Mining Temporal Patterns in Time Interval-Based Data

Article *in* IEEE Transactions on Knowledge and Data Engineering · December 2015

DOI: 10.1109/TKDE.2015.2454515

3 authors, including:

Yi-Cheng Chen
Tamkang University
37 PUBLICATIONS **313** CITATIONS

Mining Temporal Patterns in Time Interval-Based Data

Yi-Cheng Chen, Wen-Chih Peng, and Suh-Yin Lee

Abstract—Sequential pattern mining is an important subfield in data mining. Recently, applications using time interval-based event data have attracted considerable efforts in discovering patterns from events that persist for some duration. Since the relationship between two intervals is intrinsically complex, how to effectively and efficiently mine interval-based sequences is a challenging issue. In this paper, two novel representations, endpoint representation and endtime representation, are proposed to simplify the processing of complex relationships among event intervals. Based on the proposed representations, three types of interval-based patterns: temporal pattern, occurrence-probabilistic temporal pattern, and duration-probabilistic temporal pattern, are defined. In addition, we develop two novel algorithms, Temporal Pattern Miner (TPMiner) and Probabilistic Temporal Pattern Miner (P-TPMiner), to discover three types of interval-based sequential patterns. We also propose three pruning techniques to further reduce the search space of the mining process. Experimental studies show that both algorithms are able to find three types of patterns efficiently. Furthermore, we apply proposed algorithms to real datasets to demonstrate the effectiveness and validate the practicability of proposed patterns.

Index Terms—Data mining, representation, sequential pattern, temporal pattern, interval-based event

1 Introduction

Sequential pattern mining is an active research topic in data mining because of its widespread applicability. This type of application always considers the order relation and the time issue in our daily lives. Sequential pattern mining mainly deals with extracting positive behaviors that can be used to predict an event based on the activity in the preceding sequence of events. However, finding sequential patterns is a difficult issue since mining may require generating or examining a large number of intermediate subsequence combinations. Most previously proposed algorithms for mining sequential patterns, such as GSP [24], MEMISP [14], PrefixSpan [22], PSP [16], and SPADE [34], focused on discovering frequent time-point based correlations or patterns in a large database.

In various real-world scenarios, some events which intrinsically tend to persist for periods of time rather than being instantaneous occurrences, cannot be treated as "time points." In such cases, the data are typically a sequence of events with both start and finish times. For example, prior studies [3], [6], [9], [10] adopted sensor technology to monitor the electricity usage of all household appliances. Specifically, power meters are deployed to collect appliance usage log data. The times that each appliance is turned on and off can be easily identified. Obviously, such appliance usage log data are interval-based data.

- *Y.-C. Chen is with the Department of Computer Science and Information Engineering, Tamkang University, New Taipei City, Taiwan 25137, ROC. E-mail: ycchen@mail.tku.edu.tw.*
- *W.-C. Peng and S.-Y. Lee are with the Department of Computer Science, National Chiao Tung University, Hsinchu, Taiwan 300, ROC. E-mail: wcpeng@cs.nctu.edu.tw, sylee@csie.nctu.edu.tw.*

Manuscript received 14 Sept. 2012; revised 30 June 2015; accepted 5 July 2015. Date of publication 8 July 2015; date of current version 3 Nov. 2015. Recommended for acceptance by J. Pei.
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.2015.2454515

Much of the existing research mainly focuses on discovering patterns from time point-based event data. These approaches are hampered by the fact that they can only efficiently handle instantaneous events not event intervals. The features of time intervals and time points vary substantially; the pairwise relationship between two time interval-based events is intrinsically complex [2]. This complex relationship is a critical problem in the endeavor to design an efficient and effective time interval-based pattern (or **temporal pattern**) mining algorithm, since it may increase candidate generation and the workload for counting the support of candidate sequences.

In this paper, we intend to discover three types of temporal patterns: the **temporal pattern**, the **occurrence-probabilistic temporal pattern** and the **duration-probabilistic temporal pattern**, for different application domains. For a time interval-based database, the temporal pattern can reveal the correlation among intervals to help users perceive the actual behaviors. With the inclusion of information on the time of the occurrence, the occurrence-probabilistic temporal pattern is able to express the correlation among intervals and indicate the occurrence probability, enabling users to predict future activities. Clearly, intervals of different durations may reveal different knowledge. The duration-probabilistic temporal pattern expresses the correlation among intervals and the distribution of different interval lengths which can give users insights into the underlying meanings of the discovered patterns.

Table 1 gives an example database with four interval sequences. Each sequence consists of several intervals; each interval consists of an event symbol, a starting time, and a finishing time. Fig. 1 gives an example of three types of temporal patterns discovered from the example database in Table 1. Each sequence consists of several intervals; each interval consists of an event symbol, a starting time, and a finishing time. Given a minimum support threshold = 3, we

TABLE 1
An Example Database with Four Interval Sequences

SID	event symbol	start time	finish time	interval duration	event sequence
1	*A*	1	4	3	
	B	3	6	3	
	C	8	12	4	
	D	8	12	4	
2	*B*	3	6	3	
	C	8	12	4	
	D	8	12	4	
3	*C*	6	8	2	
	D	6	8	2	
4	*A*	1	3	2	
	C	6	8	2	
	D	6	8	2	
	E	12	4	2	

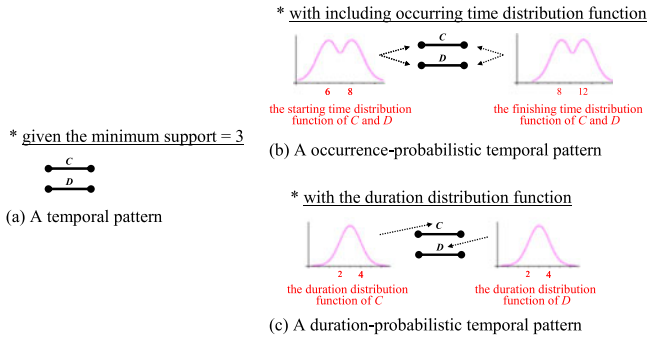


Fig. 1. The example of three types of a temporal pattern discovered from an interval-based database in Table 1.

can observe that (C *equals* D) is a temporal pattern since it appears in all sequences (i.e., the support of this pattern is 4) in the database. With the starting time and finishing time of *C* and *D*, the occurring time distribution function can be easily calculated. We can derive an occurrence-probabilistic temporal pattern based on (C *equals* D) by including the occurring time distribution functions of *C* and *D*. Moreover, with the duration time of *C* and *D*, we can derive the duration distribution function and duration-probability pattern.

Now we use two scenarios to show the applicability of two types of probabilistic temporal patterns. Prior studies [3], [6], [9] have elaborated on mining appliance patterns in behavior analysis. For several appliances, occurring time information of intervals could provide the insights into the discovered temporal patterns. The following application describes how temporal pattern with time information are utilized to detect anomalies in appliance usage.

