Mining Precise-Positioning Episode Rules from Event Sequences

Xiang Ao, Ping Luo, *Member, IEEE*, Jin Wang, Fuzhen Zhuang, and Qing He, *Member, IEEE*

Abstract—Episode Rule Mining is a popular framework for discovering sequential rules from event sequential data. However, traditional episode rule mining methods only tell that the consequent event is likely to happen within a given time interval after the occurrence of the antecedent events. As a result, they cannot satisfy the requirement of many time sensitive applications, such as program security trading and intelligent transportation management due to the lack of fine-grained response time. In this study, we come up with the concept of *fixed-gap episode* to address this problem. A fixed-gap episode consists of an ordered set of events where the elapsed time between any two consecutive events is a constant. Based on this concept, we formulate the problem of mining *precise-positioning episode rules* in which the occurrence time of each event in the consequent is clearly specified. In addition, we develop a trie-based data structure to mine such precise-positioning episode rules with several pruning strategies incorporated for improving the performance as well as reducing memory consumption. Experimental results on real datasets show the superiority of our proposed algorithms.

Index Terms—Episode rule mining, gap-constrained episode, sequence mining

1 Introduction

Frequent episode mining (FEM) has emerged as a popular research topic in the data mining community. Given a single event sequence, FEM aims to identify all frequent episodes with the frequency larger than a given threshold. Here an *episode* (also known as serial episode [1]) is a totally ordered set of events. FEM techniques have been widely conducted into analysis of real-world data, such as alarm sequences in telecom networks [1], time-stamped fault reports in car manufacturing plants [2], web navigation logs [1], [3], customer transactions [4], text [5], [6], stock data [7], [8], [9], [10], and traffic data [10].

One basic problem in FEM is to find episode rules from frequent episodes. Given a frequent episode α, a valid episode rule in the form of $lhs \rightarrow rhs$ can be generated in a straightforward manner: The antecedent lhs is the prefix of α and the consequent rhs is the suffix event in α, if its confidence is larger than a user-specified threshold. In many real applications, episode rules are further generated from the discovered frequent episodes for decision making and predicting [1], [7], [11], [12], [13].

Example 1. Fig. 1 gives a running example for episode rule mining, where capital letters denote events and arabic numbers denote timestamps. In this sequence, we see three occurrences of episode $\langle \mathsf{D}, \mathsf{A}, \mathsf{B} \rangle$ when maximum occurrence window size threshold is set to 4. Then, if we take B as the consequent, an episode rule $\langle \mathsf{D}, \mathsf{A} \rangle \rightarrow \langle \mathsf{B} \rangle$ can be generated. This rule tells that it is within 2 time intervals after the occurrence of $\langle \mathsf{D}, \mathsf{A} \rangle$ that B will occur (with 100 percent probability).

However, with such a rule discovered by traditional FEM methods, we only know the approximate time range of the occurrence of rhs. We argue that such rules are not practically useful without specifying the exact time of rhs in real-world applications. This precise-positioning episode rule mining problem is particularly motivated by time-sensitive applications. We will describe two of such applications below.

The first is program security trading. Suppose the sequence in Fig. 1 describes daily change events of a stock market, and B refers to an event depicting the price of a stock increases on a day. In the above example, we cannot decide the exact time to buy this stock after the occurrence of $\langle \mathsf{D}, \mathsf{A} \rangle$ with only a time range of B will happen. If we buy the stock at the first day and hold it for two days after $\langle \mathsf{D}, \mathsf{A} \rangle$ happens, we might lose money if the stock slumps significantly on the first day and rebounds slightly (B occurs) on the next day. Here although this rule makes a correct prediction, we would still lose money. Similar scenario will appear as well if we purchase the stock at the second day. In this case, the episode rules discovered by traditional methods would be inapplicable for this application.

Another motivating application is intelligent transportation management. Suppose the event sequence in Fig. 1 describes a traffic condition over time, and the meaning of each event is that there is a traffic congestion at a certain crossroad. Then the above rule indicates the crossroad B will be congested after viewing traffic jams on the crossroad D and A sequentially. However, without more fine-grained information about the time interval of B, it is difficult to adjust the traffic signal at B for relieving congestion. If we adjust the traffic signal at B earlier than this time, some

- *X. Ao, P. Luo, F. Zhuang, and Q. He are with the Key Lab of Intelligent Information Processing of Chinese Academy of Sciences (CAS), Institute of Computing Technology, CAS, Beijing 100190, China, and the University of Chinese Academy of Sciences, Beijing 100049, China. E-mail: {aoxiang, luop}@ict.ac.cn, {zhuangfz, heq}@ics.ict.ac.cn.*
- *J. Wang is with the Computer Science Department, University of California, Los Angeles, CA 90095. E-mail: jinwang@cs.ucla.edu.*

Manuscript received 17 Mar. 2017; revised 27 Oct. 2017; accepted 7 Nov. 2017. Date of publication 14 Nov. 2017; date of current version 2 Feb. 2018. (Corresponding author: Xiang Ao.)
Recommended for acceptance by R. Cheng.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TKDE.2017.2773493

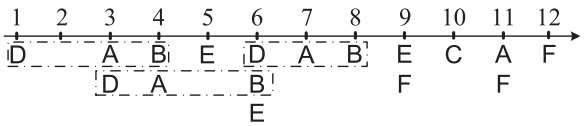


Fig. 1. The running example of event sequence.

negative effects might appear in the surrounding area. If we act later, a much more terrible jam at B might happen.

In the above two applications, traditional episode rules [1], [7] fail to exert their effects. In this paper, we study the problem of **Mi**ning **P**recise-positioning **E**pisode **R**ules (MIPER) where we need to specify the exact time instead of the approximate time range of the consequent events. To this end, we first introduce the concept of *fixed-gap episode*, which is defined as a tuple of events such that the time span between any two consecutive events is specified. Formally, the goal is to mine a *precise-positioning episode rule* $lhs \xrightarrow{\Delta t} rhs$ with the following three requirements: 1) the antecedent lhs is a minimal-occurrence episode [14]; 2) the consequent rhs is a fixed-gap episode; 3) the elapsed time between the last event in lhs and the first event in rhs is Δt.

We first discuss about the existence of fixed-gap episodes. As fixed-gap episodes require that the time span between two consecutive events is determinated, the occurrence of an event e_i will definitely induce the following event e_{i+1}. We assume that the time span between e_i and e_{i+1} follows a truncated Gaussian distribution such that

$$(t_{i+1} - t_i) \backsim N(\mu_i, \sigma^2) \; and \; t_{i+1} > t_i. \tag{1}$$

Such an assumption has been widely used to model similar real-world distributions [15]. Then the fixed-gap episodes can be generated from a stochastic process based on a chain reaction such that the occurrence of e_i results in the occurrence of its successor e_{i+1}. Here the mode of this distribution, namely $\mu_i > 0$, is the most probable value for the time span between e_i and e_{i+1}. Thus, the fixed-gap episode $(\langle e_i, e_{i+1}\rangle, \langle \mu_i\rangle)$ is likely to appear frequently in the event sequence, and its frequency would be higher than other analogous fixed-gap episodes with same events but different time spans, e.g., $(\langle e_i, e_{i+1}\rangle, \langle \mu_i \pm \sigma\rangle)$. It is clear that the fixed-gap episode $(\langle e_i, e_{i+1}\rangle, \langle \mu_i\rangle)$ is much more frequent than its variants for smaller variance, i.e., σ. As a result, it is likely that the precise-positioning episode rule candidates are mainly generated from such fixed-gap episode as the frequency of its variants cannot exceed the threshold. Otherwise, we may discover several fixed-gap episodes about e_i and e_{i+1} with different time spans but similar frequencies, and all of them might perform the consequent of candidate precise-positioning episode rules once they reach the frequency threshold.

While precise-positioning episode rules could provide richer information, the performance of mining such rules would suffer from "combination explosion". Due to the constraints of exact time, the number of candidates for precise-positioning episode rules would be much larger than that of traditional episode rules. Although this problem can be solved by extending traditional wild-card based methods [16], [17], [18], there would be serious efficiency degradation due to the exponential search space of entire possible patterns.

To address the issue of mining such rules, we first propose an enumeration based framework with the following steps: 1) mining frequent minimal-occurrence episodes on the whole sequence; 2) mining frequent fixed-gap episodes on the whole sequence; 3) for each pair of a minimal-occurrence episode and a fix-gap episode, concatenating them to generate a candidate of precise-positioning rule and then computing its confidence.

According to the definition of precise-positioning episode rule, the consequent must occur after the antecedent. Hence we can improve the proposed framework by mining fixed-gap episodes only after the occurrences of frequent minimal-occurrence episodes rather than over the whole sequence. Following this route, we develop a compact trie-based framework to mine precise-positioning episode rules directly from frequent minimal-occurrence episodes partitioning. In order to reduce the search space, we adopt downward closure-based pruning strategy on the trie structure. Meanwhile, we also make full use of intermediate results in trie nodes for subsequent traverses to avoid revisiting the original input sequence. To the best of our knowledge, this is the first work to mine episode rules with exact time information defined in their formulations.

The main contribution of this work is summarized as follows.

- We address the new problem of mining precise-positioning episode rules to satisfy the requirement of time-sensitive applications in the real world.
- We design a trie-based framework to compactly store valid precise-positioning episode rules and perform efficient mining.
- We propose effective pruning strategies to further reduce the processing time.
- We demonstrate the effectiveness of precise-positioning episode rules on two practical applications and the efficiency of the proposed algorithms based on real-world datasets.

The remainder of this paper is organized as follows. We discuss related work in Section 2. Section 3 presents preliminary definitions and problem statement. We design an enumeration framework to solve the MIPER problem in Section 4. In Section 5, we introduce a data structure PER-trie to improve the overall efficiency. In Section 6, we further propose detailed mining algorithms based on PER-trie for MIPER. We discuss a real-world application of our work in Section 7. We demonstrate the experimental studies and results in Section 8. We conclude the paper and discuss future work in Section 9.

2 Related Work

We are aware of several studies related to this work, including gap constrained pattern mining, frequent episodes and episode rules mining.

Gap Constrained Patterns. Gap constrained episode, also known as unbounded episode [3], specifies the maximal elapsed time between two neighbor events. Under the framework of gap constrained episode, the gap is still a range time interval instead of an exact value. Although we could make a gap constrained episode to support fixed-gap episode mining when the maximum gap constraint is set to 1, such algorithms fail to find all fixed-gap episodes with different time spans as our work did. The same problems also appear in gap constrained sequential patterns [19]. Another category of studies is mining sequential pattern with wild-card constraint, which is usually applied to discover patterns in string or biological sequences [16], [17], [18]. In this type of

pattern, the time interval between two continuous items in a pattern is also considered. It usually indicates minimum and maximum time intervals between two adjacent items. We may extend such methods to our problem by setting multiple gap-values to discover all fixed-gap patterns in the sequence. But it is obvious that such an approach is time consuming due to the exponential search space. Though [18] is able to discover rigid wild-card patterns with fixed gap constrains in biological sequences, it can only generate long patterns by convoluting known elementary patterns. Our algorithms, however, can discover all fixed-gap episodes and valid precise-positioning episode rules without any prior knowledge.

Frequent Episode Mining. Mining frequent episodes from event sequence was first introduced by Mannila et al. [1] where episodes are defined as directed acyclic graphs and two kinds of counting support are considered, i.e., sliding windows and minimal occurrence. After that, various frequency measures are defined to discover different kinds of episodes according to different applications, and minimal occurrence is one of widely used measures [4], [7], [14], [20], [21]. Mining general episodes can be intricate and computing-intensive, for example, discovering whether a sequence covers a general episode is **NP**-hard [22]. Existing algorithms can be categorized into two types, namely breadth-first enumeration [1], [20], [23] and depth-first enumeration methods [4], [7], [24]. Among them, the depth-first enumeration methods can be used to discover episode minimal occurrence. However, most of these algorithms require a post-processing step to verify detected occurrences [9], which still have a significant space for improvement. On the other hand, researches have been focused on mining subclasses of episodes, for example, serial episodes [25], closed episodes [6], [22], maximal episodes [21], episodes with unique labels [26], [27].

