+1}$, if there is no child node of p associated with e, then we can get a new episode $\alpha' = \alpha \rightarrow e$ with a new minimal occurrence $(\alpha', [t_i, t_{k+1}])$.

Proof: If $(\alpha', [t_i, t_{k+1}])$ is not a minimal occurrence, we should find a minimal occurrence of $(\alpha', [t_j, t_l])$ such that $[t_j, t_l]$ is subsumed by $[t_i, t_{k+1}]$. Since p is a lo-node of $\mathcal{T}_i^k$, it is impossible to exist an occurrence $(\alpha', [t_j, t_l])$ such that $t_j > t_i$, otherwise p should be a nlo-node. On the other hand, it is also impossible to exist an occurrence $(\alpha', [t_j, t_l])$ such that $t_l < t_{k+1}$ since there is no child node of p associated with e. Hence, we have $t_i = t_j$ and $t_l = t_{k+1}$. $(\alpha', [t_i, t_{k+1}])$ is the minimal occurrence of α' in the time window $[t_{k-\delta+2}, t_{k+1}]$. ■

Theorem 3: (Completeness). An E-trie $\mathcal{T}_i^k$, where $k-\delta+2 \leq i \leq k$, stores all minimal episodes occurrences whose start time is equal to t_i, and the end time is not bigger than t_k. After the updating process of MESELO algorithm when a new event set E_{k+1} is coming, the updated E-trie $\mathcal{T}_i^{k+1}$ stores all minimal episode occurrences whose start time is t_i and the end time is not bigger than t_{k+1}.

Proof: To prove this theorem, we in fact need to prove all the new inserted nodes in $\mathcal{T}_i^{k+1}$ are associated with minimal episode occurrences whose start time is t_i and the end time is t_{k+1}. Suppose there is an episode minimal occurrence $(\alpha \rightarrow e, [t_i, t_{k+1}])$ and the node associated with event e is not in $\mathcal{T}_i^{k+1}$ where $e \in E_{k+1}$. Then, such a node associated with e must exists in another $\mathcal{T}_j^{k+1}$ where $j \neq i$ if we can acquire an occurrence of α'. However, it is impossible since the occurrence window of α' can only be $[t_j, t_{k+1}]$ and $j \neq i$. Hence, all the new discovered episode minimal occurrence whose the start time is t_i and the end time is t_{k+1} must be stored in $\mathcal{T}_i^{k+1}$ instead of other tries. ■

As MESELO sequentially scans over the event sequence, it finds all and only minimal occurrences of episodes within every possible time window. Hence, the algorithm is soundness and completeness.

In MESELO algorithm, we have marked all last episode occurrences in $[t_{k-\delta+1}, t_k]$ in $\mathcal{T}_{k-\delta+1}^k$, $\mathcal{T}_{k-\delta+2}^k$, $\cdots$, $\mathcal{T}_{k-1}^k$, $\mathcal{T}_k^k$ at time stamp t_k. Then, when new events on time stamp t_{k+1} is coming and the tries in $\mathcal{M}_{in}$ are updated to $\mathcal{T}_{k-\delta+2}^{k+1}$, $\mathcal{T}_{k-\delta+3}^{k+1}$, $\cdots$, $\mathcal{T}_k^{k+1}$, $\mathcal{T}_{k+1}^{k+1}$, there will be some lo-nodes be changed to nlo-nodes since the occurrences associated with these nodes are no longer the last ones in the current time window $[t_{k-\delta+2}, t_{k+1}]$. Hence, we have to prove the following theorem.

Theorem 4: Given a group of episode tries $\mathcal{T}_{k-\delta+2}^k$, $\mathcal{T}_{k-\delta+3}^k$, $\cdots$, $\mathcal{T}_{k-1}^k$ and $\mathcal{T}_k^k$ and a new discovered minimal episode occurrences set $\mathcal{Q}$, when updating the upper index of these tries to $k+1$, if a lo-node p of E-trie $\mathcal{T}_i^k$ (where $k - \delta + 2 \leq i \leq k$) is in $\mathcal{Q}$, which means $\mathrm{ep}(p) \in \mathcal{Q}$, then $(\mathrm{ep}(p), [t_i, p.time])$ is no longer a last episode occurrence, otherwise, it is still a last episode occurrence.