Application 1 (Occurrence-probabilistic temporal pattern of appliance usage). *Extracting temporal patterns, including time information, from data collected in smart homes can provide valuable appliance usage information for electricity conservation; residents can observe the correlations among appliance usage, including the time of appliance usage (i.e., when they turn an appliance on or off). The probability that an appliance is used can be derived from the appliance usage time information. For example, in [7], knowing the time at which the light and coffee machine have been turned on/off every day within the span of a year, we can derive their usage time distribution, and deduce the* ***probability*** *and the* ***correlation*** *between the light and the coffee machine being switched on/off at a specific time. This information is very useful for several applications, such as detecting abnormalities in appliance use.*

Moreover, the duration of the interval is crucial for mining temporal patterns in several applications. Intervals of different lengths that share the same event symbol may still reveal different information. For example, the different length of duration for using an appliance may indicate different activity in the home. The duration of the usage also varies with specific users. We use the following scenario to describe why duration information can facilitate users' activity recognition.

Application 2 (Duration-probabilistic temporal pattern of appliance usage). *Activity recognition is an important task for establishing an intelligent home management system (HMS) that could help residents' daily living. The temporal pattern with duration information mined from usage databases is essential for recognizing residents' activities. Usually, with the same appliance usage, different activities may have different durations. For example, for the light in the bathroom, the usage duration for brushing teeth is different to that for taking a shower. Since the duration distribution of appliance usage can be derived from the duration information in the usage databases, we could utilize the temporal patterns with duration distributions to increase the accuracy and efficiency of activity recognition.*

To mine temporal patterns from time interval-based data, the major problem of temporal mining tasks is the complex relationships among event intervals [29]. To the best of our knowledge, numerous studies [5], [7], [8], [11], [17], [18], [20], [21], [23], [26], [27], [29], [30], [33] have utilized Allen's 13 temporal relations [2] to describe the relationship between two event intervals. We propose a concept for simplifying the processing of complex relationships in mining temporal patterns. We claim that Allen's 13 temporal relations can be reduced to three relationships: *before*, *equal*, and *after* in our proposed representations for time interval-based data. Based on this concept, two representations, **endpoint representation** and **endtime representation**, are developed to facilitate temporal pattern mining. Endpoint representation utilizes the global information of all endpoint arrangements in a sequence to unambiguously express an interval sequence. By including the time information of intervals, endtime representation could describe an interval sequence losslessly. Based on proposed representations, we propose two algorithms, **Temporal Pattern Miner (TPMiner)** and **Probabilistic Temporal Pattern Miner (P-TPMiner)** to discover three types of temporal patterns. Several pruning strategies are employed to reduce the search space and avoid unpromising processes. Experimental studies on both synthetic and real datasets indicate that the proposed algorithms are both efficient and scalable, and outperform the state-of-the-art algorithms. Finally, the experiments conducted in this study reveal that the proposed algorithms consume a substantially lower amount of memory than previous algorithms.

The remainder of this paper is organized as follows. Sections 2 and 3 present related works and preliminary, respectively. Section 4 provides the proposed presentations. Section 5 describes the pattern mining algorithms. Section 6 details the experiments and the performance study, while Section 7 presents the conclusion.

2 Related Work

Sequential pattern mining is one of the most important research themes in data mining. Recently, numerous studies on this topic have been conducted [1], [4], [12], [14], [16], [22], [24], [28], [34]. These related studies focused on time point-based event data, and include no concept of duration. A number of recent studies have investigated the mining of interval-based events [5], [7], [8], [11], [17], [18], [20], [21], [23], [26], [27], [29], [30], [33]. To the best of our knowledge, most of these studies are based on Allen's temporal relations [2]. However, Allen's temporal relations are binary in nature, and may exhibit problems to describe the relationships among more than two intervals [8], [29]. An appropriate representation is crucial. In this section, we discuss various representations and mining methods. Furthermore, we review several studies on probabilistic sequential pattern mining.

2.1 Representation

Kam and Fu [8] proposed a compact encoding method, named hierarchical representation, to efficiently express the temporal relationships among intervals. However, this method may suffer from two ambiguous problems. First, the same relationships among event intervals can be mapped to different temporal patterns. Second, a temporal pattern can represent different relationships among event intervals. Hoppner [7] proposed an unambiguous representation, relation matrix, which exhaustively lists all binary relationships among event intervals in a pattern. Temporal representation [29] utilizes the relationship among endtime points to express the temporal pattern unambiguously. Patel et al. [21] applied additional counting information to achieve an unambiguous expression called the augmented hierarchical representation. Each Allen describer contains a counter that counts the number of relation occurrences. TSKR [18] considers the noise tolerance and expresses the temporal concepts of coincidence for interval patterns. The pattern represented in TSKR is robust and easily understandable. SIPO [17] uses the partial order among semi-intervals to create an abstraction that can represent many examples with similar properties; however, this representation was based on closed sequential patterns and closed itemsets; hence, the mining is time consuming. Coincidence Representation [5] involves segmenting intervals into disjointed slices to avoid the processing of complex relationships.

In this paper, we modify the concepts of [5], [29], and utilize the arrangement of endpoints to describe an interval sequence directly. Different from [5], [29], the proposed representation includes time information on the occurrence of each endpoint, without expressing the complex relationships among endpoints.

2.2 Mining Algorithms

Several Apriori-like algorithms have been proposed to discover temporal patterns in interval-based data. Villafane et al. [26] proposed a mining method to discover time interval-based sequential patterns by transforming data sequences into containment graphs. Kam and Fu [8] proposed an Apriori-like algorithm to discover temporal patterns based on hierarchy representation. H-DFS [20] transforms an event sequence into id-lists then merges the id-lists iteratively to generate temporal patterns. Laxman et al. [11] extended the original framework of frequent episode discovery in event sequences by incorporating event duration constraints. IEMiner [21] uses several optimization strategies to reduce the search space and remove non-promising candidate temporal sequences. Yoshida et al. [33] efficiently mined sequential patterns by using temporal annotations that quantify the duration between successive symbols.

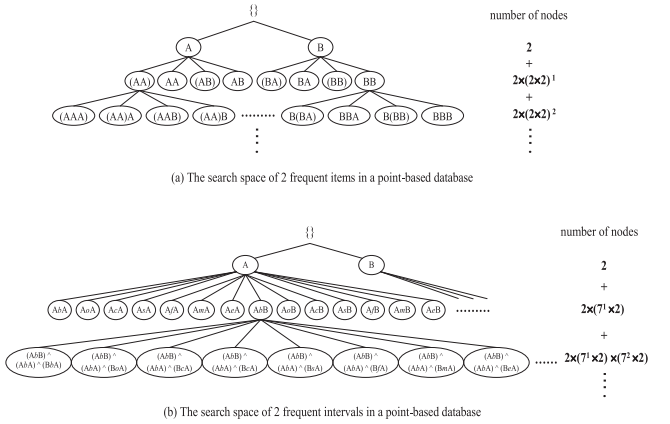


Fig. 2. Examples of the search spaces of two frequent items and two frequent intervals in databases.

In addition, several pattern-growth algorithms have been introduced to increase the efficiency of temporal pattern discovery. ARMADA [27] was proposed to find frequent temporal patterns in a large database. This algorithm requires candidate generation to determine the relationship for growing patterns from local frequent intervals. TPrefixSpan [29] entails generating all possible candidates and then discovering frequent events and scanning the projected databases recursively to discover all temporal patterns. HTPM [30] was developed to mine hybrid temporal pattern from event sequences, which contain both point-based and interval-based events. Sadasivam and Duraiswamy [23] modified the TPrefixSpan algorithm and proposed a refinement method to reduce the number of database scans.