Episode Rule Mining. Inchoate episode rules are considered a "second-stage" output derived from frequent episodes [1], [7]. Episode rules are usually represented in the form of a time range in which the consequent will happen. Meger et al. [13], [28] constructed episode rules with gap constraint episodes and proposed the algorithm to find the optimal window size. Fournier-Viger et al. [29] mined partially-ordered sequential rules in which items are unordered in both the antecedent and the consequent. Such kind of rules may improve prediction accuracy in some applications. Lin et al. [30] focused on the utility of episode rules and proposed an algorithm to directly mine high utility episode rules. Our work focuses on mining precise-positioning episode rules motivated by critical applications in which we need to trigger possible right responses at a more fine-grained right time.

3 Preliminaries and Problem Statement

In this section, we first give some preliminary definitions in frequent episode mining (Definitions 1, 2, 3, 4, and 5) [1], [7], [9], [20]. Then, we propose some new concepts about precise-positioning episode rules (Definitions 6, 7, 8, 9, 10, 11, and 12), and finally formulate the mining problem.

3.1 Preliminaries

Definition 1 (Event Sequence). *Let $\mathcal{E}$ be a finite set of events. An* event sequence, *denoted $\vec{S} = \langle (E_1, t_1), (E_2, t_2), \ldots, (E_n, t_n) \rangle$, is an ordered sequence of events, where each $E_i \neq \emptyset$ and $E_i \subseteq \mathcal{E}$ consists of all events associated with timestamp t_i, and $t_j < t_k$ for any $1 \leq j < k \leq n$.*

For example, Fig. 1 shows an event sequence $\vec{S} = \langle (\{\mathsf{D}\}, 1), (\{\mathsf{A}, \mathsf{D}\}, 3), (\{\mathsf{A}, \mathsf{B}\}, 4), (\{\mathsf{E}\}, 5), (\{\mathsf{B}, \mathsf{D}, \mathsf{E}\}, 6), (\{\mathsf{A}\}, 7), (\{\mathsf{B}\}, 8), (\{\mathsf{E}, \mathsf{F}\}, 9), (\{\mathsf{C}\}, 10), (\{\mathsf{A}, \mathsf{F}\}, 11), (\{\mathsf{F}\}, 12) \rangle$.

Definition 2 (Episode). *An* episode *α is defined as a non-empty totally ordered set of events of the form $\langle e_{\alpha_1}, \ldots, e_{\alpha_j}, \ldots, e_{\alpha_k} \rangle$ where $e_{\alpha_i} \in \mathcal{E}$ for all $i \in [1, k]$ and the event e_{α_i} occurs before the event e_{α_j} for any $1 \leq i < j \leq k$. An episode α of length k is referred to a k-episode.*

Definition 3 (Episode Occurrence). *Given an episode $\alpha = \langle e_{\alpha_1}, \ldots, e_{\alpha_j}, \ldots, e_{\alpha_k} \rangle$ and a sequence $\vec{S}, [t_{\alpha_1}, \ldots, t_{\alpha_i}, \ldots, t_{\alpha_k}]$ is an occurrence of α if and only if (1) e_{α_i} is an element of the event set E_{α_i} at time t_{α_i} for all $i \in [1, k]$; (2) $t_{\alpha_1} < t_{\alpha_2} < \cdots < t_{\alpha_k}$. The time window $[t_{\alpha_1}, t_{\alpha_k}]$ is called an* occurrence window *of α. In this study, we only consider the episode occurrences whose window size is smaller than a user-specified threshold δ, namely $t_{\alpha_k} - t_{\alpha_1} < \delta$. The set of all occurrences of α in the sequence $\vec{S}$ is denoted by* $\mathrm{ocSet}(\alpha)$.

For example, if $\delta = 6$ in Fig. 1, $\mathrm{ocSet}(\langle \mathsf{D}, \mathsf{A}, \mathsf{B} \rangle) = \{[1, 3, 4], [3, 4, 6], [3, 4, 8], [3, 7, 8], [6, 7, 8]\}$.

Definition 4 (Minimal Episode Occurrence (MEO)). *Consider two time windows $[t_i, t_j]$ and $[t'_i, t'_j]$. $[t'_i, t'_j]$ is* subsumed *by $[t_i, t_j]$ if $t_i \leq t'_i$ and $t'_j \leq t_j$. An occurrence window $[t_i, t_j]$ of an episode α is a* minimal episode occurrence *of α if no other occurrence window $[t'_i, t'_j]$ of α is subsumed by $[t_i, t_j]$.*

For example, $\mathrm{moSet}(\langle \mathsf{D}, \mathsf{A}, \mathsf{B} \rangle) = \{[1, 4], [3, 6], [6, 8]\}$ when $\delta = 6$ for the sequence in Fig. 1. The time window $[3, 8]$ contains occurrences of $\langle \mathsf{D}, \mathsf{A}, \mathsf{B} \rangle$, but it is not a minimal occurrence since $\langle \mathsf{D}, \mathsf{A}, \mathsf{B} \rangle$ also occurs in $[3, 6]$.

Definition 5 (Support of Episode). *The* support *of an episode α, denoted as $\mathrm{sup}(\alpha)$, is defined as the number of its distinct MEOs, i.e., $\mathrm{sup}(\alpha) = |\mathrm{moSet}(\alpha)|$. An episode is* frequent *if and only if its support is not less than a user-specified parameter min_sup.*

For example, the episode $\langle \mathsf{D}, \mathsf{A}, \mathsf{B} \rangle$ is frequent when $min\_sup = 3$ in Fig. 1.

3.2 Definitions and Problem Statement

Definition 6 (Fixed-Gap Episode). *A* fixed-gap episode *β is defined as a tuple in the form $(\langle e_{\beta_1}, \ldots, e_{\beta_i}, \ldots, e_{\beta_k} \rangle, \langle \Delta t_1, \ldots, \Delta t_i, \ldots, \Delta t_{k-1} \rangle)$ where $e_{\beta_i} \in \mathcal{E}$ for $i \in [1, k]$ and the event e_{β_i} occurs before the event e_{β_j} for any $1 \leq i < j \leq k$. Additionally, the time span of the occurring time between event $e_{\beta_{j+1}}$ and event e_{β_j} is $\Delta t_j, j \in [1, k-1]$.[1] We denote a fixed-gap episode with length k as fixed-gap k-episode.*

For example, in Fig. 1, $(\langle \mathsf{E}, \mathsf{A} \rangle, \langle 2 \rangle)$ is a fixed-gap 2-episode. The time span between E and A is 2.

Definition 7 (Fixed-Gap Episode Occurrence (FEO)). *Given a fixed-gap episode $\beta = (\langle e_{\beta_1}, \ldots, e_{\beta_i}, \ldots, e_{\beta_k} \rangle, \langle \Delta t_1, \ldots, \Delta t_i, \ldots, \Delta t_{k-1} \rangle), [t_{\beta_1}, \ldots, t_{\beta_i}, \ldots, t_{\beta_k}]$ is an* occurrence *(FEO) of β if and only if (1) e_{β_i} is an element of event set E_{β_i} at time t_{β_i} for all $i \in [1, k]$; (2) $t_{\beta_j} < t_{\beta_{j+1}}$ and $t_{\beta_{j+1}} - t_{\beta_j} = \Delta t_j$ for all $j \in [1, k-1]$. Similar to Definition 3, t_{β_1} and t_{β_k} constitute an* occurrence window *of β, which is denoted as $[t_{\beta_1}, t_{\beta_k}]$.*

1. A single event e is a kind of special fixed-gap episode, and we denote it as $(\langle e \rangle, null)$.

We can denote a FEO as $(\beta, [t_{\beta_1}, t_{\beta_k}])$ using its start time and end time. The set of all occurrences of β is denoted by $\mathrm{ocSet}(\beta)$. For example, $\mathrm{ocSet}((\langle \mathsf{E}, \mathsf{A}\rangle, \langle 2\rangle)) = \{[5, 7], [9, 11]\}$ in Fig. 1.

Definition 8 (Support of Fixed-Gap Episode). *The* support *of a fixed-gap episode β, denoted as* $\mathrm{sp}(\beta)$, *is defined as the number of its distinct occurrences, i.e.,* $\mathrm{sp}(\beta) = |\mathrm{ocSet}(\beta)|$.

Note that in Fig. 1 $(\langle \mathsf{D}, \mathsf{A}, \mathsf{B}\rangle, \langle 2, 1\rangle)$ (occurring in $[1, 4]$) and $(\langle \mathsf{D}, \mathsf{A}, \mathsf{B}\rangle, \langle 1, 2\rangle)$ (occurring in $[3, 6]$) are two different fixed-gap episodes. They are different in terms of the time spans between adjacent events. However, $[1, 4]$ and $[3, 6]$ are two different minimal occurrences of the same episode $\langle \mathsf{D}, \mathsf{A}, \mathsf{B}\rangle$ under the traditional episode definitions.

Definition 9 (Precise-Positioning Episode Rule). *A* precise-positioning episode rule *(PER) Γ is an implication of the form $\alpha \xrightarrow{\Delta t} \beta$, where $\alpha = \langle e_{\alpha_1}, \ldots, e_{\alpha_l}\rangle$ is a frequent minimal occurrence based episode, and $\beta = (\langle e_{\beta_1}, \ldots, e_{\beta_k}\rangle, \langle \Delta t_1, \ldots, \Delta t_{k-1}\rangle)$ is a fixed-gap episode. Also, it must satisfy the following two conditions: (1) the elapsed time between e_{α_l} and e_{β_1} is a fixed value Δt; (2) For a given threshold ϵ, $\sum_{i=1}^{k-1} \Delta t_i + \Delta t \leq \epsilon$. Here, ϵ is a user-specified threshold named* threshold of maximum window size for consequent occurrence. *The function of ϵ is to decrease the computation burden in case that the distance between the consequent and the antecedent of a same rule is too large.*

For example, $\langle \mathsf{D}, \mathsf{A}\rangle \xrightarrow{2} (\langle \mathsf{E}, \mathsf{F}\rangle, \langle 3\rangle)$ is a PER when ϵ is set to 5. It means that once $\langle \mathsf{D}, \mathsf{A}\rangle$ happens, two time intervals later E will happen, and then F will happen three more time intervals later. Note that we do not limit the number of events in the consequent as traditional episode rules do. But the length of the consequent of a PER could be ϵ at most.

Definition 10 (Occurrence of PER). *Given a precise-positioning episode rule $\Gamma = \alpha \xrightarrow{\Delta t} \beta$, a MEO of $(\alpha, [t_{\alpha_1}, t_{\alpha_p}])$, and a FEO of $(\beta, [t_{\beta_1}, t_{\beta_q}])$, we call $[t_{\alpha_1}, t_{\alpha_p}, t_{\beta_1}, t_{\beta_q}]$ the* occurrence *of Γ if and only if $t_{\beta_1} - t_{\alpha_p} = \Delta t$. This occurrence is denoted as $(\Gamma, [t_{\alpha_1}, t_{\alpha_p}, t_{\beta_1}, t_{\beta_q}])$, and t_{α_1} and t_{β_q} are called* start time *and* end time, *respectively. The set of all occurrences of Γ is denoted as* $\mathrm{ocSet}(\Gamma)$.

Definition 11 (Support of PER). *The* support *of precise-positioning episode rule $\Gamma = \alpha \xrightarrow{\Delta t} \beta$, denoted as* $\mathrm{sp}(\Gamma)$, *is defined as the number of its distinct occurrences, i.e.,* $\mathrm{sp}(\Gamma) = |\mathrm{ocSet}(\Gamma)|$.

For example, the support of $\langle \mathsf{D}, \mathsf{A}\rangle \xrightarrow{2} (\langle \mathsf{E}, \mathsf{F}\rangle, \langle 3\rangle)$ is 2 in the example in Fig. 1 when $\delta = 3$ and $\epsilon = 5$.