Proof: For convenience we assume $\mathrm{ep}(p) = \alpha$ in the proof. Since MESELO algorithm updates every $\mathcal{T}_i^k$ in a reverse chronological order, for a lo-node p in $\mathcal{T}_i^k$, if $\alpha \in \mathcal{Q}$, there must exist a new expanded lo-node q (which is also a leaf node) in an episode trie $\mathcal{T}_j^{k+1}$ where $j > i$ and $\mathrm{ep}(q) = \alpha$. According to Definition 11, $(\alpha, [t_i, p.time])$ is not a last occurrence of α in $[t_{k-\delta+2}, t_{k+1}]$ because there is another occurrence of $(\alpha, [t_j, q.time])$ such that $t_j > t_i$. Hence, the node p need to be changed into a nlo-node. Similarly, if $\alpha \notin \mathcal{Q}$, then we

cannot find another occurrence of α behind of $(\alpha, [t_i, p.time])$ in $[t_{k-\delta+2}, t_{k+1}]$. Hence, p is still a lo-node in this situation. ■

VI. Complexity Analysis

In this section, we analyze the time and space complexity of MESELO. In particular, we focus on analyzing the cost of computation and storage in updating $\mathcal{M}_{in}$ for every new event set and only consider the worst case.

For simplicity, we assume that the number of event types is m, i.e., $|\mathcal{E}| = m$, and each event set always contains all m events. According to node expansion strategy in MESELO, every node at most contains m child nodes. Hence, the worst case is that every non-leaf node in each E-trie always contains m child nodes. It leads to the bulkiest structure, and will cost the longest time for updating operations.

As we mentioned before, when an event set E_{k+1} is coming, the updating process for $\mathcal{M}_{in}$ includes constructing a new E-trie $\mathcal{T}_{k+1}^{k+1}$ and updating the $\delta - 1$ episode tries $\mathcal{T}_i^k$ to $\mathcal{T}_i^{k+1}$ where $k - \delta + 2 \leq i \leq k$. For constructing $\mathcal{T}_{k+1}^{k+1}$, MESELO will perform m insertions since we assume there are m events in E_{k+1}. Then, for updating each $\mathcal{T}_i^k$ to $\mathcal{T}_i^{k+1}$ ($k - \delta + 2 \leq i \leq k$), only leaf nodes in $\mathcal{T}_i^k$ can be further expanded as each non-leaf node has already contained m child nodes. Given an E-trie $\mathcal{T}_i^k$, the number of leaf node in $\mathcal{T}_i^k$ is m^{k-i+1}. Further, m child nodes can be inserted into each leaf node. Hence, in this step the total number of node insertion is $m(m^{\delta-1} + m^{\delta-2} + \cdots + m^2 + m)$. In sum, for every new coming event set, the overall time complexity for updating $\mathcal{M}_{in}$ will be $O(m^\delta + m^{\delta-1} + m^{\delta-2} + \cdots + m^3 + m^2 + m)$ $= O(\frac{m(m^\delta - 1)}{m-1})$. In short, the time complexity is $O(m^\delta)$.

After the update process in the worst case we have discussed, every E-trie in $\mathcal{M}_{in}$ has its bulkiest structure. Then, given an E-trie $\mathcal{T}_i^{k+1}$, the maximum number of nodes is $\sum_{j=i}^{k+1} m^{k-j+2}$, where $k - \delta + 2 \leq i \leq k + 1$. The total number of nodes of all E-tries in $\mathcal{M}_{in}$ will be $\sum_{i=k-\delta+2}^{k+1} \sum_{j=i}^{k+1} m^{k-j+2}$. Hence, the space complexity for storing all nodes in memory will be $O(m^\delta + 2m^{\delta-1} + 3m^{\delta-2} + \cdots + (\delta-2)m^3 + (\delta-1)m^2 + \delta m) = O(\frac{m^2(m^\delta-1)}{(m-1)^2} - \frac{m\delta}{m-1})$. In short, the space complexity is also $O(m^\delta)$.