Actually, mining temporal patterns is more arduous than mining point-based patterns because of the complex relationships among intervals. We use search space to indicate the difference between two mining problems. For example, given a point-based database consisting of two frequent items, the first three levels of search space for mining sequential patterns are shown in Fig. 2a. Clearly, if the length of the longest sequential pattern is h, the size of the search space is $2 + 2 \times (2 \times 2)^1 + 2 \times (2 \times 2)^2 + \cdots + 2 \times (2 \times 2)^{h-1} = \mathrm{O}(4^h)$. However, compared to mining sequential pattern, the search space for discovering temporal patterns from an interval-based database containing two frequent intervals is much larger as illustrated in Fig. 2b. With the normalization of interval, the Allen's 13 relationships can be reduced to seven relations [9]. If the length of the longest temporal pattern is h, the size of the search space is $2 + 2 \times (7^1 \times 2) + 2 \times (7^1 \times 2) \times (7^2 \times 2) + \cdots + 2 \times (7^1 \times 2) \times (7^2 \times 2) \times \ldots \times (7^{h-1} \times 2) = \mathrm{O}(7^{h^2} 2^h)$. Obviously, the complex relationships among intervals create a huge search space and complicate the mining processes for temporal patterns.

From the analysis above, we know that the complex relationship is a critical concern when designing an efficient and effective algorithm. Particularly for **Apriori-like algorithms**, in each iteration, the complex relationships among intervals may lead to generating a huge number of candidate sequences and creating a tedious workload of support counting for candidate sequences. Moreover, for **pattern-growth algorithms**, the complex relationships within temporal pattern mining may increase the complexity. Numerous additional low-level processes and operations are required, which is prohibitively expensive in terms of time and space when mining temporal patterns. The motivation of our study is that, if we can simplify the processing of complex relationships among event intervals within temporal patterns, we may design an efficient and effective pattern-growth algorithm.

2.3 Probabilistic Sequential Pattern

Several studies [13], [19], [25], [31], [32], [35], [36] have focused on the problem of mining probabilistic sequential patterns from uncertain data. Muzammal and Raman [19] devise an optimization strategy involving incremental support computation and probabilistic pruning to reduce the cost of sequential pattern mining. Zhao et al. [36] developed U-PrefixSpan to provide solutions for mining probabilistically sequential patterns. Li et al. [13] proposed pattern enumeration algorithms with breadth-first and depth-first strategies for probabilistic frequent spatial-temporal sequential pattern mining. The concept of probabilistic sequential pattern mining has also been used in uncovering surprising periodic patterns in a sequence of events [31], or long sequential patterns in noisy environments [32]. Zakour et al. [35] introduce a sliding window approach to extract frequent interval sequences from discrete temporal sequences. ENFrame [25] allows users to query and process probabilistic data and show the importance of probabilistic sequential pattern mining from uncertain data. Similar to the motivation in [37], in this study, we utilize the time information to derive a probabilistic function for each endpoint of a temporal pattern.

3 Preliminary

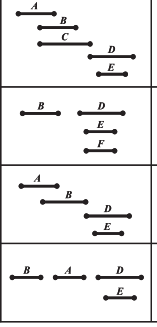
Definition 1 (Event interval). *Let $E = \{e_1, e_2, \ldots, e_k\}$ be the set of event symbols. Without loss of generality, we define a set of uniformly spaced time points based on the natural number N. We say the triplet $(e_i, s_i, f_i) \in E \times N \times N$ is an event interval, where $e_i \in E, s_i, f_i \in N$ and $s_i < f_i$. The two time points s_i, f_i are called endtime points or the occurring times of event e_i, where s_i is the starting time and f_i is the finishing time. The set of all event intervals over E is denoted by $\mathscr{I}$.*

Definition 2 (Interval sequence and maximal property). *An interval sequence is a series of event interval triplets $\langle (e_1, s_1, f_1), (e_2, s_2, f_2), \ldots, (e_n, s_n, f_n)\rangle$, where $s_i \le s_{i+1}$ and $s_i < f_i$. Every interval (e_i, s_i, f_i) must be maximal in a sequence (i.e., no (e_j, s_j, f_j) exists in the sequence such that $e_i = e_j$ and $[s_i, f_i), [s_j, f_j)$ overlap or meet each other]. This assumption is called the maximal property, which is defined as follows:*

$$\forall (e_i, s_i, f_i), (e_j, s_j, f_j), \in \mathscr{I}, i \neq j : (s_i \le s_j) \wedge (f_i \ge s_j) \rightarrow e_i \neq e_j. \quad (1)$$

TABLE 2
Example of a Temporal Database and Proposed Representations

SID	event symbol	start time	finish time
1	A	2	7
1	B	5	10
1	C	5	12
1	D	16	22
1	E	18	20
2	B	1	5
2	D	8	14
2	E	10	13
2	F	10	13
3	A	6	12
3	B	7	14
3	D	14	20
3	E	17	19
4	B	8	16
4	A	18	21
4	D	24	28
4	E	25	27



(endpoint sequence / endtime sequence)
A^+ $(B^+$ $C^+)$ A^- B^- C^- D^+ E^+ E^- D^-
2 5 5 7 10 12 16 18 20 22
B^+ B^- D^+ $(E^+$ $F^+)$ $(E^-$ $F^-)$ D^-
0 5 8 10 10 13 13 14
A^+ B^+ A^- $(B^-$ $D^+)$ E^+ E^- D^-
6 7 12 14 14 17 19 20
B^+ B^- A^+ A^- D^+ E^+ E^- D^-
8 10 13 16 20 21 22 23

Eq. (1) is also called the maximality assumption [8]. The maximal property guarantees that each event interval is maximal in the series. If the maximal property is violated, then both event intervals can be merged and replaced by their union $(e_i, \min(s_i, s_j), \max(f_i, f_j))$.

Definition 3 (Temporal database). *In a database $DB = \{r_1, r_2, \ldots, r_m\}$, each record r_i, where $1 \le i \le m$, consists of a sequence ID and an event interval. DB is called a temporal database.*

Actually, if all records in the database *DB* with the same sequence ID are grouped together and ordered according to nondecreasing starting endpoint, the database can be transformed into a collection of interval sequences, i.e., *DB* can be viewed as a collection of interval sequences. For example, the temporal database in Table 2 consists of 17 event intervals and four interval sequences. Given an interval sequence $Q = \langle (e_1, s_1, f_1), (e_2, s_2, f_2), \ldots, (e_n, s_n, f_n)\rangle$, the set $T_Q = \{s_1, f_1, s_2, f_2, \ldots, s_i, f_i, \ldots, s_n, f_n\}$ is called a **time set** corresponding to Q, where $1 \le i \le n$. If all elements of T_Q are sorted in nondecreasing order, then a sequence $TS_Q = \langle t_1, t_2, \ldots, t_{2n}\rangle$ can be derived, where $t_i \in T_Q, t_i \le t_{i+1}$. TS_Q is called an **endtime sequence** corresponding to Q.