Definition 12 (Confidence of PER). *Given a precise-positioning episode rule $\Gamma = \alpha \xrightarrow{\Delta t} \beta$, the* confidence *of Γ is defined as*

$$\mathrm{conf}(\Gamma) = \frac{\mathrm{sp}(\Gamma)}{\mathrm{sp}(\alpha)}. \quad (2)$$

A precise-positioning episode rule is called valid *if and only if its antecedent is frequent and its confidence is not less than a user-specified minimum confidence threshold min_conf.*

For example, the PER $\langle \mathsf{D}, \mathsf{A}\rangle \xrightarrow{2} (\langle \mathsf{E}, \mathsf{F}\rangle, \langle 3\rangle)$ is valid with its confidence of 67 percent for the sequence in Fig. 1 when $\delta = 3$ and $\epsilon = 5$.

Based on the above definitions, we formulate the problem of MIPER as follows:

Problem Statement of MIPER. Given an event sequence $\vec{S}$ (Definition 1), the problem of *valid precise-positioning episode rule mining* is to find all valid PER on $\vec{S}$ satisfying the following four user-specified parameters: the minimum support threshold min_sup (Definition 5), the minimum confidence threshold min_conf (Definition 12), the threshold of maximum window size for antecedent occurrence δ (Definition 3) and the threshold of maximum window size for consequent occurrence ϵ (Definition 9).

4 The Enumeration Approach for MIPER

In this section, we introduce an enumeration approach, denoted as MIP-ENUM, to solve the MIPER problem. We first propose the enumeration framework and then design a method to mine frequent fix-gap episodes. In order to show a running example of our proposed methods, we always set the parameters as $min\_sup = 3$, $\delta = 4$, $\epsilon = 5$ and $min\_conf = 0.6$ from Sections 4, 5, and 6 unless otherwise specified.

4.1 Framework of MIP-ENUM

The basic idea of MIP-ENUM is to enumerate PER candidates by concatenating discovered MEOs with FEOs and subsequently filter the infrequent ones according to their confidence values. The pseudo-code of MIP-ENUM is shown as Algorithm 1.

Algorithm 1. MIP-ENUM

Input: $\vec{S}$: the event sequence
min_sup: threshold of minimum support
δ: threshold of maximum window size for antecedent occurrence
min_conf: threshold of minimum confidence
ϵ: threshold of maximum window size for consequent occurrence
Output: the set of valid PER

```
1  begin
2    mining frequent minimal-occurrence episodes from S⃗
       with frequency threshold of min_sup and occurrence
       window size threshold of δ
3    mining frequent fixed-gap episodes from S⃗ with
       frequency threshold of min_sup × min_conf and
       occurrence window size threshold of ε
4    ⊎_{1≤i≤n} A^i ← discovered MEOs by their end time
5    ⊎_{1≤j≤n} C_j ← discovered FEOs by their start time
6    foreach A^i do
7      for all (α, [t_{α_1}, t_i]) ∈ A^i do
8        for Δt = 1 to ε do
9          for (β, [t_{i+Δt}, t_{β_q}]) ∈ C_{i+Δt} do
10           generate a PER occurrence by
               (α --Δt--> β, [t_{α_1}, t_i, t_{i+Δt}, t_{β_q}])
11   filter PER candidates whose confidence is less than
       min_conf and return
12 end
```

In order to perform candidate generation, we first mine both frequent episodes and fixed-gap episodes from the input sequence by setting different frequency and occurrence window size thresholds (Lines 2 and 3). Next, we group the MEOs and FEOs by their end time and start time, respectively. In particular, denoted by $\mathcal{A}^i$ as the set of MEOs whose end time is t_i and $\mathcal{C}_j$ as the set of FEOs whose start time is t_j (Line 4 and 5), where $i, j \in [1, n]$ and n is

the length of the sequence. The enumeration process is then performed, and a PER occurrence, $(\alpha \xrightarrow{\Delta t} \beta, [t_{\alpha_1}, t_i, t_{i+\Delta t}, t_{\beta_q}])$, can be formed by concatenating a MEO $(\alpha, [t_{\alpha_1}, t_i])$ from $\mathcal{A}^i$ and a FEO $(\beta, [t_{i+\Delta t}, t_{\beta_q}])$ from $\mathcal{C}_{i+\Delta t}$ (Lines 6-10). Finally, the filtering step is carried out to output all valid PERs.

For example, $(\langle \mathsf{D}, \mathsf{A} \rangle, [3, 4])$ is an element of $\mathcal{A}^4$ and $((\langle \mathsf{E}, \mathsf{F} \rangle, \langle 3 \rangle), [6, 9])$ is an element of $\mathcal{C}_6$ from our running example sequence. When $\Delta t = 2$, we can generate a PER candidate $\Gamma = \langle \mathsf{D}, \mathsf{A} \rangle \xrightarrow{2} (\langle \mathsf{E}, \mathsf{F} \rangle, \langle 3 \rangle)$ with an occurrence of $\Gamma, [3, 4, 6, 9]$. $\langle \mathsf{D}, \mathsf{A} \rangle \xrightarrow{2} (\langle \mathsf{E}, \mathsf{F} \rangle, \langle 3 \rangle)$ is a valid PER in the running example with a confidence of 0.67.

4.2 Mining Frequent Fixed-Gap Episodes

In MIP-ENUM, mining frequent fix-gap episodes (Line 3 of Algorithm 1) is an essential phase. This phase can be accomplished by taking conventional FEM algorithms followed by filtering steps. However, it is inefficient because multiple infrequent fixed-gap episodes may aggregate as a frequent episode which may derive unnecessary computations when performing postprocessings. Hence, we propose an algorithm to efficiently produce frequent fixed-gap episodes.

The algorithm adopts a level-wise mining process [31] which always mines fixed-gap $k+1$-episodes based on the frequent fixed-gap k-episode. We show its pseudo-code in Algorithm 2. First, the events whose frequency exceed the minimum threshold are added into a set $\mathcal{L}_k$ for further iterations. They are also kept to the output set $\mathcal{G}$ since they are parts of frequent fixed-gap episodes (Lines 3-5). Next, we begin the level-wise mining process. In more detail, for every frequent fixed-gap k-episode β in $\mathcal{L}_k$, every occurrence of β is considered independently. For each FEO $(\beta, [t_{\beta_1}, t_{\beta_k}])$ in $\mathrm{ocSet}(\beta)$, we scan every event set E_i on $\vec{S}$ where $i \in [t_{\beta_k} + 1, t_{\beta_1} + \epsilon - t_{\beta_k}]$ and generate new fixed-gap $k+1$-episode occurrences (Lines 8-11). After that, we filter the infrequent fixed-gap $k+1$-episode and keep the frequent ones for the next iteration.

Algorithm 2. Mining Frequent Fixed-Gap Episodes

Input: $\vec{S}$: the event sequence
min_freq: threshold of minimum frequency
ϵ: threshold of maximum occurrence window size
Output: $\mathcal{G}$: the set of frequent fixed-gap episodes

```
1  begin
2    initialize L_k, G ← ∅
3    foreach event e ∈ S such sp(e) ≥ min_freq do
4    L_k ← L_k ∪ e
5    G ← G ∪ L_k
6    while L_k ! = ∅ do
7      initialize L_{k+1} ← ∅
8      foreach frequent fixed-gap k-episode β in L_k do
9        foreach (β, [t_{β_1}, t_{β_k}]) ∈ ocSet(β) do
10         for i = t_{β_k} + 1 to t_{β_1} + ε − t_{β_k} do
11           scan event set E_i on S to generate new fixed-gap
             k + 1-episodes occurrences
12     L_{k+1} ← frequent fixed-gap k + 1-episodes
13     G ← G ∪ L_{k+1}
14     L_k ← L_{k+1}
15   return G
16 end
```

Take $\beta = (\langle \mathsf{B}, \mathsf{E} \rangle, \langle 1 \rangle)$ as an example. We have $\mathrm{ocSet}(\beta) = \{[4, 5], [8, 9]\}$. For the FEO $((\langle \mathsf{B}, \mathsf{E} \rangle, \langle 1 \rangle), [4, 5])$, we scan the event sets E_6-E_9. Then the event sets E_{10}, E_{11} and E_{12} are checked when we consider the FEO $((\langle \mathsf{B}, \mathsf{E} \rangle, \langle 1 \rangle), [8, 9])$. Finally we can get a new frequent fixed-gap 3-episode $(\langle \mathsf{B}, \mathsf{E}, \mathsf{A} \rangle, \langle 1, 2 \rangle)$ whose support is two.[2]

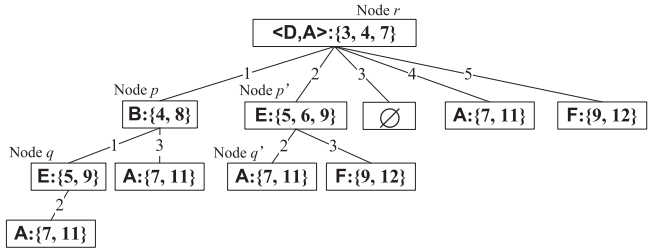


Fig. 2. Complete PER-trie of $\mathcal{T}_{\langle \mathsf{D},\mathsf{A} \rangle}$.

Complexity. For the input sequence with length n, if the maximum size of an event set on the input sequence is m, MIP-ENUM will output $O(nm(m+1)^{\epsilon-1})$ candidate fixed-gap episodes. Moreover, there are $O(nm^{\delta})$ possible frequent minimal-occurrence episodes in the input sequence, MIP-ENUM thus may generate $O(\epsilon n^2 m^{\delta+1}(m+1)^{\epsilon-1})$ PER candidates. The number is rather large in most cases.

From the above analysis, we can see that the MIP-ENUM method may generate immoderate amounts of fixed-gap episodes as well as more PER candidates and is thus generally inefficient. The reason is because that MIP-ENUM mines frequent fixed-gap episodes over the whole input sequence, and generates PER candidates through a quadratic enumeration. Therefore, we need to find a more efficient way to solve this problem.

5 Precise-Positioning Episode Rule Trie

In this section, we introduce a trie-based framework, named PER-trie, to store precise-positioning episode rules compactly. Such data structure is also helpful for facilitating the efficiency of discovering valid PERs. We can also take advantage of its features to improve the mining process. Detailed algorithms will be presented in Section 6.

5.1 Structure of PER-trie

Given a frequent minimal-occurrence episode α, a *PER-trie*, denoted as $\mathcal{T}_\alpha$ is a trie-like data structure which stores valid precise-positioning episode rules whose antecedent is α.

The root node r of a PER-trie $\mathcal{T}_\alpha$, denoted by $(r.episode{:}r.tlist)$, consists of two fields: the episode field $r.episode$ and the end time set field $r.tlist$. Here $r.episode$ registers the minimal-occurrence episode α; and $r.tlist$ records all the end times of minimal occurrence of α, i.e., $r.tlist = \{t_j | [t_i, t_j] \in \mathrm{moSet}(\alpha)\}$.

Example 2. Fig. 2 shows a PER-trie of $\mathcal{T}_{\langle \mathsf{D},\mathsf{A} \rangle}$ in which the root node r is $(\langle \mathsf{D}, \mathsf{A} \rangle{:}\{3, 4, 7\})$ since we have $\mathrm{moSet}(\langle \mathsf{D}, \mathsf{A} \rangle) = \{[1, 3], [3, 4], [6, 7]\}$.

The non-root node q, denoted by $(q.event{:}q.tlist)$, consists of two fields: the event field $q.event$ and the occurrence time set field $q.tlist$. Here $q.event$ registers which event this node represents, and $q.tlist$ is a set containing the occurrence time of such event after a fixed *distance* to the elements of $p.tlist$, where p is the parent node of q.

2. Since we set $min\_sup = 3$ and $min\_conf = 0.6$ for running examples, the lower bound of frequency of frequent fixed-gap is thus derived to $2 = \lceil 3 \times 0.6 \rceil$.

The edge between a parent node p and its child node q has a *distance* field, which is a positive integer to record the length between the two nodes. We denote it as $\mathrm{d}(p, q)$. In the PER-trie, for any element $t' \in q.tlist$, there exists an element $t \in p.tlist$ such that $t' - t = \mathrm{d}(p, q)$.