In most applications, the size of event set m and the maximum occurrence window threshold δ are both small. Thus, the proposed method is practically useful in online episode mining.

VII. Experiments

A. Experiments Settings and Data Preparation

In this section, we evaluate the performance of the proposed algorithms on both online mode and batch mode. Experiments are performed on a server with a 2.00 GHz Intel Xeon E5-2620 Processor and 32G gigabytes memory, running on Windows Server 2008, and all of the algorithms are implemented in Java. For online mode, as we suppose $\mathcal{M}_{ex}$ is always in a dynamically growth and usually has masses of elements, we store every component in $\mathcal{M}_{ex}$ to a database. In these comparisons, a remote MySQL database server with a 2.00GHz Intel Xeon E5-2620 Processor and 16G gigabytes memory running on Linux is used, and the two machines are connected by 100Mb a local area network.

1) Experiments Settings: Here, we compare the proposed algorithm to the baseline methods for frequent episode mining in both online and batch mode.

Online mode. For online mode comparison, we design an intuitive brute force method, denoted as BRUTE, as a baseline. In BRUTE, every episode minimal occurrence is stored in an ordered record table $\mathcal{B}$. Once an event set E_{k+1} occurring at time stamp t_{k+1} is arriving, BRUTE always performs three steps. First, the minimal occurrences of episodes whose start time equals to $t_{k-\Delta+1}$ are loaded from $\mathcal{B}$. Since these elements expire in the current valid sequence window, we have to decrease the frequency of these episodes. Second, the minimal occurrences of episodes whose start time ranges from $[t_{k-\delta+2}, t_k]$ are loaded from $\mathcal{B}$, and BRUTE performs an exhaustive generation of new episode occurrences by expanding these minimal occurrences with events in E_{k+1}. Third, the algorithm checks whether every new episode occurrence is a minimal occurrence and only updates the frequency of the episodes with new minimal occurrences.

To further demonstrate the effectiveness of the proposed concept of the *last episode occurrence*, we deliberately design another baseline, denoted as MESELO-BS. MESELO-BS performs the similar steps as MESELO except that it expands episode occurrences on every node of any episode trie. In MESELO, there are two constraints to ensure every expansion deriving to a minimal episode occurrence: 1) only *lo-nodes* are considered to be expanded; 2) for a *lo-node* p, only if there is no child node of p associated with the input event e, e can be inserted as a child of p. However, these two rules are not considered in node expansion of MESELO-BS. Thus, we have to check the minimal occurrences of episodes by a post-processing step.

In both online or traditional frequent episode mining problem, each occurrence of same event has different meanings. While online frequent pattern algorithms do not use time information of transactions or items, they view every occurrence of a specific item in different transactions as the same. Hence, we can hardly conduct them to our problem directly or through a slight modifications. Due to the above reasons, we do not include these algorithms in such comparison.

Batch mode. Several state-of-the-art methods for offline frequent episode mining, namely MINEPI+ [18], PPS [27], UP-Span [41] and DFS [1], are compared in batch mode. All of these algorithms directly use minimal occurrence as frequency of episodes or compute minimal episode occurrences during the process. There are still some differences among them. PPS can only output the *longest distinct frequent episodes*. UP-Span is originally proposed for mining *high utility episodes*, however computing minimal occurrences of episodes is the core part of this algorithm. DFS computes minimal episode occurrences as an important operation though it is interested in other frequency measures. For UP-Span and DFS we slightly modified these two methods to our best knowledge in order to fit our problem. It is worth mentioning that these state-of-the-art methods have not compared with each other in previous literatures, and we are the first to compare them in terms of

TABLE I. STATISTICAL INFORMATION ON DIFFERENT DATA SETS

Data set	#Time stamp	#Events	Avg. #Events per Time stamp
Stock-1	2509	4	1.0
Stock-2		8	2.4
Stock-3		16	4.7
Stock-4		24	7.0
Stock-5		32	9.5
Stock-6		40	11.4
Retail	88,162	16,470	10.3
Kosarak	990,002	41,270	8.1
chainStore	1,112,949	46,086	7.3
BMS	59,601	497	2.5

the execution time in one paper.