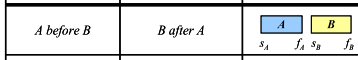
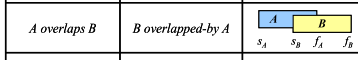
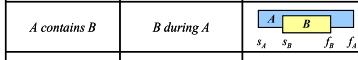
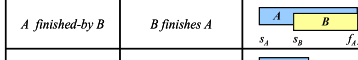
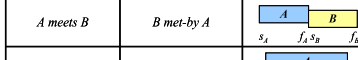
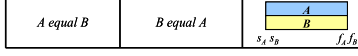
4 Proposed Representation

The time interval-based mining problem is much more arduous than the time point-based mining problem. Since two time intervals may overlap, the relationship among event intervals is more complex than that of the event points. In this paper, two new representations, *endpoint representation* and *endtime representation*, are proposed to effectively express temporal patterns.

From our observation, the complex relationships among event intervals are the major bottleneck for mining temporal patterns. Endpoint representation utilizes the endpoint arrangements to express the relations among intervals in sequence unambiguously. Definition 4 introduces this representation.

Definition 4 (Endpoint sequence). *For an interval sequence $Q = \langle (e_1, s_1, f_1), (e_2, s_2, f_2), \ldots, (e_i, s_i, f_i), \ldots, (e_n, s_n, f_n)\rangle$, where $(e_i, s_i, f_i) \in \mathscr{I}$ and corresponding $TS_Q = \langle t_1, t_2, \ldots,$*

TABLE 3
Proposed Representations of Allen's 13 Temporal Relations

temporal relation	inversed relation	pictorial example (s: starting time, f: finishing time)	endpoint representation	endtime representation
A before B	*B after A*		$A^+A^-B^+B^-$	$\begin{pmatrix} A^+ & A^- & B^+ & B^- \\ s_A & f_A & s_B & f_B \end{pmatrix}$
A overlaps B	*B overlapped-by A*		$A^+B^+A^-B^-$	$\begin{pmatrix} A^+ & B^+ & A^- & B^- \\ s_A & s_B & f_A & f_B \end{pmatrix}$
A contains B	*B during A*		$A^+B^+B^-A^-$	$\begin{pmatrix} A^+ & B^+ & A^- & B^- \\ s_A & s_B & f_A & f_B \end{pmatrix}$
A starts B	*B started-by A*		$(A^+B^+)B^-A^-$	$\begin{pmatrix} (A^+ & B^+) & B^- & A^- \\ s_A & s_B & f_B & f_A \end{pmatrix}$
A finished-by B	*B finishes A*		$A^+B^+(A^-B^-)$	$\begin{pmatrix} A^+ & B^+ & (A^- & B^-) \\ s_A & s_B & f_A & f_B \end{pmatrix}$
A meets B	*B met-by A*		$A^+(A^-B^+)B^-$	$\begin{pmatrix} A^+ & (A^- & B^+) & B^- \\ s_A & f_A & s_B & f_B \end{pmatrix}$
A equal B	*B equal A*		$(A^+B^+)(A^-B^-)$	$\begin{pmatrix} (A^+ & B^+) & (A^- & B^-) \\ s_A & s_B & f_A & f_B \end{pmatrix}$

$t_j, \ldots, t_{2n}\rangle$, *a function* Φ *that maps an interval* (e_i, s_i, f_i) *into two endpoints,* e_i^+ *and* e_i^-, *is defined as follows:*

$$\Phi(t_j,Q) = \begin{cases} e_i^+ \text{if} t_j = s_i, \\ e_i^- \text{if} t_j = f_i, \end{cases} \tag{2}$$

where e_i^+ *and* e_i^- *are the starting point and finishing point of an interval* (e_i, s_i, f_i), *respectively. The endpoints* $e_k^*, \ldots, e_\ell^*$ (* *can be* $^+$ *or* $^-$) *are collected in brackets as a* ***pointset*** *if they occur at the same time in* TS_Q, *denoted as* $e_k^*, \ldots, e_\ell^*$. *An endpoint sequence* ES_Q *of* Q *is denoted as* $\langle p_1, \ldots, p_i, \ldots, p_{2n}\rangle$, *where* p_i *is an endpoint (i.e.,* e_i^**). Moreover, to deal with multiple occurrences of events, an* ***occurrence number*** *is attached to the endpoint to distinguish multiple occurrences of the same event type in an endpoint sequence.*

For example, in Table 2, Sequence 2 is $\langle(B,1,5),(D,8,14),(E,10,13),(F,10,13)\rangle$ and its corresponding endpoint sequence is $\langle B^+B^-D^+(E^+F^+)(E^-F^-)D^-\rangle$. An endpoint sequence ES_Q is also called the **endpoint representation** of Q. The sequence $\langle A_1^+B_1^+(B_1^-D^+)D^-(A_1^-\ B_2^+)B_2^-A_2^+A_2^-\rangle$ is an endpoint sequence with an occurrence number where both events A and B occur twice. With a temporal database DB, by Definition 4, we can transform it into a set of tuples $\langle SID, ES_Q\rangle$ where SID is the sequence ID of each event sequence Q in DB, ES_Q is the endpoint representation of Q. As can be seen in the last column of Table 1, we can transform four event sequences into corresponding endpoint sequences.

Endpoint representation utilizes the interval boundaries and further considers the arrangement layouts to unambiguously express an event sequence. Based on this representation, we can simplify the processing of complex relations among intervals. The relationship between two endpoints is simple, i.e., only *before*, *equal*, or *after*. Allen's 13 temporal relations [2] can effectively be reduced to three relationships. For two different event intervals A and B, the endpoint representation of Allen's 13 temporal relations between A and B is categorized in Table 3. Endpoint representation requires only the endpoint layout in the event sequence; therefore, a temporal database can be efficiently transformed into endpoint representation by simply capturing the endpoint arrangements.

As mentioned above, the time information is also critical for numerous applications. We propose another representation, **endtime representation**, which not only expresses relations among intervals but also reveals the occurrence time. Given an interval sequence $Q = \langle(e_1, s_1, f_1), \ldots, (e_n, s_n, f_n)\rangle$ and the corresponding time sequence $TS_Q = \langle t_1, \ldots, t_i, \ldots, t_{2n}$, by Definition 4, we can derive an endpoint sequence $ES_Q = \langle p_1, \ldots, p_i, \ldots, p_{2n}$ where $p_i = e_j^*$ (* *can be* $^+$ *or* $^-$). The endtime representation of Q is defined as a pair,

$$(ES_Q, TS_Q) = \begin{pmatrix} p_1 & \ldots & p_i & \ldots & p_{2n} \\ t_1 & \ldots & t_i & \ldots & t_{2n} \end{pmatrix}. \tag{3}$$

The endtime representation of Allen's 13 temporal logics between A and B is categorized in Table 3. Note that the occurring time of endpoint p_i in ES_Q is t_i in TS_Q. For example, in Table 2, Sequence 2 is $\langle(B,1,5),(D,8,14),(E,10,13),(F,10,13)\rangle$, and its corresponding time sequence is $\langle 1,5,8,10,10,13,13,14\rangle$. An endpoint sequence $\langle B^+B^-D^+(E^+F^+)(E^-F^-)D^-\rangle$ can be derived using Definition 4. The endtime representation of Sequence 2 is the pair $\begin{pmatrix} B^+ & B^- & D^+ & (E^+ & F^+) & (E^- & F^-) & D^- \\ 1 & 5 & 8 & 10 & 10 & 13 & 13 & 14 \end{pmatrix}$. Actually, the occurrence time is essential for predicting future events. The next occurrence time can occasionally be estimated by analyzing the previous occurrence data. Endtime representation includes the endpoint of the interval and the time information to losslessly express an event sequence.