Example 3. The node p in Fig. 2 shows a non-root node of (B:$\{4, 8\}$). The distance between it and the parent root node r is 1. Since $r.tlist = \{3, 4, 7\}$, $p.tlist$ records the occurrence time of B on the 1-interval-later time stamps of $r.tlist$, namely $\{4, 5, 8\}$. Thus $p.tilst = \{4, 8\}$. For another node $q = (\mathsf{E}{:}\{5, 9\})$, we have $d(p, q) = 1$, thus $q.tlist$ records the occurrence time of event E 1-interval-later time stamps 4 and 8 which is $\{5, 9\}$.

Then given a PER-trie $\mathcal{T}_\alpha$, a PER can be represented by a path from the root node to any non-root node. Specifically, we assume that the path from the root r to a non-root node q is through the nodes $q_1 \Rightarrow q_2, \ldots, \Rightarrow q_k$. Then, the node q refers to a PER $\alpha \xrightarrow{d(r,q_1)} (\langle q_1.event, \ldots, q_k.event, q.event\rangle, \langle \mathrm{d}(q_1, q_2), \mathrm{d}(q_1, q_2), \ldots, \mathrm{d}(q_k, q)\rangle)$.

Example 4. The path from the root node r to the node q in Fig. 2 denotes a PER $\langle \mathsf{D}, \mathsf{A}\rangle \xrightarrow{1} (\langle \mathsf{B}, \mathsf{E}\rangle, \langle 1, 1\rangle)$. Similarly, the node q' refers to another PER $\langle \mathsf{D}, \mathsf{A}\rangle \rightarrow (\langle \mathsf{E}, \mathsf{A}\rangle, \langle 2, 2\rangle)$.

5.2 Features of PER-trie

Here we introduce two essential features of PER-trie. Such features will be helpful to further improve the performance of the mining process.

Lemma 1 (Support Counting). *Given a non-root node q' on a PER-trie $\mathcal{T}_\alpha$, the support of the PER associated with q' is equal to the cardinality of the occurrence time set field of q', i.e., $|q'.tlist|$.*

Proof. Without loss of generality, We denote the root node of $\mathcal{T}_\alpha$ by r, and denote the rule associated to an arbitrary non-root node q' by Γ.

Consider any pair of parent-child node in the trie, e.g., p (parent) and q (child), for any element $t_j \in q.tlist$, we can always find a $t_i \in p.tlist$ satisfying $t_i + \mathrm{d}(p, q) = t_j$. In other words, we have $p.event$ and $q.event$ occurring on t_i and t_j respectively and $t_j - t_i = \mathrm{d}(p, q)$ for any pair of parent-child node p and q on the trie. Hence, for every element $t_j \in q'.tlist$, we can always derive to a unique occurrence of Γ by backtracking the PER-trie. Thus we have $\mathrm{sp}(\Gamma) \geq |q'.tlist|$.

On the other hand, $\mathrm{sp}(\Gamma) \leq |q'.tlist|$ is straightforward. Suppose there exists another occurrence of Γ whose end time is not contained in $q'.tlist$. Then we could find the end time of such occurrence of α (Γ's antecedent) *not* in $r.tlist$. However, $r.tlist$ registers all the end times in $\mathrm{moSet}(\alpha)$ by its definition. It contradicts with our assumption. Hence, all the end times of Γ's occurrences must appear in $q'.tlist$, and we have $\mathrm{sp}(\Gamma) \leq |q'.tlist|$.

Hence, $\mathrm{sp}(\Gamma) = |q'.tlist|$. The original proposition is thus proved. □

Lemma 2 (Downward Closure). *Given a pair of parent-child node p (parent) and q (child) in a PER-trie, the support of the rule associated with p is not less than that of q.*

Proof. It is straightforward to get $q.tlist \subseteq \{t_k | t_k = t_i + \mathrm{d}(p, q)\}$ where $t_i \in p.tlist$. Hence, we have $|q.tlist| \leq |p.tlist|$. According to the Lemma 1, $|q.tlist|$ and $|p.tlist|$ are equal to the support of PERs associated with q and p, respectively. The original proposition is thus proved. □

Example 5. The support of the PER $\Gamma = \langle \mathsf{D}, \mathsf{A}\rangle \xrightarrow{2} (\langle \mathsf{E}\rangle, \langle 2\rangle)$ associated to the node $p' = (E{:}\{5, 6, 9\})$ in Fig. 2 is 3. While for its child, $q' = (A{:}\{7, 11\})$, its corresponding PER $\Gamma' = \langle \mathsf{D}, \mathsf{A}\rangle \xrightarrow{2} (\langle \mathsf{E}, \mathsf{A}\rangle, \langle 2, 2\rangle)$ has a support of 2. Here $\mathrm{sp}(\Gamma') \leq \mathrm{sp}(\Gamma)$.

Lemma 2 ensures the support of a PER associated with a node p in a PER-trie cannot be less than the support of all its descendant nodes. Hence we can design pruning techniques based on such lemma in the mining process.

6 Directly Mining Valid PER

In this section, we design a more efficient approach MIP-TRIE with the help of the proposed PER-trie structure in the above section. Unlike MIP-ENUM, MIP-TRIE can directly discover valid precise-positioning episode rules from the event sequence without generating PER candidates. We first propose the overall framework of MIP-TRIE and then introduce two optimized algorithms MIP-TRIE(DFS) and MIP-TRIE(PRU).

6.1 Overview of MIP-TRIE

The key ingredients to the efficiency of MIP-TRIE are (i) frequent minimal-occurrence episodes partitioning strategy, (ii) the usage of PER-trie to store valid PERs and (iii) novel algorithms for mining valid PERs without generating candidates.

The MIP-TRIE algorithm is divided into two phases. The first phase is to mine frequent minimal-occurrence episodes as possible antecedents of PER. The second phase, subsequently, is to mine valid PERs based on each possible antecedent. We give the pseudo-code of the framework of MIP-TIRE in Algorithm 3.

Algorithm 3. MIP-TRIE Framework

Input: $\vec{S}$: the event sequence
min_sup: threshold of minimum support
δ: threshold for the antecedent occurrence window size
min_conf: threshold of minimum confidence
ϵ: threshold for the consequent occurrence window size
Output: $\mathcal{R}$: the set of valid PER

```
1  begin
2    initialize R ← ∅
3    A ← mining frequent minimal-occurrence episodes from S
     with the thresholds of min_sup and δ
4    foreach α ∈ A do
5      ET_α ← the set of end time of each MEO of α
6      R ← R ∪ MIP-TRIE(α, ET_α, ε, sp(α) × min_conf)
7    return R
10 end
```

It first mines frequent minimal-occurrence episodes over the whole input sequence (Line 3). Next, an internal procedure MIP-TRIE is invoked for every frequent minimal-occurrence episode α. And its output is a complete PER-trie which stores all valid PERs in the input sequence whose antecedent is α.

The MIP-TRIE procedure takes the following four parameters as input: The first parameter is the antecedent of possible rules, e.g., α. The second one refers to the end time set of the antecedent's MEOs. The third, ϵ, is the maximum

window size for the consequent occurrences. And the last one refers to the minimum frequency threshold that makes a candidate PER valid. For example, if α is considered as the antecedent, then this threshold should be set to $\mathrm{sp}(\alpha) \times min\_conf$ (Line 6 in Algorithm 3).

MIP-TRIE can take advantage of the minimal-occurrence episode based partitioning. Compared with MIP-ENUM, MIP-TRIE only needs to discover corresponding fixed-gap episodes on subsequences after antecedents rather than the whole input sequence, which alleviates the number of useless fixed-gap episodes. Additionally, MIP-TRIE updates the minimum frequency threshold for fixed-gap episodes after dealing with each episode α. In this way, we can also reduce the processing time since more lenient parameter settings will lead to longer execution times.

6.2 Depth-First Construction of PER-trie

The internal procedure MIP-TRIE in Algorithm 3 returns a PER-trie storing all valid PERs of a given antecedent. The algorithm in fact transfers the valid PER mining process to a *complete* PER-trie construction for each frequent minimal-occurrence episode. In this section, we give a DFS-based approach to construct a complete PER-trie. We denote such algorithm as MIP-TRIE(DFS). This algorithm performs a depth-first enumeration over the trie structure. The main idea is to recursively expand a frequent sequence s (of length l) to generate all frequent sequences of length $l+1$ with s as the prefix [32]. Here in our MIP-TRIE(DFS), we start from the root node of a PER-trie, and recursively expand nodes on the PER-trie until no more nodes can be expanded.

We give its pseudo-code in Algorithm 4. It first creates a root node r with a given antecedent α (Line 2). Then, an internal function ExtendNode is invoked starting from r to perform node expansions (Line 3). For a node q to be extended, it first calculates the distance $d(q, r)$. Then upper bound of the number of enumerations is correspondingly $\epsilon - d(q, r)$, and the distance of the root node to itself is 0 (Lines 2 and 3). Next, it conducts on multiple rounds of enumerations over the original event sequence. First, an offset value is appended to every element in $q.tlist$ to get a set of variables $position\_List$ (Line 4). All event sets occurring at time $position\_List$ are then collected. Among these event sets, it searches events with frequency higher than min_freq and store them to a set E' (Line 5). For any event $e' \in E'$, it extends a new node q' as the child of q, which represents a new discovered valid PER. Once q' is generated, the same function is immediately invoked recursively in which q' is taken as the node for the next extension (Lines 6-11). When such an event set E' cannot be found any longer, the construction process will stop.

Algorithm 4. MIP-TRIE(DFS)

Input: α: a frequent minimal-occurrence episode
ET_α: the set of end time of each MEO of α
ϵ: the threshold of maximum window size for consequent occurrence
min_freq: minimum frequency threshold generating a valid PER
Output: $\mathcal{T}_\alpha$: the complete PER-trie w.r.t. antecedent α

1 **begin**
2 build a root node r of $\mathcal{T}_\alpha$ where $r.episode \leftarrow \alpha$ and $r.tlist \leftarrow ET_\alpha$
3 ExtendNode(r, ϵ, min_freq)
4 **end**

Function ExtendNode(q, ϵ, min_freq)

Input: q: node to be extended
ϵ: the threshold of maximum window size for consequent occurrence
min_freq: minimum frequency threshold generating a valid PER
Output: $\mathcal{T}_\alpha$: updated PER-trie w.r.t antecedent α

1 **begin**
2 $d \leftarrow$ distance between the root r and the node q
3 **for** $i = 1$ to $(\epsilon - d)$ **do**
4 $position\_List \leftarrow \{t' | t' = t + i, t \in q.tlist\}$
5 scan $\vec{S}$ on each time stamp in $position\_List$ and get an event set E' such that the frequency of every event in E' is not less than min_freq
6 **foreach** $e' \in E'$ **do**
7 $q'.event \leftarrow e'$
8 $Q \leftarrow$ the set of occurrence times of e' in $position\_List$
9 $q'.tlist \leftarrow Q$
10 put q' as the child of q
11 ExtendNode(q', ϵ, min_freq)
12 **end**

Example 6. Consider p = (B:{4, 8}) in Fig. 2, the distance between the node p and the node r = (⟨D, A⟩:{3, 4, 7}) is 1, the upper bound for enumerations of p is thus 4. Next, we begin four rounds of enumerations. When $i = 1$, event sets E_5 and E_9 are collected, and we can expand a child node q = (E:{5, 9}) of p. Then the node q is immediately extended, and we can similarly derive to a child node (A:{7, 11}) whose distance to q is 2. Finally, the complete PER-trie of $\mathcal{T}_{\langle \mathsf{D},\mathsf{A} \rangle}$ is shown as Fig. 2.

6.3 Improved Construction of PER-trie

Although MIP-TRIE(DFS) can significantly outperform MIP-ENUM by avoiding the generation of candidates, there is still much room for improvement. While performing node expansions, MIP-TRIE(DFS) may scan repetitive positions over the input sequence to collect event sets as well as their frequencies. For example, as shown in Fig. 2 event set $E_7 = \{\mathsf{A}\}, E_8 = \{\mathsf{B}\}$ and $E_{11} = \{\mathsf{A}, \mathsf{F}\}$ will be re-collected when expanding the root node r by distance 4 and the node p' by distance 2. Meanwhile, E_7 and E_{11} will be re-collected when expanding the node p by distance 3 and the node q by distance 1. Among them only the event A exceeds the minimum frequency threshold. However, we can only filter event B and F after we redundantly count their frequencies in MIP-TRIE(DFS).