In this study we do not compare the widely-cited method MINEPI [30] due to the following reasons. First, MINEPI adopts the apriori-like candidate generation, and thus some frequent episodes with longer length will be missed since the minimal occurrence does not hold the anti-monotone property. In other words, the output of MINEPI is different from that of the problem considered in this study. Second, some initial experiments show that MINEPI always performs the worst among all the methods.

Here, the proposed MESELO performs to process a dynamic event sequence. For fair comparison, in this batch mode we can perform a prior scan to find frequent events, then all compared methods perform the mining on the sequence containing only frequent events.

2) Data Preparation: Ten real data sets are used to evaluate the performance of the algorithms, and the data sets can be divided into two groups according to mining tasks, namely *online mode data sets* and *batch mode data sets*, respectively.

The *online mode data sets* include six sequences from China Stock Exchange Daily Trading list (denoted as Stock-1 to 6) over 2509 trading days from January 1st, 2004 to May 9th, 2014. Each sequence contains the events from the stocks of the corresponding industry (1–pharmaceuticals industry, 2–security industry, 3–electricity power industry, 4–iron and steel industry, 5–nonferrous-material industry and 6–estate industry). The events for a stock are generated based on its everyday closing price. Specifically, we calculate the increase ratio r of price between two consecutive trading days and then discretize the values of r into 4 levels: UH ($r >= 3.5\%$), UL ($0\% \leq r \leq 3.5\%$), DL ($-3.5\% \leq r < 0\%$), DH ($r \leq -3.5\%$). Thus, every day each stock can generate one of the four above events, and the events from the stocks belonging to a specific industry are put together to form the event sequence for the corresponding industry.

The *batch mode data sets* include four classic real data sets, namely Retail [16], ChainStore [13], Kosarak [16] and BMS [17]. Among them, Retail and chainStore are market basket data from stores, while Kosarak and BMS are all click-stream data from some web sites. The processing method on these data sets are detailed in [41]. Table I shows the statistic information of all data sets used in the experiments.

B. Experiment Results

1) Online Mode Comparison: In this comparison, we sequentially read every event set of the coming time stamp, and perform online frequent episode mining. We record the execution time at each time stamp and use their average value as the measure for the comparison. It should be mentioned that this average time over all time stamps is only related to δ (the maximum occurrence window threshold). The reason is identifying the new minimal occurrences by updating E-tries in $\mathcal{M}_{in}$ dominates the execution time of the MESELO algorithm at each time stamp. However, the other two parameters min_sup and Δ are only used to maintain the frequent episode set at every time stamp, which consumes only a small fraction of time since the set of frequent and infrequent episodes usually vary slightly at two consecutive time stamps.

With all aforementioned descriptions, we show the execution time on 6 sequences with different values of δ when $min\_sup = 10$ and $\Delta = 2500$. In Figure 6, the execution time on different sequences are shown in sub-figures. In each sub-figure we also zoom in the area of the execution time for MESELO and MESELO-BS to clearly show their difference. This zooming-in figure is embedded inside each sub-figure.