Definition 5 (Temporal pattern). *Consider two endpoint sequences,* $\alpha = \langle a_1, a_2, \ldots, a_i, \ldots, a_n\rangle$ *and* $\beta = \langle b_1, b_2, \ldots, b_j, \ldots, b_m\rangle$, *where* a_i *and* b_j *are pointsets and* $n \le m$. *The sequence* α *is called a subsequence of* β, *denoted as* $\alpha \subseteq \beta$, *if integers* $1 \le i_1 \le i_2 \le \ldots \le i_n \le m$ *exist such that* $a_1 \subseteq b_{i1}$, $a_2 \subseteq b_{i2}, \ldots, a_n \subseteq b_{in}$. *The sequence* β *is also called a supersequence of* α. *Given a temporal database DB in endpoint representation, a tuple* $\langle sid, Q\rangle$ *is said to contain an endpoint sequence* α *(*Q *is an endpoint sequence), if* α *is a subsequence of* Q. *The support of* α *in DB is the number of tuples in the database containing* α, *i.e.,* $\boldsymbol{support}\ (\boldsymbol{\alpha}) = |\{\langle \boldsymbol{sid}, \boldsymbol{Q}\rangle | (\langle \boldsymbol{sid}, \boldsymbol{Q}\rangle / \in \boldsymbol{DB}) \wedge (\boldsymbol{\alpha} \subseteq \boldsymbol{Q})\}|$. *For a positive integer, min_sup, as the support threshold,* α *is called frequent if support* $(\alpha) \ge$ *min_sup. A frequent endpoint sequence is called a temporal pattern if all endpoints in the sequence appear in pairs, i.e., every starting (finishing) endpoint has a corresponding finishing (starting) endpoint.*

Consider the database in Table 2 with $min\_sup = 2$ as an example. An endpoint sequence $\langle A^+B^+A^-B^-$ is a temporal pattern because it occurs in Sequences 1 and 3, and its *support* $= 2 \ge$ *min_sup*. An endpoint sequence $\langle A^+C^+A^-C^-\rangle$ is not frequent since it occurs only in Sequence 1, and its *support* $= 1 \le$ *min_sup*. Although $\langle A^+B^+A^-\rangle$ is also frequent, it is not a temporal pattern because B^+ has no corresponding finishing endpoint in the sequence.

Definition 6 (Occurrence-probabilistic temporal pattern). *Given a temporal database DB in endtime representation and a threshold, min_sup, the set of frequent sequences, FS, includes all frequent endpoint sequences in DB. A probabilistic temporal pattern OPP is defined as,*

$$OPP = (\alpha, f(\alpha)) = \begin{pmatrix} a_1 & \ldots & a_i & \ldots & a_n \\ f_1 & \ldots & f_i & \ldots & f_n \end{pmatrix}, \tag{4}$$

where $\alpha = \langle a_1, \ldots, a_n\rangle \in FS$ *and* f_i *is the occurrence-probability function of* a_i.

We modify the idea of multivariate kernel density estimation [11] to estimate the occurrence-probability function of each a_i in α. Suppose the time information of a_i in DB is $\{t_{i1}, t_{i2}, \ldots t_{im}\}$, the occurrence-probability function is defined as,

$$f_i(x) = (K(x), h, \{t_{i1}, t_{i2}, \ldots, t_{im}\}) = \frac{1}{mh}\sum_{j=1}^{m} K\left(\frac{x - t_{ij}}{h}\right),$$
$$\text{where } K \text{ is Gaussian Normal kernel, } K(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2}$$
$$\text{and } h = \frac{range(\{t_{i1}, t_{i2}, \ldots, t_{im}\})}{\sqrt{m}}. \tag{5}$$

Definition 7 (Duration-probabilistic temporal pattern). *Given a temporal database DB in endpoint representation and a threshold, min_sup, the set of frequent sequences, FS, includes all frequent endpoint sequences in DB. A duration-probabilistic temporal pattern*

$$DPP = (\alpha, g(\alpha)) = \begin{pmatrix} a_1 & \ldots & a_i & \ldots & a_n \\ g_1 & \ldots & g_i & \ldots & g_n \end{pmatrix},$$
$$\text{where } \alpha = \langle a_1, \ldots, a_n \rangle \in FS \text{ and } g_i \text{ is the duration-probability function of } a_i. \tag{6}$$

Suppose a_i and a_j are the starting and finishing endpoints of an interval with event symbol e, respectively ($a_i = e^+$ and $a_j = e^-$); and in DB, the time information of a_i and a_j are $\{t_{i1}, t_{i2}, \ldots t_{im}\}$ and $\{t_{j1}, t_{j2}, \ldots t_{jm}\}$, respectively. For the starting endpoint $a_i (i.e., e^+)$, we can derive a duration set $\{k_1, k_2, \ldots k_m\}$ where $k_p = t_{jp} - t_{ip}$ and $1 \le p \le m$. The duration-probability function g_i is defined as,

$$g_i(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}, \text{ where } \mu = \frac{\sum_{p=1}^{m} k_p}{m} \text{ and}$$
$$\sigma = \sqrt{\frac{\sum_{p=1}^{m} (k_p - \mu)^2}{m}}. \tag{7}$$

Note that since each interval in a pattern has two endpoints, we only calculate the probability function g_i of the starting endpoint a_i. The probability function g_j of the finishing endpoint a_j actually need not be considered in *TP*. Consider the database in Table 2 as an example. Given *min_sup* = 2, $\langle A^+A^-D^+D^-\rangle$ is a frequent endpoint sequence. The occurrence- and duration-probabilistic temporal pattern with respect to $\langle A^+A^-D^+D^-\rangle$ is $\begin{pmatrix} A^+ & A^- & D^+ & D^- \\ f_{A^+} & f_{A^-} & f_{D^+} & f_{D^-} \end{pmatrix}$ and $\begin{pmatrix} A^+ & A^- & D^+ & D^- \\ g_{A^+} & & g_{D^+} & \end{pmatrix}$, respectively. Here, we illustrate f_{A+} and g_{A+} as further discussion. The time information of A^+ is {2, 6, 13} in the database in Table 2; hence, $f_{A+}(x) = \frac{1}{3h\sqrt{2\pi}}(e^{-\frac{1}{2}(\frac{x-2}{h})^2} + e^{-\frac{1}{2}(\frac{x-6}{h})^2} + e^{-\frac{1}{2}(\frac{x-13}{h})^2})$ with $h = \frac{range(\{2,6,13\})}{\sqrt{3}} = \frac{13-2}{\sqrt{3}} = 6.35$. Based on the set of starting time of interval A in the database, a probabilistic function can be derived to estimate the occurrence probability of A for a specified time query. Moreover, since the time information of A^- is {7, 12, 16}, the duration set of A is {5, 6, 3}. Hence $\mu = \frac{5+6+3}{3} = 4.67$ and $\sigma = \sqrt{\frac{(5-4.67)^2+(6-4.67)^2+(3-4.67)^2}{3}} = 3.35$; $g_{A+}(\mathrm{x}) = \frac{1}{3.35\sqrt{2\pi}} e^{-\frac{(x-4.67)^2}{2\times(3.35)^2}}$. The practicability of the occurrence- and duration-probabilistic temporal pattern is discussed in detail in Section 6. With the above definitions, the problem addressed in this paper is formulated as below,