In order to eliminate duplicate event sets and sequence scans, we propose an improved approach, denoted as MIP-TRIE(PRU). The basic idea is to adjust the order of node extensions and reuse intermediate results of the established PER-trie as much as possible during the traversing process.

MIP-TRIE(PRU) adopts an improved strategy for building complete PER-trie, and we give the pseudo-code of such method in Algorithm 6. Specifically, it first expands the root node r via the same way as that in Algorithm 4 (Lines 2-9) and performs ϵ iterations in the outer loop. But unlike the DFS based method, when reaching a new child node (denoted by q') of r, a top-down inner traverse procedure is triggered. That is, for every non-root node w on the established PER-trie except that q', it computes a variable set to record the time stamps that are needed to be checked (Line 13). Then if the cardinality of the intersection of such a variable set and $q'.tlist$ exceeds the minimum frequency threshold, a new child node

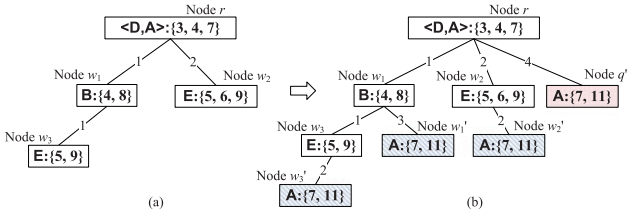


Fig. 3. The result for running Algorithm 6 to the PER-trie $\mathcal{T}_{\langle \mathsf{D},\mathsf{A}\rangle}$ when $i = 4$.

will be directly generated to the node w, and a new valid PER can be derived (Lines 14-17). Otherwise, all descendants of w can be safely pruned according to the Lemma 2 (Line 19). Eventually, it will return a complete PER-trie after the outer loop is finished.

Algorithm 6. MIP-TRIE(PRU)

Input: α: a frequent minimal-occurrence episode
ET_α: the set of end time of each MEO of α
ϵ: the threshold of maximum window size for consequent occurrence
min_freq: minimum frequency threshold generating a valid PER
Output: $\mathcal{T}_\alpha$: the complete PER-trie w.r.t antecedent α

```
1  begin
2    build a root node r of T_α where r.episode ← α and r.tlist ← ET_α
3    for i = 1 to ε do
4    position_List ← {t'|t' = t + i, t ∈ r.tlist}
5    scan S⃗ on each time stamp in position_List and get an event set E' such that the frequency of every event in E' is not less than min_freq
6    foreach e' ∈ E' do
7      q'.event ← e'
8      q'.tlist ← occurrence time of e' in position_List
9      put q' as the child of r
10     foreach non-root node w of T_α except q' do
11       d ← distance between r and w
12       if i > d then
13         tmp_List = {t'|t + i − d, t ∈ w.tlist}
14         S ← tmp_List ∩ q'.tlist
15           if |S| ≥ min_freq then
16             w'.event ← q'.event, w'.tlist ← S
17             put w' as the child of w
18           else
19             /* Pruning with Lemma 2*/
20 end
```

Example 7. The mining process of Algorithm 6 when $i = 4$ is given as Fig. 3. We first collect the event sets E_7, E_8 and E_{11} and expand a child node, namely q', of the root node r. The node q' is shade by dots as shown in Fig. 3b. Next, we separately traverse the non-root nodes w_1, w_2 and w_3 in Fig. 3a and compute the set S in the Line 14 of Algorithm 6. In more detail, for the node w_1, we first get $tmp\_List = \{7, 11\}$ and derive to $S = \{7, 11\}$. Hence, we generate a new node w_1' as a child node of w_1 and discover a valid PER $\langle \mathsf{D}, \mathsf{A}\rangle \xrightarrow{1} (\langle \mathsf{B}, \mathsf{A}\rangle, \langle 3\rangle)$ with a confidence of 0.67. The nodes w_2' and w_3' can be generated in the similar way. The mining results for such iteration is shown as Fig. 3b. In this process, event sets E_7, E_8 and E_{11} are collected only once to count events frequencies in MIP-TRIE(PRU), while in MIP-TRIE(DFS) they are collected four times.

Example 8. The result for performing Algorithm 6 when $i = 5$ is given as Fig. 4. The node $q' = (\mathsf{F}{:}\{9, 12\})$ (shaded by dots) is first constructed, and then we access the node $w_1 = (\mathsf{B}{:}\{4, 8\})$. For w_1, we compute the set S (refer to Line 14 in Algorithm 6) and get $S = \{12\}$. Since its cardinality is lower than min_freq, we do not add new child node for w_1. Furthermore, we can prune all descendants of w_1 from the subsequential traverses, namely node w_4, w_5 and w_7, according to Lemma 2. Finally, after we visit w_2, w_3 and w_6, the only new generated node, i.e., w_2', is shaded by lines in the figure.

In the previous example, 42.86 percent nodes are pruned during the procedure of traversing on the PER-trie. Moreover, event set E_8, E_9 and E_{12} are collected only once in MIP-TRIE(PRU) while they are visited eight times in MIP-TRIE(DFS). Therefore, we can see that MIP-TRIE(PRU) is more efficient as it avoids duplicated scans on the original sequence and meanwhile adopts pruning strategies based on downward closure property.

6.4 Soundness and Completeness

In this section, we prove the soundness and completeness of the proposed MIP-TRIE algorithm. First of all, we show the proposed two algorithms are equivalent in Lemma 3.

Lemma 3. *The MIP-TRIE(DFS) algorithm and MIP-TRIE (PRU) algorithm are equivalent.*

Proof. To prove this lemma, we need to prove that given a non-root node w' on a PER-trie, the set S in the Line 14 of Algorithm 6 is equivalent to the set Q in the Line 8 of function ExtendNode.

Without loss of generality, we suppose $w' = (e'{:}w'.tlist)$ and its distance to the root node $r = (\alpha{:}r.tlist)$ is d_1. And the parent node of w' with $d(w, w') = d_2$ is denoted by w. According to Algorithm 6, we have a child node, $q' = (e'{:}q'.tlist)$, of the root node r whose distance to r is d_1 as well.

First, we prove $Q \subseteq S$. According to the definition of Q, it records the occurrence time stamps of e' that appear d_2 time intervals after the elements in $w.tlist$. We then have $Q \subseteq tmp\_List$ (refer to Line 13 of Algorithm 6) since it stores all time stamps d_2 time intervals after the elements in $w.tlist$. On the other hand, remember that the distance between w' and r is d_1, we have $Q \subseteq q'.tlist$ (refer to Line 8 of Algorithm 6) because $q'.tlist$ stores all occurrence times of e' whose distance to r is d_1. As a consequence, we have $Q \subseteq S = tmp\_List \cap q'.tlist$.

Next, we use apagoge to prove $S \subseteq Q$. Suppose there is an element, denoted as t_j, in S which is not belong to Q. Then we have $t_j \in \{t'|t' = t + d_1, t \in r.tlist\}$, and there is an occurrence of e' on such a t_j. Remember Q stores the occurrence times of e' in $\{t'|t' = t + (d_1 - d_2) + d_2, t \in r.tlist\}$. Hence, t_j must belong in Q which contradicts with our assumption. Thus $S \subseteq Q$. Finally we have $S = Q$. □

Lemma 3 guarantees MIP-TRIE(DFS) and MIP-TRIE (PRU) have the same results. Now we prove the soundness and the completeness of the proposed MIP-TRIE algorithm.

Theorem 1 (Soundness). *Every non-root node in PER-trie generated by MIP-TRIE algorithm is associated with a valid precise-positioning episode rule.*

Proof. Since MIP-TRIE invokes the MIP-TRIE function by delivering $\mathrm{sp}(\alpha) \times min\_conf$ as the minimum frequency

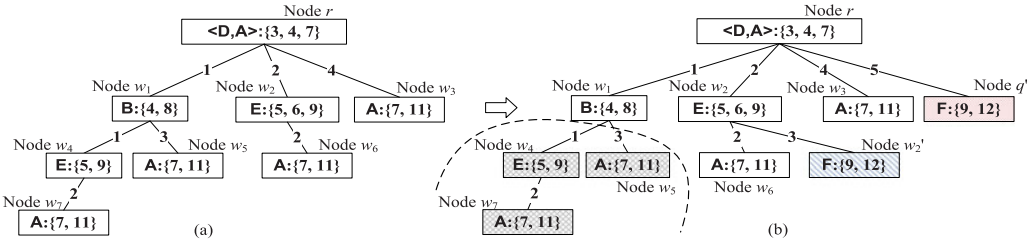


Fig. 4. The result for running Algorithm 6 to PER-trie $\mathcal{T}_{\langle \mathsf{D},\mathsf{A}\rangle}$ when $i = 5$.

threshold for α, each non-root node q in the PER-trie holds $|q.tlist| \geq \mathrm{sp}(\alpha) \times min\_conf$. With Lemma 1, the support of the PER associated with q is $|q.tlist|$, hence such PER must be valid because its confidence is not less than min_conf where min_conf is the user-specified minimum confidence threshold of our problem. The soundness of the algorithm is thus proved. □

Theorem 2. (Completeness). *Given a frequent minimal-occurrence episode α, the complete PER-trie generated by the MIP-TRIE records all valid precise-positioning episode rules whose antecedent is α.*

Proof. Since MIP-TRIE enumerates every possible gap between the antecedent and the consequent, i.e., Δt, from 1 to ϵ by fixed step as 1, it will not leave out any PER candidate based on the definition of PER. Then according to Theorem 1, every node on PER-trie represents a valid PER, the complete PER-trie in which the episode field of the root node is α thus contains all valid precise-positioning episode rules whose antecedent is α. The completeness of the algorithm is thus proved. □

6.5 Complexity Analysis of Building PER-trie

In this section, we analyze both space and time complexity of building a PER-trie. For simplicity, we only discuss the worst case here.

The space complexity of building a PER-trie is decided by the number of nodes of a complete PER-trie. As a result, the DFS and improved methods have the same space complexity. We incorporate them together for discussion. The worst case is that on each timestamp the event sequence has all distinct $|\mathcal{E}|$ events.

We start from the simplest case. Given a PER-trie $\mathcal{T}_\alpha$, we denote its root node as r. Then considering a child node of r, denote as q, whose distance to r is ϵ, i.e., $\mathrm{d}(r, q) = \epsilon$. Since the node q reaches the maximum window size for consequent occurrence, it has no more descendant node. Hence the subtree with q as the root node has only one node. There are all together $|\mathcal{E}|$ nodes, in the worst case, as q which are child nodes of r and their distance to r is ϵ. We denote these nodes by $\mathrm{Node}(\epsilon) = \{q \mid q \text{ is the child node of } r \text{ and } \mathrm{d}(r, q) = \epsilon\}$, and we have $|\mathrm{Node}(\epsilon)| = |\mathcal{E}|$.

Then, for other $\mathrm{Node}(i)$ where $1 \leq i \leq \epsilon - 1$, we have $|\mathrm{Node}(i)| = |\mathcal{E}|(1 + \sum_{j=i+1}^{\epsilon} |\mathrm{Node}(j)|)$. After the reduction, we get $|\mathrm{Node}(i')| = |\mathcal{E}|(|\mathcal{E}| + 1)^{\epsilon - i'}$ for every $i' \in [1, \epsilon]$. Finally the number of nodes on a complete PER-trie will be $1 + \sum_{i=1}^{\epsilon} |\mathrm{Node}(i)| = (|\mathcal{E}| + 1)^{\epsilon}$, and the space complexity is $O(|\mathcal{E}|^{\epsilon})$ in the worst case.