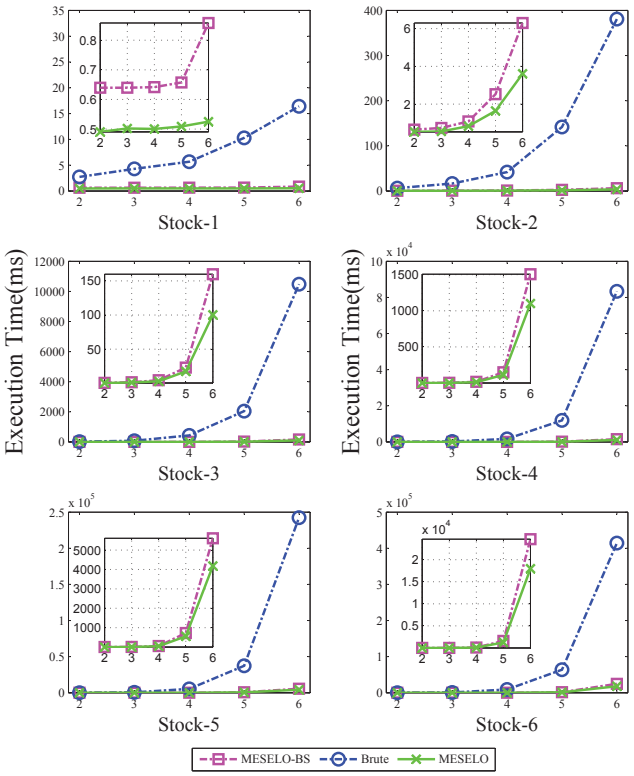


Fig. 6. The execution time for varying δ by fixing $min\_sup = 10$ and $\Delta = 2500$.

From the figure, we can clearly observe that MESELO-BS and MESELO have better performance compared with the BRUTE as the maximum occurrence window increases. The gap between the execution time of BRUTE and that of other two algorithms becomes more significant as δ increases. The MESELO series algorithms outperform BRUTE at least one magnitude of order and over 100 times when $\delta = 6$. Also, MESELO always outperforms MESELO-BS under different parameter settings, and the gap in terms of the execution time becomes more significant as δ increases. It clearly shows that the strategies used in MESELO to reduce the number of node expansions greatly reduce the execution time.

To further demonstrate the effectiveness of the strategies

for reducing the number of node expansions in MESELO, we compare the average size of the E-tries (the number of nodes in trie) from MESELO and MESELO-BS. Here, the episode tries completing all their node expansions are considered. With the strategies to reduce the node expansions, MESELO will have the smaller episode tries. Thus, in Figure 7 we show the relative size of the episode tries from MESELO compared to those from MESELO-BS. This relative size equals to 1 means that the two tries have the same size. As shown in this figure, the tries from MESELO is much smaller than those from MESELO-BS. This saving is much more significantly when δ increase. When $\delta = 10$ only $20\% - 50\%$ of the trie nodes are needed in MESELO compared to MESELO-BS. We can observe that the strategies for reducing node expansions in MESELO perform well especially in the cases with small event set size and large δ. The direct reason for that is there are more non-last-occurrence nodes in such a situation. We emphasize that a small event set as well as a big δ leads to a deep episode trie with limited kinds of events on the nodes. It makes many last-occurrence nodes transfer to the non-last-occurrence nodes during the updating process via the definition of last episode occurrence. Note that once a non-last-occurrence node is typed, it will not be expanded any further, and that is why the strategies can contribute much on space saving in these cases.

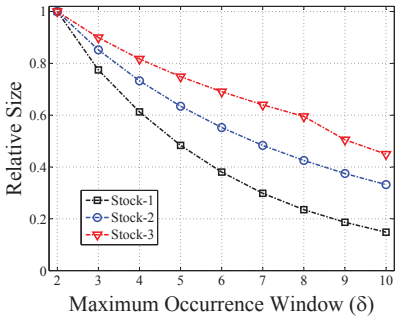


Fig. 7. The relative size of E-trie from MESELO compared to MESELO-BS.