Problem of mining temporal patterns. *Given a minimum support threshold min_sup, find in temporal database DB a set of temporal patterns, a set of occurrence-probabilistic temporal patterns, and a set of duration-probabilistic temporal patterns.*

5 Proposed Algorithms

In this section, based on the proposed representation (i.e., endpoint representation and endtime representation), we develop two algorithms, Temporal Pattern Miner and Probabilistic Temporal Pattern Miner to efficiently discover three types of temporal patterns.

5.1 TPMiner

In this paper, we borrow the concept of the projection database [22], which uses a divide-and-conquer strategy, to accomplish the discovery of temporal patterns.

Definition 8 (Projected database) *Let α be an endpoint sequence in a database DB in endpoint representation. The α-projected database, denoted as $DB_{|\alpha}$, is the collection of suffixes of endpoint sequences in DB with regard to prefix α. If DB is in endtime representation, each suffix sequence collected in $DB_{|\alpha}$ is a sequence pair comprising the endpoint sequence and the endtime sequence in DB with regard to prefix α.*

Algorithm 1 illustrates the main framework of TPMiner. For a temporal database, event intervals associated with the same sequence ID are grouped into an interval sequence. TPMiner first transforms the temporal database into the endpoint representation, and then scans the database to calculate the count of each endpoint concurrently. TPMiner removes infrequent endpoints below the given support threshold, *min_sup* (Lines 2-4, Algorithm 1). For each frequent starting endpoint *s*, we build the projected database and call TPSpan recursively (Lines 5-7, Algorithm 1) to discover sets of all temporal patterns. Finally, TPMiner outputs all temporal patterns (Line 8, Algorithm 1).

By borrowing the idea of PrefixSpan [22], procedure TPSpan is developed with three search space pruning methods (Algorithm 3). For a prefix α, TPSpan scans its projected database $DB_{|\alpha}$ once to discover all local frequent endpoints and remove infrequent ones (Line 9, Algorithm 2). Frequent endpoint s can be appended to the original prefix to generate a new frequent sequence α' with the length increased by 1. As such, the prefixes are extended (Lines 3-9, Algorithm 2). If all endpoints in a frequent endpoint sequence appear in pairs, i.e., every starting (finishing) endpoint has a corresponding finishing (starting) endpoint, we can output this frequent endpoint sequence (Lines 11-14, Algorithm 1). Finally, we construct the projected database with the frequently extended prefixes and recursively call TPSpan until the prefixes can no longer be extended (line 16, algorithm 1).

Algorithm 1: *TPMiner* (*DB*, *min_sup*)

Input: *DB*: a temporal database, ***min_sup***: the minimum support threshold
Output: *TP*: set of all temporal patterns,

01: $TP \leftarrow \emptyset$
02: transform *DB* into endpoint presentation;
03: find all frequent endpoints and remove infrequent endpoints in *DB*;
04: $FE \leftarrow$ all frequent "**starting endpoints**";
05: **for** each $s \in FE$ **do**
06: construct $DB|_s$;
07: ***TPSpan*** (s, $DB|_s$, *min_sup*, *TP*);
08: **output** *TP*

Procedure *TPSpan* (α, $DB|_\alpha$, *min_sup, TP*)
09: $FE \leftarrow$ ***count_support*** ($DB|_\alpha$, *min_sup*); // **scan-pruning strategy**
10: $FE \leftarrow$ ***point_pruning*** (FE, α); // **point-pruning strategy**
11: **for** each $s \in FE$ **do**
12: append s to α to form α';
13: **if** α' is a temporal pattern **then** // if all endpoints appear in pair in α'
14: $TP \leftarrow TP \cup \alpha'$;
15: $DB|_{\alpha'} \leftarrow$ ***DB_construct*** ($DB|_\alpha$, α'); // **postfix-pruning strategy**
16: ***TPSpan*** (α', $DB|_{\alpha'}$, *min_sup*, *TP*);

Procedure *count_support* ($DB|_\alpha$, *min_sup*)
17: **for** each endpoint sequence $q \in DB|_\alpha$ **do**
18: mark a "*stop_position*" at the first (leftmost) finishing endpoint which has corresponding starting endpoint in α;
19: accumulate the support of every endpoint from the beginning of q to the *stop_position* of q; // **scan-pruning strategy**
20: find every frequent endpoint v in $DB|_\alpha$ such that:
$support(v) \geq min\_sup$ and v can be assembled to the last pointset of α, or $\langle v \rangle$ can be appended to α;
21: $FE \leftarrow$ all frequent endpoints
22: **return** *FE*;

Procedure *point_pruning* (*FE*, α)
23: $temp\_points \leftarrow \emptyset$;
24: **for** each $s \in FE$ **do**
25: **if** s is a "finishing point" **then** // **point-pruning strategy**
26: **if** exist corresponding " starting point" in α **then**
27: $temp\_points \leftarrow temp\_points \cup s$;
28: **if** s is a "starting point" **then**
29: $temp\_points \leftarrow temp\_points \cup s$;
30: **return** *temp_points*;

Procedure *DB_construct* ($DB|_\alpha$, α')
31: $temp\_seq \leftarrow \emptyset$;
32: find all postfix sequences of α' in $DB|_\alpha$ to form $DB|_{\alpha'}$;
33: **for** each postfix sequence $q \in DB|_{\alpha'}$ **do**
34: eliminate the "finishing points" in q which has no corresponding "starting point" in α'; // **postfix-pruning strategy**
35: $temp\_seq \leftarrow temp\_seq \cup q$;
36: **return** *temp_seq*