Next we discuss the time complexity. When we build a PER-trie through DFS, we need to scan every event set for each node. Hence its time complexity will be $O((|\mathcal{E}| + 1)^{\epsilon} \cdot |\mathcal{E}| \cdot |r.tlist|)$ where $r.tlist$ is the end time set field of the root node r in such PER-trie. That means we need to check $|r.tlist|$ positions on the sequence, and every event set contains $|\mathcal{E}|$ events in the worst case. Its simplified form leads to $O(|\mathcal{E}|^{\epsilon+1} \cdot |r.tlist|)$.

While for the improved method described in Section 6.3, we only need to scan the original sequence for the children of root node, and perform list merging for other nodes. Specifically, the size of every list will be equal to $|r.tlist|$ in the worst case, and we can perform the merging operations of two lists in a linear time. Consequently, the time complexity of the improved method for build a PER-trie will be $O(\epsilon \cdot |\mathcal{E}| \cdot |r.tlist| + (|\mathcal{E}| + 1)^{\epsilon} \cdot |r.tlist|)$, and its simplified form derives to $O(|\mathcal{E}|^{\epsilon} \cdot |r.tlist|)$.

From the formula of complexity, we can observe both MIP-TRIE(DFS) and MIP-TRIE(PRU) algorithm have the same space complexity while MIP-TRIE (PRU) can save around $|\mathcal{E}|$ times in time for PER-trie construction. On the other hand, the following parameters are more sensitive for the proposed algorithms, namely ϵ, min_sup and min_conf. ϵ is intuitive while min_sup and min_conf jointly determine the real event numbers we need to consider during the process of mining PERs.

7 Applications

In this section, we provide an application scenario of our proposed techniques on a real-world dataset of Beijing traffic condition to demonstrate the existence and the significance of the fixed-gap episodes.

The data we used here is a log sequence of road condition in urban streets of Beijing. The events are collected every fifteen minutes in morning and evening rush hours (7:00 a.m. to 9:00 a.m. and 17:00 p.m. to 19:00 p.m.) ranging from Jan. 4 to Oct. 11, 2005. The content is the traffic status of certain main streets and crossroads. We selected 3 urban areas in Beijing which usually suffer from traffic jams to conduct on the investigation, denoted as Cases 1–3 in Fig. 5. In this figure, the driving directions are labeled with a line from spot "1" to "2", and the included cross-roads are marked by elliptical markers, labeled by A, B, C and D, respectively. We use the cross-road labels, i.e., A, B, C and D, to represent the events of congestions at the corresponding crossroads. As the data has a fixed update rhythm, we index every time stamp of the event sequence within fifteen minutes. In this way, we generate three event sequences for the selected areas.

We mined frequent fixed-gap episodes from these three sequences and obtained fixed-gap episodes on each sequence (shown as congestions). We observed that for each crossroad pair, the most frequent fixed-gap episode has clearly superiority in frequency compared with its variant that consists of the same events but different time spans. To demonstrate that observation, we collected the most frequent

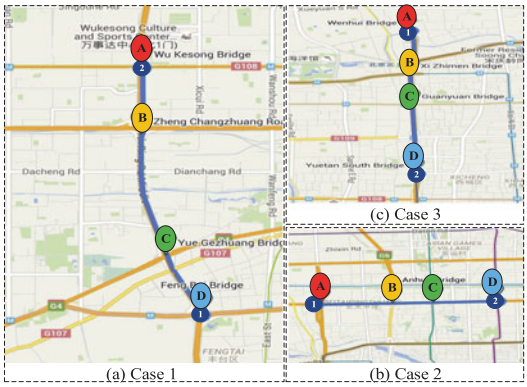


Fig. 5. The road information in traffic condition dataset.

fixed-gap episode for each crossroad pair and compute the difference ratio with its highest-ranking variant as follows:

$$\text{Ratio}(\beta, \beta') = \frac{\text{sp}(\beta) - \text{sp}(\beta')}{\text{sp}(\beta)}, \tag{3}$$

where β is the most frequent fixed-gap episode for a given crossroad pair, and β' is its corresponding variant. We can see that the results on Tables 1, 2, and 3. Further, for the relationship between more crossroads, e.g., A, B and D in Case 1, the fixed-gap episode $(\langle \mathsf{A}, \mathsf{B}, \mathsf{D} \rangle, \langle 1, 1 \rangle)$ ranks the first. Its support outperforms that of followed variations $(\langle \mathsf{A}, \mathsf{B}, \mathsf{D} \rangle, \langle 1, 2 \rangle)$ and $(\langle \mathsf{A}, \mathsf{B}, \mathsf{D} \rangle, \langle 2, 1 \rangle)$ by 61.5 and 76.9 percent, respectively. Similar patterns were also found from the sequences for Cases 2 and 3. These observations are consistent with the assumed chain reaction mechanism in Section 1. The reason is that we can find a specific μ as the frequently time span between any two consecutive events in a fixed-gap episode.

Next we performed a predictive analysis on such data. In particular, we splitted the sequence into two parts by date as training set and test set, respectively. Then we validated whether the most frequent fixed-gap episodes for each crossroad pair in the training set were still ranked highest on the test set. The results showed that the most frequent episodes for each crossroad pair in the training set still ranked first in the test set.

Based on such observations, we believe frequent fixed-gap episode mining identifies the patterns at a finer granularity. With fixed-gap episodes, we can design management strategies. For example, on Case 1 temporally close the entrance to the main road at B fifteen minutes later once we monitor a jam at A, and still keep the entrance at D open for a moment since the traffic status at D is influenced by that of A within about thirty minutes. However, under traditional methods, we will not differentiate $\langle \mathsf{A}, \mathsf{B} \rangle$ and $\langle \mathsf{A}, \mathsf{D} \rangle$ as both of them indicate they will happen frequently within a time interval of thirty minutes.

8 Experiments

In this section, we evaluate both effectiveness and efficiency of the proposed algorithms. We first show the effectiveness of PERs in real data from China stock market. Then we evaluate the efficiency of the proposed algorithms on three widely used benchmark datasets. Experiments on efficiency are performed on a CentOS 6.4 server with 1.87 GHz Intel Xeon E7-4807 CPU and 128 GB memory. All of the algorithms are implemented in Java.

TABLE 1
Frequency Difference Ratio on Case 1

No.	β	β'	Ratio(β, β')
1	$(\langle \mathsf{A}, \mathsf{B} \rangle, \langle 1 \rangle)$	$(\langle \mathsf{A}, \mathsf{B} \rangle, \langle 2 \rangle)$	55.6%
2	$(\langle \mathsf{A}, \mathsf{C} \rangle, \langle 1 \rangle)$	$(\langle \mathsf{A}, \mathsf{C} \rangle, \langle 2 \rangle)$	47.5%
3	$(\langle \mathsf{A}, \mathsf{D} \rangle, \langle 2 \rangle)$	$(\langle \mathsf{A}, \mathsf{D} \rangle, \langle 3 \rangle)$	42.4%
4	$(\langle \mathsf{B}, \mathsf{C} \rangle, \langle 1 \rangle)$	$(\langle \mathsf{B}, \mathsf{C} \rangle, \langle 3 \rangle)$	27.2%
5	$(\langle \mathsf{B}, \mathsf{D} \rangle, \langle 1 \rangle)$	$(\langle \mathsf{B}, \mathsf{D} \rangle, \langle 2 \rangle)$	12.5%
6	$(\langle \mathsf{C}, \mathsf{D} \rangle, \langle 1 \rangle)$	$(\langle \mathsf{C}, \mathsf{D} \rangle, \langle 2 \rangle)$	50%

TABLE 2
Frequency Difference Ratio on Case 2

No.	β	β'	Ratio(β, β')
1	$(\langle \mathsf{B}, \mathsf{A} \rangle, \langle 1 \rangle)$	$(\langle \mathsf{B}, \mathsf{A} \rangle, \langle 2 \rangle)$	36.8%
2	$(\langle \mathsf{C}, \mathsf{A} \rangle, \langle 2 \rangle)$	$(\langle \mathsf{C}, \mathsf{A} \rangle, \langle 3 \rangle)$	47.3%
2	$(\langle \mathsf{D}, \mathsf{A} \rangle, \langle 2 \rangle)$	$(\langle \mathsf{D}, \mathsf{A} \rangle, \langle 3 \rangle)$	38%
4	$(\langle \mathsf{C}, \mathsf{B} \rangle, \langle 1 \rangle)$	$(\langle \mathsf{C}, \mathsf{B} \rangle, \langle 2 \rangle)$	30%
5	$(\langle \mathsf{D}, \mathsf{B} \rangle, \langle 2 \rangle)$	$(\langle \mathsf{D}, \mathsf{B} \rangle, \langle 3 \rangle)$	55.1%
6	$(\langle \mathsf{D}, \mathsf{C} \rangle, \langle 1 \rangle)$	$(\langle \mathsf{D}, \mathsf{C} \rangle, \langle 2 \rangle)$	42.9%

TABLE 3
Frequency Difference Ratio on Case 3

No.	β	β'	Ratio(β, β')
1	$(\langle \mathsf{A}, \mathsf{B} \rangle, \langle 1 \rangle)$	$(\langle \mathsf{A}, \mathsf{B} \rangle, \langle 2 \rangle)$	44.1%
2	$(\langle \mathsf{A}, \mathsf{C} \rangle, \langle 1 \rangle)$	$(\langle \mathsf{A}, \mathsf{C} \rangle, \langle 2 \rangle)$	54.2%
3	$(\langle \mathsf{A}, \mathsf{D} \rangle, \langle 1 \rangle)$	$(\langle \mathsf{A}, \mathsf{D} \rangle, \langle 2 \rangle)$	36%
4	$(\langle \mathsf{B}, \mathsf{C} \rangle, \langle 1 \rangle)$	$(\langle \mathsf{B}, \mathsf{C} \rangle, \langle 2 \rangle)$	58.6%
5	$(\langle \mathsf{B}, \mathsf{D} \rangle, \langle 1 \rangle)$	$(\langle \mathsf{B}, \mathsf{D} \rangle, \langle 2 \rangle)$	26.8%
6	$(\langle \mathsf{C}, \mathsf{D} \rangle, \langle 1 \rangle)$	$(\langle \mathsf{C}, \mathsf{D} \rangle, \langle 2 \rangle)$	47.6%

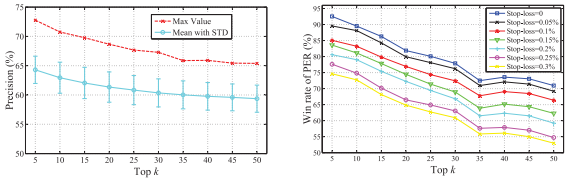
8.1 Effectiveness of PER

We first evaluate the effectiveness of PER on real sequences from China stock market. In particular, we aim at capturing indications of correlations among industry sectors and use them to help the design of time trading strategies. Compared with traditional episode mining methods, the proposed precise-positioning episode rule mining framework could better fulfill the task. In this investigation, we mine valid PERs from the event sequence built from price series of industry sectors in stock market and evaluate their predictive ability in future.

8.1.1 Data Preparation and Experimental Settings

There are all together 29 industry sectors in China stock market based on a predefined taxonomy of stocks by CITIC Securities (a finance corporation in China). A sector usually contains multiple stocks. We treat each sector as a *pseudo stock*. And we take the average daily price change ratio of all the stock in a sector as the daily change ratio of such pseudo stock. Finally there are 29 time series. Then, we discretize the values of change ratio into two categories and generate two kinds of events: UP (if the price increases) and DN (otherwise) for each sector. To simplify the problem, we only consider the relationship between two custom sectors. Thus, we can merge the events from any two different sectors to form an event sequence. We construct 150 groups of related sector pairs to form an experimental dataset, and we perform investigations on such a dataset.

The data we used involves 1,129 trading days ranging from Jan. 1, 2010 to Aug. 29, 2014. We separate it into two parts to generate the training and test sets as follows: The data from the first 4 years (including 967 trading days) are



(a) Precisions of applying top k PER from training set to test set.

(b) The winning rate of PER compared with TDR.