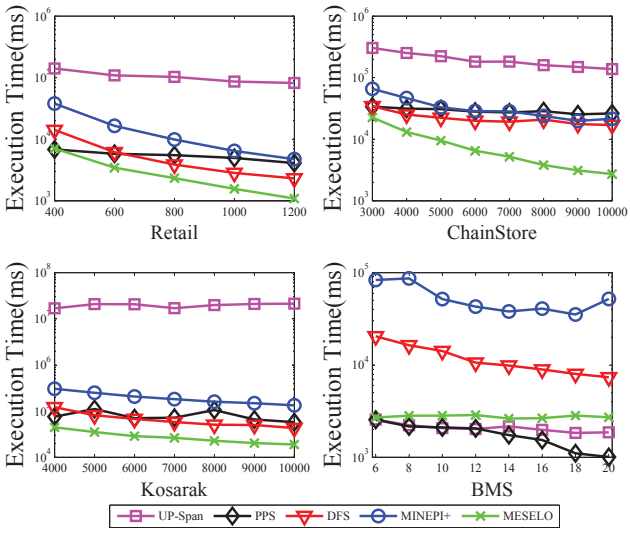


Fig. 8. The execution time vs. min_sup.

2) Batch Mode Comparison: Figure 8 shows the execution time on 4 different sequences with different settings of min_sup when $\delta = 3$. The scale of y-axis is in log style in Figure 8. We can see that MESELO holds the lowest value in execution time compared with the other algorithms in 3 of 4 data sets with different values of min_sup. In the data sets of Retail and ChainStore, the gap between MESELO and other algorithms become more significant as min_sup increases. The bigger value of min_sup remove more infrequent events for MESELO. Since the time complexity of MESELO is $O(m^{\delta})$ (where m is the size of event set at a time stamp), the reduction in the size of event set will significantly reduce the execution time. In the Kosarak data set, MESELO still outperforms all baselines. However, in the BMS data set PPS and UP-Span are slightly better than MESELO, and their performances are similar with the same order of magnitude.

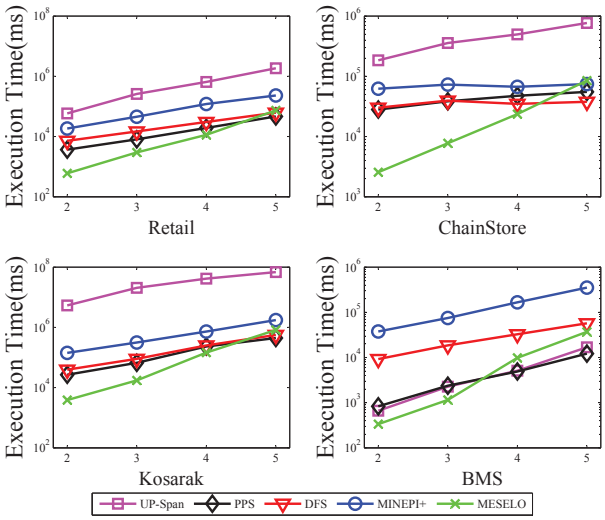


Fig. 9. The execution time vs. δ.

Figure 9 shows the execution time on the 4 data sets with different settings of δ when min_sup is fixed (i.e. min_sup = 400 in Retail, min_sup = 3000 in ChainStore, min_sup = 4000 in Kosarak and min_sup = 6 in BMS). Since the 4 sequences have different length, we select different values of min_sup for different data sets such that $min\_sup/|\vec{S}|$ is in the range of $[0.01\%, 0.45\%]$. The scale of y-axis is in log style in Figure 9. As shown in the figure, the execution time of all these methods increases as δ increases. MESELO outperforms the baselines when $\delta \leq 4$ except on BMS data set. The steeper growth slope of MESELO compared with other algorithms indicates that MESELO is more sensitive to δ.

By looking closer on BMS data set, we observe that UP-Span performs extremely well compared with its performance on other three data sets. We conjecture what makes this data set different is because its distinct smaller event set, shorter sequence length, and most importantly, obviously less number of events per timestamp. By analyzing all compared batch mode FEM algorithms, we learn that the performance of UP-Span and PPS is related to the number of events per timestamp, and BMS has obviously less value on such criterion.