In consideration of the properties of endpoints, we propose three pruning strategies (scan-pruning, point-pruning, and postfix-pruning) for efficiently and effectively reducing the searching space. First, in order to calculate the support of all endpoints in $DB|_\alpha$, scanning each sequence from the beginning to the end is unnecessary. Instead, we only need to scan from the start of each sequence and stop at the first finishing endpoint which has a corresponding starting endpoint in prefix α. Because the endpoints always appear in pairs in a pattern, a frequent sequence will never become a pattern if it has no chance of obtaining all pairs of endpoints in its subsequent growth. This strategy is called **scan pruning**. As shown in procedure *count_support*, for each sequence in $DB|_\alpha$, we first find the *stop_position*, and then accumulate the supports of all endpoints from the beginning of the sequence to the *stop_position* (Lines 17-19, Algorithm 1). If *stop_position* is not found, we set the *stop-position* as the last endpoint in the sequence. Let the database in Table 2 with $min\_sup = 2$ serve as an example. For a prefix $\langle A^+ \rangle$, the projected database $DB_{|\langle A+ \rangle}$ contains three sequences: $\langle (B^+C^+)\underline{A^-}B^-C^-D^+E^+E^-D^- \rangle$, $\langle B^+\underline{A^-}(B^-D^+)E^+E^-D^- \rangle$, and $\langle \underline{A^-}D^+E^+E^-D^- \rangle$. Obviously, all of the *stop_position*s are A^- in these three sequences. We only need to accumulate the support of endpoints from the beginning of the sequence to A^- (i.e., A^-, B^+ and C^+). Since *support* $(A^+) = 3$, *support* $(B^+) = 2$ and *support* $(C^+) = 1$, we can find two frequent endpoints. All other endpoints need not be accumulated even though they may be frequent. For example, although D^+ is also frequent in Table 1, it is meaningless to grow the frequent sequence $\langle A^+D^+ \rangle$. Since A^- occurs before D^+ in all sequences in the example database, $\langle A^+D^+ \rangle$ will never be a pattern in subsequent mining steps.

Algorithm 2: *P-TPMiner* (*DB*, *min_sup*)

Input: *DB*: a temporal database, ***min_sup***: the minimum support threshold
Output: *OPP*: set of all occurrence-probabilistic temporal patterns,
DPP: set of duration-probabilistic temporal patterns in *DB*

01: $OPP \leftarrow \emptyset$; $DPP \leftarrow \emptyset$;
02: transform *DB* into endtime presentation;
03: find all frequent endpoints and remove infrequent endpoints in *DB*;
04: $FE \leftarrow$ all frequent "**starting endpoints**";
05: **for** each $s \in FE$ **do**
06: find all occurring time information of s in *DB*;
07: $f_s \leftarrow$ calculate the occurrence-probability function of s by Definition 6;
08: $g_s \leftarrow$ calculate the duration-probability function of s by Definition 7;
09: construct $DB|_s$;
10: ***P-TPSpan*** (s, f_s, g_s, $DB|_s$, *min_sup*, *OPP*, *DPP*);
11: **output** *OPP* and *DPP;*

Procedure *P-TPSpan* ($\alpha, f(\alpha), g(\alpha), DB|_\alpha$, *min_sup*, *OPP*, *DPP*)
12: $FE \leftarrow$ ***count_support*** ($DB|_\alpha$, *min_sup*); // **scan-pruning strategy**
13: $FE \leftarrow$ ***point_pruning*** (FE, α); // **point-pruning strategy**
14: **for** each $s \in FE$ **do**
15: append s to α to form α';
16: find all occurring time information of s in $DB|_\alpha$;
17: $f_s \leftarrow$ calculate the occurrence-probability function of s by Definition 6;
18: $f(\alpha') \leftarrow f(\alpha) + f_s$;
19: $g_s \leftarrow$ calculate the duration-probability function of s by Definition 7;
20: $g(\alpha') \leftarrow g(\alpha) + g_s$;
21: **if** α' is a temporal pattern **then** // if all endpoints appear in pair in α'
22: $OPP \leftarrow OPP \cup (\alpha', f(\alpha'))$;
23: $DPP \leftarrow DPP \cup (\alpha', g(\alpha'))$;
24: $DB|_{\alpha'} \leftarrow$ ***DB_construct*** ($DB|_\alpha$, α'); // **postfix-pruning strategy**
25: ***P-TPSpan*** ($\alpha', f(\alpha'), g(\alpha'), DB|_{\alpha'}$, *min_sup*, *OPP*, *DPP*);

Second, the starting and finishing endpoints definitely occur in pairs in a sequence. We only need to project the frequent finishing endpoints which have the corresponding starting endpoints in their prefixes (Lines 25-27, Algorithm 1). For example, for a prefix $\langle D^+ \rangle$, the projected database $DB_{|\langle D+ \rangle}$ contains three sequences: $\langle E^+E^+D^- \rangle$, $\langle E^+E^+D^- \rangle$, and $\langle E^+E^-D^- \rangle$. Although we can find three frequent endpoints: E^+, E^- and D^-, actually, it is not necessary to process E^- since $\langle D^+E^- \rangle$ will never become a pattern in subsequent mining steps. This strategy is called **point pruning**, and can prune off non-qualified patterns before constructing the projected database, as shown in procedure *point_pruning* in Algorithm 1.

Finally, when constructing a projected database, some endpoints in the postfix sequences need not be considered. With respect to a prefix sequence $\langle \alpha \rangle$, a finishing endpoint in a projected postfix sequence is called essential if it has corresponding starting items in $\langle \alpha \rangle$. When constructing the projected database $DB_{|\langle \alpha \rangle}$, only the essential endpoints in the

postfix sequences are collected. All nonessential items are eliminated because they can be ignored during the discovery of temporal patterns. The last pruning method is called **postfix pruning,** and can eliminate useless items when constructing the projected database (lines 32-34, Algorithm 1). The experimental studies indicate that scan-pruning, point-pruning, and postfix-pruning strategies can significantly improve both computation time and memory usage.

5.2 P-TPMiner

Algorithm 2 illustrates the pseudo code of P-TPMiner. Differing from TPMiner, T-TPMiner transforms the database into endtime representation, and then finds frequent endpoints and removes infrequent ones (Lines 2-3, Algorithm 2). For each frequent starting endpoint s, all its time information $\{t_{s1}, t_{s2}, \ldots t_{sm}\}$ in DB is determined and the occurrence- and duration-probability functions f_s and g_s are estimated using Definitions 6 and 7, respectively (Lines 5-9, Algorithm 2). Then, we build the projected database and call P-TPSpan recursively (Line 10, Algorithm 2) to discover sets of all temporal patterns. Finally, P-TPMiner outputs all occurrence- and duration-probabilistic temporal patterns (Line 11, Algorithm 2).

In contrast to TPSpan, P-TPSpan includes the concept of probabilistic function calculation. For a prefix α, P-TPSpan first calls the procedures count_support and point_pruning to find all local frequent endpoints (Lines 12 and 13, Algorithm 2). Frequent endpoint s can be appended to the original prefix to generate a new frequent sequence αprime; (Line 15, Algorithm 2). We also use the time information of s in $DB_{|\alpha}$ to estimate the occurrence- and duration-probability functions f_s and g_s by Definitions 6 and 7, respectively. Then, the prefixes are extended by including f_s and g_s into $f(\alpha')$ and $g(\alpha')$, respectively (Lines 16-20, Algorithm 2). If all endpoints in a frequent endpoint sequence appear in pairs, i.e., every starting (finishing) endpoint has a corresponding finishing (starting) endpoint, we can output this frequent endpoint sequence, including its occurrence and duration probability function, as the occurrence- and duration-probabilistic temporal pattern, respectively (Lines 21-23, Algorithm 2). Finally, we can discover all patterns by constructing the projected database with the frequently extended prefixes and by recursively running P-TPSpan until the prefixes can no longer be extended (lines 24-25, Algorithm 2). Note that, similar to TPSpan, P-TPSpan also utilizes three pruning strategies to effectively reduce the search space (procedures count_support, point_pruning, and DB_construct in Algorithm 1).