Fig. 6. The effectiveness of PER.

taken as the training set, and the rest (including 162 trading days) are used as the test set. Valid PERs are mined from the training set and are evaluated on the test set. We empirically decided the parameter settings for the mining: min_sup = 145, min_conf = 0.5, $\delta = 3$ and $\epsilon = 5$. As it is difficult to believe a rule telling an industry sector may go up many days (e.g., two weeks) after viewing an antecedent happen. So we set small values for both δ and ϵ.

8.1.2 Results

Predictive Ability of PER. We obtained multiple PERs from every sequence under our settings. We first aim at checking whether the PERs with high confidence on the training data still perform well on the test data.

In more detail, we rank the rules based on their confidence values on the training data. Then, for the top-k rules in the training data we calculate the average precision on the test set. Fig. 6a shows the results. First of all, we can see that both the average and max values tend to decrease as the value of k increases. It indicates that the rules, which perform better in the training set, are likely to perform better in the test set. Also, the average precision on test data exceeds 59.3 percent when $k = 50$. Recall that the baseline methods, which randomly select one result from two choices, only have precisions around 50 percent on these sequences. More importantly, the max precision reaches 65.4 percent when $k = 50$. It indicates that some predictive rules existing in pairs of sectors perform well on the test set. These rules have the potential business impact when considered by time trading strategies over the securities of the corresponding sectors.

In addition, we generate 150 random datasets with the same scale and perform the same mining and evaluation process on these sequences. We found that the predictive ability of PERs in random datasets are stable with the mean of 50 percent. It is another encouraging observation demonstrating that there might be indeed some underlying regularities in real price sequences of stocks in Chinese market.

PER versus Traditional Episode Rules. We next compare PER with traditional episode rules to show the predictive power of the precision timestamp. Specifically, we degrade PER with *single event* in consequent to traditional episode rule (denoted as TDR hereafter) and demonstrate their differences in predictive ability. In order to show the difference between them more clearly, we collected the rules with $\Delta t = 5$ from top 50 rules on the training sets to construct an experimental set. The reason we adopt the rules whose consequent has only one event is because we can easily make them become TDR by directly concealing their Δt between the antecedent and the consequent. Otherwise, the transformation process will be complicated and unfair. In such comparison, we still evaluate the precision of the rules on the test set as the previous study does. We adopt the following strategy for evaluation: Given a TDR, we trade immediately according to the event saying in its consequent after its antecedent appears. In other words, we buy the corresponding pseudo stock if the consequent predicts "UP" and short it otherwise. Then we hold the position until the expected event happens or the maximum window size for consequent occurrence is reached. We can get the precision of a rule by computing the return in the process of the holdings.

For making a fair comparison between PER and TDR, we will close out when the float loss exceeds a threshold during the holdings by a TDR. Here we vary the stop-loss threshold in a small range, i.e., 0 to 0.3 percent, as the daily price change ratio of each industry sector is small. Recall that such ratio is computed by the mean of all stocks' daily price change ratios in the same industry sector, and 0.3 percent is ranking at around 33th percentile among all ranked daily price change ratios of industry sectors. Fig. 6b illustrates the results of average winning rate of PER at top k under different stop-loss thresholds. The winning rate of PER is calculated by the times that PER outperforms TDR divided by the total number of rules in the experimental set. It is clear that PER has a better average precision when this measure is higher than 50 percent. From the figure, we can see that PER outperforms TDR under all the settings, though the winning rate decreases as k increases. We conclude from this comparison that with the constraints by PER, precisions of rule could make valid contribution to real applications.

TABLE 4
Statistical Information on Data Sets

Data set	#Time stamp	#Events	Avg. #Events per Time stamp
Retail	88,162	16,470	10.3
Kosarak	990,002	41,270	8.1
MSNBC	31,790	17	5.3

8.2 Efficiency Evaluation

Next, we evaluate the efficiency of our algorithms on three real data sets, namely, Retail, Kosarak and MSNBC.[3] Among them Retail consists of sales basket data from stores, while Kosarak and MSNBC are datasets of click-stream data from web sites. We followed the pre-processing method in [4] to make the data fit our problem. The detailed statistic information are shown in Table 4.

8.2.1 Experiment Setup

Though there is no prior work of PER mining, we can extend existing frequent episode mining algorithms to generate PER candidates with post-processing and filter the invalid ones by minimum confidence thresholds. Following this idea, we compared the proposed MIP-ENUM, MIP-TRIE(DFS) and MIP-TRIE(PRU) with the state-of-the-art FEM algorithms including the DFS [24] and the MINEPI + [7]. We also explored the UP-Span [4] and the MESELO [9] algorithm, but it ran out of memory under given settings, and hence we did not include it in the results. There are 4 parameters to be tuned in the problem of MIPER. Hence, we perform four groups of experiments to evaluate the effect of each parameter on the performance. In each experiment, we vary one parameter while keeping the other three

3. The datasets are available at http://www.philippe-fournier-viger.com/spmf/index.php?link=datasets.php

TABLE 5
Parameter Settings of Efficiency Validations over Different Datasets

Data set	min_sup	δ	ϵ	min_conf
Retail	6,000	3	5	0.35
Kosarak	140,000	3	5	0.35
MSNBC	10,000	3	5	0.35

parameters fixed. The detailed settings on the fixed parameter values for different datasets are shown as Table 5. For example, for dataset Retail, when the minimum support threshold (min_sup) is fixed in the experiments, its value is set to 6,000. The reason we use different minimum support thresholds for different data sets is that these sequences are different in terms of length and event dense.

8.2.2 Results on Time Efficiency

Fig. 7 show the results on time efficiency with varying parameters for different datasets. The execution times on different datasets with different parameter settings are shown in each sub-figure. From these results, we have the following common observations:

- The extended baselines fail to report results on each setting of parameters because they ran out of memory. The reason is that we have to use much lower support threshold, i.e., $min\_sup \times min\_conf$, to mine frequent episodes and need to record every detailed occurrence timestamp of each frequent episode to help generating PER candidates. As a rule of thumb, more lenient parameter settings will result in longer running times and this extra information would consume a lot of space. This observation further demonstrates the effectiveness of our MIP methods.
- The DFS and MINEPI+ methods have the similar performance. It indicates that post-processing operations for obtaining valid PERs dominate the whole mining progress of these baselines. Additionally, MIP-ENUM can clearly outperform DFS and MINEPI+ though it is a straightforward method to solve the MIPER problem.
- MIP-TRIE algorithms are significantly more efficient than MIP-ENUM. Usually, the performance gain can be as large as two orders of magnitude. For instance, Fig. 7a shows that MIP-TRIE(PRU) achieves more than 400 times performance gain than MIP-ENUM when $min\_sup = 500$ on Retail dataset. To understand the difference between MIP-ENUM and MIP-TRIE in a detailed manner, we look at the following statistics: number of frequent minimal-occurrence episodes (1,034), the number of fixed-gap episodes (31,540), the number of PER candidates (53,080,718) and the number of valid PER (12,746) mined over the Retail in such setting. For other settings by varying min_sup and ϵ, we exhibit the corresponding statistics over the Retail dataset as Tables 6 and 7. Obviously, we can observe MIP-ENUM generates a much larger number of PER candidates from the two tables. As we only need to mine the valid PERs after the occurrences of the frequent minimal-occurrence episodes in MIP-TRIE algorithms, the search space is greatly reduced, which results in impressive time saving.
- MIP-TRIE(PRU) is more efficient than MIP-TRIE (DFS). The reason is that some repetitive sequence scans are reduced by re-using the intermediate results within the established PER-trie. Besides, we can also enhance the pruning power with the help of downward-closure property.

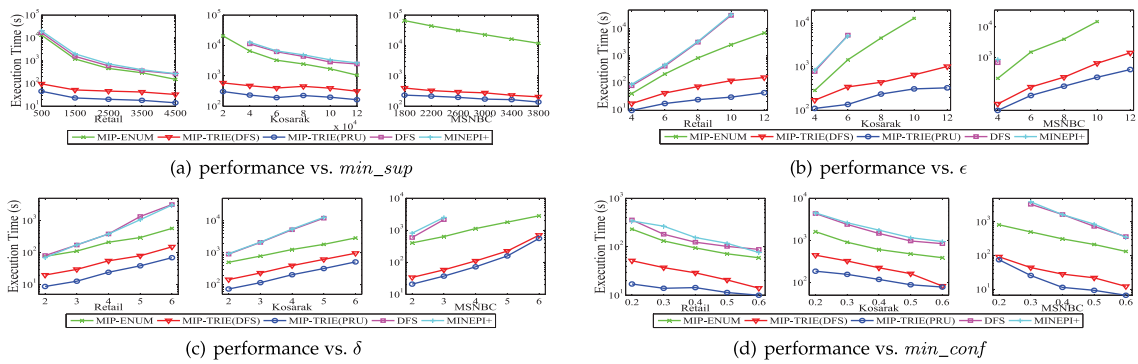
TABLE 6
Output Statistics on Retail Dataset, Varying min_sup

min_sup	#frequent episode	#fix-gaped episode	#PER candidate	#valid PER
4,500	94	2,242	356,918	1,117
3,500	119	3,286	662,830	1,429
2,500	157	4,814	1,335,358	1,849
1,500	325	9,401	5,447,247	3,926
500	1,034	31,540	53,080,718	12,746

TABLE 7
Output Statistics on Retail Dataset, Varying ϵ

ϵ	#frequent episode	#fix-gaped episode	#PER candidate	#valid PER
4	30	332	15,600	256
6	30	1,604	86,850	398
8	30	5,463	342,510	536
10	30	15,210	1,078,260	666
12	30	36,238	2,880,990	793

Fig. 7a shows the execution time of the five methods with different settings of min_sup. We observe that the superiority of MIP-TRIE algorithms becomes more obvious as min_sup decreases. It outperforms the MIP-ENUM by more than two orders of magnitude when min_sup is small. On the other hand, MIP-TRIE(PRU) has a mild advantage compared with MIP-TRIE(DFS) algorithm. Thus, we can conclude that the MIP-ENUM is more sensitive to min_sup. The reason is that



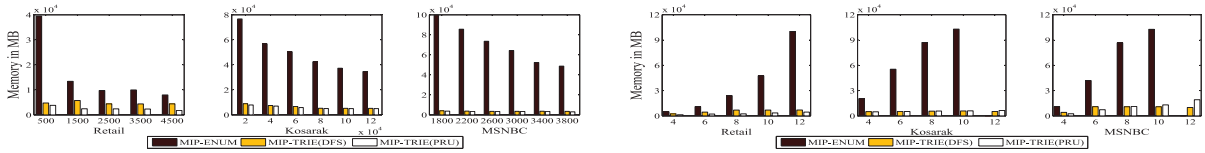
(a) performance vs. min_sup

(b) performance vs. ϵ

(c) performance vs. δ

(d) performance vs. min_conf

Fig. 7. Time efficiency comparisons.



(a) performance vs. min_sup (b) performance vs. ϵ

Fig. 8. Memory consumption comparisons.

when $min\_sup \times min\_conf$ decreases, a much large number of frequent fixed-gap episodes would be considered to form PER candidates, which results in a longer execution time. For the conventional methods, namely DFS and MINEPI+, they cannot finish on some computation intensive cases such as every setting on the MSNBC dataset and $min\_sup = 20{,}000$ on the Kosarak dataset. We check the data statistics of the MSNBC dataset and find it is much denser than the other two datasets, and it requires much more memory for DFS and MINEPI+.

Fig. 7b shows the execution time of the compared methods with different settings of ϵ. In such comparison, MIP-ENUM also ran out of memory on the Kosarak and MSNBC datasets when ϵ was set to 12. We thus only report the results of two methods in this setting. From the figure, the performance of MIP-ENUM degrades seriously as ϵ increases. At the same time, MIP-TRIE methods are less sensitive to ϵ. Here, ϵ controls the scale of the consequent of PERs, namely fixed-gap episodes. Since MIP-TRIE methods only mine the valid rules within some limited area, the impact of ϵ on MIP-TRIE methods becomes less significant. On the other hand, the gap between MIP-TRIE(PRU) and MIP-TRIE (DFS) becomes evident as the parameter increases. The reason is that a bigger ϵ might derive a deeper PER-trie. In this case, the effects of pruning strategies applied in MIP-TRIE(PRU) become more obvious. For the conventional methods, we can observe DFS and MINEPI+ are very sensitive to the parameter of ϵ. They can only report the running time on very few settings, and their running time is significantly greater than the proposed methods. For example, MIP-TRIE(PRU) outperforms DFS and MINEPI+ by 1,092 and 1,184 times, respectively, on the Retail dataset when ϵ is set to 10.