The comparison on batch mode data show that: 1) the overall performance of MESELO for batch mode episode mining is better than the existing state-of-the-art methods; 2) MESELO is more sensitive to min_sup and δ. It is more appropriate for discovering short episodes in which the time interval between the first and the last events is not big.

VIII. Conclusion and Future Work

In this paper, we formulate the online frequent episode mining problem, which is especially useful to time-critical applications with growing sequences. We propose an efficient algorithm (named MESELO) for this problem. By the concept of last episode occurrence, MESELO can detect the minimal episode occurrences without performing a post-process checking. Also, utilizing the proposed episode trie, MESELO stores all the minimal episode occurrences in a compact way. Experiments on ten real data sets show the efficiency of the proposed algorithm, which is at least one magnitude of order faster than other baseline methods.

In our future work we will consider attributes on events to identify more interesting episodes. Additionally, to further increase time efficiency and reduce memory consumption, we will develop the *approximate* method for online frequent episode mining based on the current *exact* solution.

Acknowledgment

This work is supported by the National Natural Science Foundation of China (No. 61473273, 61473274, 61175052, 61203297), National High-tech R&D Program of China (863 Program) (No.2014AA015105, 2013AA01A606, 2012AA011003). Li is supported in part by NSF grants IIS-10-18865, CCF-11-17369 and IIS-14-08928 as well as HP Labs Innovation Research Awards. Any opinions, findings, and conclusions in this publication are those of the authors and do not necessarily reflect the views of the funding agencies.

References

[1] A. Achar, A. Ibrahim, and P. Sastry, "Pattern-growth based frequent serial episode discovery," *DKE*, 2013.

[2] A. Achar, S. Laxman, and P. Sastry, "A unified view of the apriori-based algorithms for frequent episode discovery," *KAIS*, 2012.

[3] X. Ao, P. Luo, C. Li, F. Zhuang, Q. He, and Z. Shi, "Discovering and learning sensational episodes of news events," in *WWW Companion*, 2014.

[4] M. Atallah, W. Szpankowski, and R. Gwadera, "Detection of significant sets of episodes in event sequences," in *ICDM*, 2004.

[5] B. Bouqata, C. D. Carothers, B. K. Szymanski, and M. J. Zaki, "Vogue: a novel variable order-gap state machine for modeling sequences," in *PKDD*, 2006.

[6] G. Casas-Garriga, "Discovering unbounded episodes in sequential data," in *PKDD*, 2003.

[7] J. H. Chang and W. S. Lee, "Finding recent frequent itemsets adaptively over online data streams," in *KDD*, 2003.

[8] Y. Chi, H. Wang, P. S. Yu, and R. R. Muntz, "Moment: Maintaining closed frequent itemsets over a stream sliding window," in *ICDM*, 2004.

[9] M. Chlistalla, B. Speyer, S. Kaiser, and T. Mayer, "High-frequency trading–better than its reputation?" *Deutsche Bank Research Briefing*, 2011.

[10] C. Giannella, J. Han, J. Pei, X. Yan, and P. S. Yu, "Mining frequent patterns in data streams at multiple time granularities," *Next generation data mining*, vol. 212, 2003.

[11] R. Gwadera, M. J. Atallah, and W. Szpankowski, "Reliable detection of episodes in event sequences," *KAIS*, 2005.

[12] C. Hidber, *Online association rule mining*, 1999.

[13] http://cucis.ece.northwestern.edu/projects/DMS/MineBench.html, "Minebench version 2.0 dataset and technical report."

[14] http://en.wikipedia.org/wiki/High-frequency_trading#cite_note-iosco-1, "High frequency trading."

[15] http://en.wikipedia.org/wiki/Trie, "Trie."

[16] http://fimi.cs.helsinki.fi/data/, "Frequent itemset mining implementations repository."

[17] http://forum.ai-directory.com/read.php?5,33, "The data mining forum."

[18] K.-Y. Huang and C.-H. Chang, "Efficient mining of frequent episodes from complex sequences," *Information Systems*, 2008.

[19] S. Laxman, "Discovering frequent episodes: fast algorithms, connections with hmms and generalizations," Ph.D. dissertation, Indian Institute of Science, 2006.

[20] S. Laxman, P. Sastry, and K. Unnikrishnan, "Discovering frequent episodes and learning hidden markov models: A formal connection," *IEEE TKDE*, 2005.

[21] ——, "Discovering frequent generalized episodes when events persist for different durations," *IEEE TKDE*, 2007.

[22] ——, "A fast algorithm for finding frequent episodes in event streams," in *KDD*, 2007.

[23] C. Leber, B. Geib, and H. Litz, "High frequency trading acceleration using fpgas," in *FPL*, 2011.

[24] C.-S. Leung and Q. I. Khan, "Dstree: a tree structure for the mining of frequent sets from data streams," in *ICDM*, 2006.

[25] C.-S. Leung, Q. I. Khan, and T. Hoque, "Cantree: a tree structure for efficient incremental mining of frequent patterns," in *ICDM*, 2005.

[26] Y. Li, J. Bailey, L. Kulik, and J. Pei, "Efficient matching of substrings in uncertain sequences," in *SDM*, 2014.

[27] X. Ma, H. Pang, and K.-L. Tan, "Finding constrained frequent episodes using minimal occurrences," in *ICDM*, 2004.

[28] G. S. Manku and R. Motwani, "Approximate frequency counts over data streams," in *VLDB*, 2002.

[29] H. Mannila and H. Toivonen, "Discovering generalized episodes using minimal occurrences." in *KDD*, 1996.

[30] H. Mannila, H. Toivonen, and A. Inkeri Verkamo, "Discovery of frequent episodes in event sequences," *DMKD*, 1997.

[31] N. Meger, C. Leschi, N. Lucas, and C. Rigotti, "Mining episode rules in stulong dataset," in *ECML/PKDD*, 2004.

[32] B. Mozafari, H. Thakkar, and C. Zaniolo, "Verifying and mining frequent patterns from large windows over data streams," in *ICDE*, 2008.

[33] A. Ng and A. W.-C. Fu, "Mining frequent episodes for relating financial events and stock trends," in *PAKDD*, 2003.

[34] D. Patnaik, P. Butler, N. Ramakrishnan, L. Parida, B. J. Keller, and D. A. Hanauer, "Experiences with mining temporal event sequences from electronic medical records: initial successes and some challenges," in *KDD*, 2011.

[35] D. Patnaik, S. Laxman, B. Chandramouli, and N. Ramakrishnan, "Efficient episode mining of dynamic event streams," in *ICDM*, 2012.

[36] R. Sipos, D. Fradkin, F. Moerchen, and Z. Wang, "Log-based predictive maintenance," in *KDD*, 2014.

[37] S. K. Tanbeer, C. F. Ahmed, B.-S. Jeong, and Y.-K. Lee, "Sliding window-based frequent pattern mining over data streams," *Information sciences*, 2009.

[38] N. Tatti and B. Cule, "Mining closed episodes with simultaneous events," in *KDD*, 2011.

[39] N. Tatti and J. Vreeken, "The long and the short of it: Summarising event sequences with serial episodes," in *KDD*, 2012.

[40] L. Wan, L. Chen, and C. Zhang, "Mining frequent serial episodes over uncertain sequence data," in *EDBT*, 2013.

[41] C.-W. Wu, Y.-F. Lin, S. Y. Philip, and V. S. Tseng, "Mining high utility episodes in complex event sequences," in *KDD*, 2013.

[42] J. X. Yu, Z. Chong, H. Lu, and A. Zhou, "False positive or false negative: Mining frequent itemsets from high speed transactional data streams," in *VLDB*, 2004.

[43] W. Zhou, H. Liu, and H. Cheng, "Mining closed episodes from event sequences efficiently," in *PAKDD*, 2010.

[44] H. Zhu, P. Wang, X. He, Y. Li, W. Wang, and B. Shi, "Efficient episode mining with minimal and non-overlapping occurrences," in *ICDM*, 2010.