Figs. 7c and 7d show the execution time of the three methods with different settings of δ and min_conf, respectively. We can observe similar trends with the previous evaluations. Besides, since δ has closer correlation with mining antecedents, and min_conf is a filter for valid PERs, there is no doubt that they have impacts on the MIPER problem but not the essential factors dominating the differences between the MIP-ENUM and the MIP-TRIE algorithms. However, both δ and min_conf still have clear effects on DFS and MINEPI+ since these two parameters contribute to determine the maximum window size and the minimum frequency threshold, respectively, when mining frequent episodes.

8.2.3 Results on Memory Consumptions

Next, we investigate the memory consumption evaluations. Here we only focus on comparing the proposed methods in this paper. Since min_sup and ϵ have more significant impacts on the efficiency of PER mining algorithms, we evaluate the memory consumption of the algorithms by varying either parameter min_sup or ϵ but fix the others. Figs. 8a and 8b show the memory consumption of the algorithms on the three datasets under different minimum support threshold min_sup and maximum window size for consequent occurrence threshold ϵ, respectively. Basically, we can observe from the figures that MIP-TRIE algorithms use much less memory than MIP-ENUM, and meanwhile they are more stable in memory using compared with MIP-ENUM. When $\epsilon = 12$, MIP-ENUM even run out of memory and cannot come up with a result on two datasets. For the two algorithms MIP-TRIE(PRU) and MIP-TRIE(DFS), we can observe the former one holds moderate advantages on most of the cases. One different case occurs when varying ϵ on the MSNBC dataset. MIP-TRIE(DFS) uses less memory when ϵ becomes larger. The reason might be that this dataset has shorter sequence length and less events in the sequence. But in most cases, the results show that the best algorithm is MIP-TRIE(PRU) and the worst one is MIP-ENUM in memory consumption.

9 Conclusion and Future Work

In this paper, we formulate the problem of mining precise-positioning episode rules, which is helpful for real-world applications where automatic responses are needed in a timely manner. This is the first attempt to mine episode rules with precise time of the consequent events. We define the consequent of precise-positioning episode rules as fixed-gap episodes, in which the events occur with determined time intervals. Next, we propose one approach based on enumeration as well as two approaches based on a compact trie structure to enhance the pruning power and reduce execution time of the mining process. We demonstrated the effectiveness of our methods in two case studies on finance and transportation. We also conducted an extensive set of experiments to evaluate the efficiency of the proposed methods.

There are also several possible interesting future work of this study. First, investigating the generating mechanism of fixed-gap and studying its correlation between parameter settings. We depicted the possible mechanism for generating fixed-gap episodes and knew σ in Eq. (1) has impact on the number of frequent fixed-gap episodes in sequences. To study the correlation between such mechanism, especially that between σ and frequency parameter selection might be an interesting future work since an estimation of σ may give reasonable suggestions to the setting of frequency thresholds. Second, we could apply MIPER for more complicated type of data, e.g., road network and sequence of stocks with more simultaneous sectors.

Acknowledgments

This work is supported by the National Natural Science Foundation of China (No. 61602438, 91546122, 61573335, 61473274, 61773361, and 61473273), National Key R & D Program of China (No. 2017YFB1002104), and Guangdong Provincial Science and Technology Plan Projects (No. 2015 B010109005).

References

[1] H. Mannila, H. Toivonen, and A. Inkeri Verkamo, "Discovery of frequent episodes in event sequences," *Data Mining Knowl. Discovery*, vol. 1, pp. 259–289, 1997.

[2] A. Ibrahim, S. Sastry, and P. S. Sastry, "Discovering compressing serial episodes from event sequences," *Knowl. Inf. Syst.*, vol. 47, pp. 405–432, 2016.
[3] G. Casas-Garriga, "Discovering unbounded episodes in sequential data," in *Proc. Eur. Conf. Principles Data Mining Knowl. Discovery*, 2003, pp. 83–94.
[4] C.-W. Wu, Y.-F. Lin, S. Y. Philip, and V. S. Tseng, "Mining high utility episodes in complex event sequences," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2013, pp. 536–544.
[5] X. Ao, P. Luo, C. Li, F. Zhuang, Q. He, and Z. Shi, "Discovering and learning sensational episodes of news events," in *Proc. 23rd Int. Conf. World Wide Web*, 2014, pp. 217–218.
[6] N. Tatti and B. Cule, "Mining closed strict episodes," in *Proc. IEEE Int. Conf. Data Mining*, 2010, pp. 501–510.
[7] K.-Y. Huang and C.-H. Chang, "Efficient mining of frequent episodes from complex sequences," *Inf. Syst.*, vol. 33, pp. 96–114, 2008.
[8] A. Ng and A. W.-C. Fu, "Mining frequent episodes for relating financial events and stock trends," in *Proc. Pacific-Asia Conf. Advances Knowl. Discovery Data Mining*, 2003, pp. 27–39.
[9] X. Ao, P. Luo, C. Li, F. Zhuang, and Q. He, "Online frequent episode mining," in *Proc. IEEE Int. Conf. Data Eng.*, 2015, pp. 891–902.
[10] X. Ao, P. Luo, J. Wang, F. Zhuang, and Q. He, "Mining precise-positioning episode rules from event sequences," in *Proc. IEEE Int. Conf. Data Eng.*, 2017, pp. 83–86.
[11] S. Laxman, P. Sastry, and K. Unnikrishnan, "Discovering frequent generalized episodes when events persist for different durations," *IEEE Trans. Knowl. Data Eng.*, vol. 19, no. 9, pp. 1188–1201, Sep. 2007.
[12] S. Laxman, V. Tankasali, and R. W. White, "Stream prediction using a generative model based on frequent episodes in event sequences," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2008, pp. 453–461.
[13] N. Meger and C. Rigotti, "Constraint-based mining of episode rules and optimal window sizes," in *Proc. 8th Eur. Conf. Principles Practice Knowl. Discovery Databases*, 2004, pp. 313–324.
[14] H. Mannila and H. Toivonen, "Discovering generalized episodes using minimal occurrences," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 1996, pp. 146–151.
[15] G. McLachlan and D. Peel, *Finite Mixture Models*. Hoboken, NJ, USA: Wiley, 2004.
[16] Y. Wu, L. Wang, J. Ren, W. Ding, and X. Wu, "Mining sequential patterns with periodic wildcard gaps," *Appl. Intell.*, vol. 41, pp. 99–116, 2014.
[17] C. Li, Q. Yang, J. Wang, and M. Li, "Efficient mining of gap-constrained subsequences and its various applications," *ACM Trans. Knowl. Discovery Data*, 2012, Art. no. 2.
[18] I. Rigoutsos and A. Floratos, "Combinatorial pattern discovery in biological sequences: The TEIRESIAS algorithm," *Bioinf.*, vol. 14, pp. 55–67, 1998.
[19] I. Miliaraki, K. Berberich, R. Gemulla, and S. Zoupanos, "Mind the gap: Large-scale frequent sequence mining," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2013, pp. 797–808.
[20] A. Achar, S. Laxman, and P. S. Sastry, "A unified view of the apriori-based algorithms for frequent episode discovery," *Knowl. Inf. Syst.*, vol. 31, pp. 223–250, 2012.
[21] X. Ma, H. Pang, and K.-L. Tan, "Finding constrained frequent episodes using minimal occurrences," in *Proc. IEEE Int. Conf. Data Mining*, 2004, pp. 471–474.
[22] N. Tatti and B. Cule, "Mining closed episodes with simultaneous events," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2011, pp. 1172–1180.
[23] A. Achar, S. Laxman, and P. Sastry, "A unified view of automata-based algorithms for frequent episode discovery," arXiv:1007.0690, 2010, https://arxiv.org/abs/1007.0690
[24] A. Achar, A. Ibrahim, and P. Sastry, "Pattern-growth based frequent serial episode discovery," *Data Knowl. Eng.*, vol. 87, pp. 91–108, 2013.
[25] L. Wan, L. Chen, and C. Zhang, "Mining dependent frequent serial episodes from uncertain sequence data," in *Proc. IEEE Int. Conf. Data Mining*, 2013, pp. 1211–1216.
[26] A. Achar, S. Laxman, R. Viswanathan, and P. Sastry, "Discovering injective episodes with general partial orders," *Data Mining Knowl. Discovery*, vol. 25, pp. 67–108, 2012.
[27] J. Pei, H. Wang, J. Liu, K. Wang, J. Wang, and P. S. Yu, "Discovering frequent closed partial orders from strings," *IEEE Trans. Knowl. Data Eng.*, vol. 18, no. 11, pp. 1467–1481, Nov. 2006.
[28] N. Meger, C. Leschi, N. Lucas, and C. Rigotti, "Mining episode rules in STULONG dataset," in *Proc. 8th Eur. Conf. Principles Practice Knowl. Discovery Databases*, Sep. 2010, pp. 33–45, https://liris.cnrs.fr/membres?idn=crigotti
[29] P. Fournier-Viger, C.-W. Wu, V. S. Tseng, L. Cao, and R. Nkambou, "Mining partially-ordered sequential rules common to multiple sequences," *IEEE Trans. Knowl. Data Eng.*, vol. 27, no. 8, pp. 2203–2216, Aug. 2015.
[30] Y. F. Lin, C. W. Wu, C. F. Huang, and V. S. Tseng, "Discovering utility-based episode rules in complex event sequences," *Expert Syst. Appl.: Int. J.*, vol. 42, pp. 5303–5314, 2015.
[31] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules in large databases," in *Proc. 20th Int. Conf. Very Large Data Bases*, 1994, pp. 487–499.
[32] J. Han, J. Pei, and Y. Yin, "Mining frequent patterns without candidate generation," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2000, pp. 1–12.

Xiang Ao received the BS degree in computer science from Zhejiang University, in 2010 and the PhD degree in computer science from Institute of Computing Technology, Chinese Academy of Sciences, in 2015. He is an assistant professor in the Institute of Computing Technology, Chinese Academy of Sciences. His research interests include mining patterns from large and complex data, particularly on sequence, graph, and transactions. He has published several papers in some top tier journals and conference proceedings, such as the *IEEE Transactions on Knowledge and Data Engineering*, IEEE ICDE, WWW, etc.

Ping Luo received the PhD degree in computer science from the Institute of Computing Technology, Chinese Academy of Sciences. He is an associate professor in the Institute of Computing Technology, Chinese Academy of Science (CAS). His general area of research is knowledge discovery and machine learning. He has published several papers in some prestigious refereed journals and conference proceedings, such as the *IEEE Transactions on Information Theory*, the *IEEE Transactions on Knowledge and Data Engineering*, ACM SIGKDD, ACM CIKM, and IJCAI. He is a member of the IEEE Computer Society and the ACM.

Jin Wang received the master's degree in computer science from Tsinghua University, in 2015. He is working toward the PhD degree in the Computer Science Department, University of California, Los Angeles. His research interests include text analysis and processing, stream data management, and database system.

Fuzhen Zhuang received the PhD degree in computer science from the Institute of Computing Technology, Chinese Academy of Sciences. He is an associate professor in the Institute of Computing Technology, Chinese Academy of Science (CAS). His research interests include transfer learning, machine learning, data mining, distributed classification and clustering, and natural language processing.

Qing He received the BS degree from Hebei Normal University, in 1985 and the MS degree from Zhengzhou University, in 1987, both in mathematics. He received the PhD degree in fuzzy mathematics and artificial intelligence from Beijing Normal University, in 2000. He is a professor as well as a doctoral tutor in the Institute of Computing Technology, Chinese Academy of Science (CAS), and he is a professor with the University of Chinese Academy of Sciences (UCAS). His interests include data mining, machine learning, classification, and fuzzy clustering. He is a member of the IEEE.