6 Experimental Results and Performance Study

TPMiner and P-TPMiner can efficiently discover temporal patterns and two types of probabilistic temporal patterns, respectively. Therefore, to evaluate the performance of TPMiner, we implement four temporal pattern mining algorithms, ARMADA [27], H-DFS [20], IEMiner [21], and TPrefixSpan [29], for comparison. All algorithms were implemented in C++ language and were tested on a workstation with Intel i7–3370 3.4 GHz with 8 GB main memory.

TABLE 4
Parameters of Synthetic Data Generator

Parameters	Description
$\|D\|$	Number of event sequences
$\|C\|$	Average size of event sequences
$\|S\|$	Average size of potentially frequent sequences
N_S	Number of potentially frequent sequences
N	Number of event symbols

A comprehensive performance study has been conducted on both synthetic and real world datasets.

To show the efficiency and practicability of TPMiner and P-TPMiner, we perform five kinds of experiments. First, we compare the running time of TPMiner and the other temporal pattern mining algorithms using synthetic datasets. We also show the number of generated patterns with different thresholds. Second, we analyze the memory usage of TPMiner and other temporal mining algorithms. Furthermore, we show the scalability of the TPMiner algorithms. Third, the performance and memory usage of P-TPMiner are analyzed to show the additional cost for mining probabilistic temporal patterns. Then, we discuss the improvements in the proposed pruning strategies. Finally, TPMiner and P-TPMiner are applied to some real datasets to compare the performance and show the practicability of (probabilistic) temporal pattern mining.

6.1 Data Generation

The synthetic datasets in the experiments are generated using the synthetic generation program proposed by Agrawal and Srikant [1]. The original data generation program was designed to generate time point-based data; therefore, the generator for the temporal pattern mining algorithms requires modifications accordingly. The parameter settings for the temporal data generator are shown in Table 4.

A set of maximal potentially large sequences are first created to generate event sequences. The number of maximal potentially large sequences is N_S. A maximal potentially large sequence is generated by first selecting the size from a Poisson distribution with mean equal to $|S|$. Then, we randomly choose the event interval symbols in the maximal potentially large sequence from N events. Since the time interval in a sequence has duration, the data generator for temporal pattern mining algorithms requires an additional tuning for the experimental data generation. We adopt the modification proposed by Wu et al. [29]. All the duration times of the event intervals are classified into three categories: long, medium and short. The long, medium and short interval events have an average length of 12, 8 and 4 respectively. For each event interval, we first randomly decide its category and then determine its length by drawing a value from a normal distribution.

Finally, we randomly select the temporal relations between consecutive intervals and form a maximal potentially large sequence. Since we adopt normalized temporal patterns [9], the temporal relationships could be chosen from the set {*before*, *meets*, *overlaps*, *is-finished-by*, *contains*, *starts*, *equal*}. After all maximal potentially large sequences are determined, we generate $|D|$ event sequences. Each

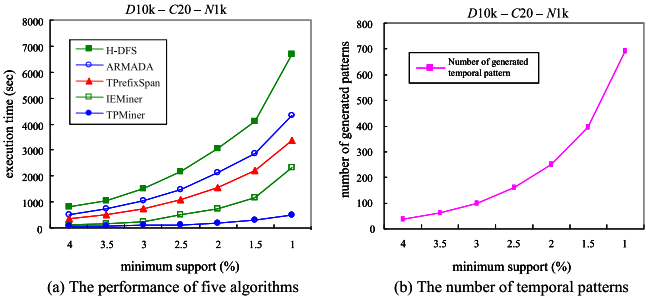


Fig. 3. Runtime performance results on dataset *D*10k – *C*20 – *N*1k.

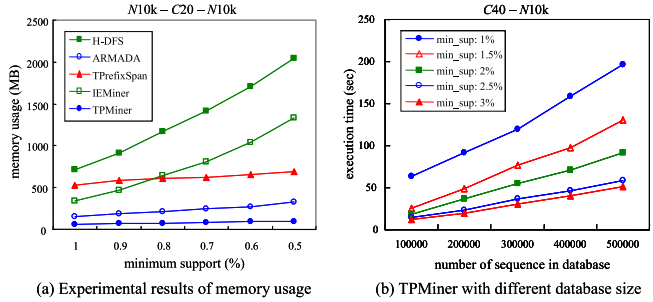


Fig. 5. Experimental results of memory usage and scalability test.

event sequence is generated by first deciding the size from a Poisson distribution with a mean equal to $|C|$. Then, each event sequence is generated by assigning a series of maximal potentially large sequences.

6.2 Runtime Performance for Temporal Pattern Mining on Synthetic Datasets

In all of the following experiments, several parameters are fixed: specifically, $|S| = 4$ and $N_S = 5{,}000$. The other parameters are configured for comparing the temporal pattern mining algorithms. To ensure that the comparison is fair, we only discuss the algorithms generating conventional temporal patterns. We compare the execution times of five algorithms: TPMiner, ARMADA, H-DFS, IEMiner, and TPrefixSpan.

The first experiment is on the dataset *D*10k–*C*20–*N*1k, which contains 10,000 event sequences (the average length of a sequence is 20 and the number of event intervals is 1,000). Figs. 3a and 3b show the runtimes of the five algorithms, and the number of generated temporal patterns at different support thresholds, respectively. The minimum support thresholds vary from 1 to 4 percent. Obviously, when the minimum support value decreases, the processing time required for all algorithms increases. However, the runtimes for ARMADA, H-DFS, IEMiner, and TPrefixSpan increases drastically compared with that of TPMiner. When minimum support is 1 percent, the dataset contains a large number of temporal patterns. From the figure, we can observe that TPMiner is about six times faster than IEMiner, more than 8.7 times faster than ARMADA, about 10 times faster than TPrefixSpan, and more than 15 times faster than H-DFS.

The second experiment is performed on dataset *D*100k–*C*40–*N*10k, which is much larger since it contains 100,000 event sequences (the average length is 40 and the number of event intervals is 10,000). Figs. 4a and 4b show the execution time and the number of generated temporal patterns at different support thresholds, respectively. However, the minimum support thresholds are varied from 0.5 to 1 percent to generate a larger number of frequent patterns from the large dataset. The dataset contains a large number of long temporal patterns when minimum support is reduced to 0.5 percent. We can see that TPMiner is about 6.4 times faster than IEMiner, about 7.8 times faster than ARMADA, more than 9.9 times faster than TPrefixSpan, and more than 14.7 times faster than H-DFS.

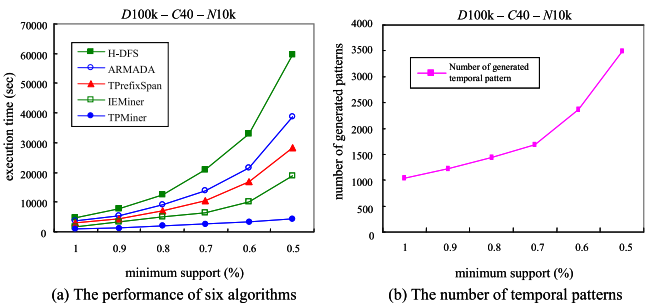


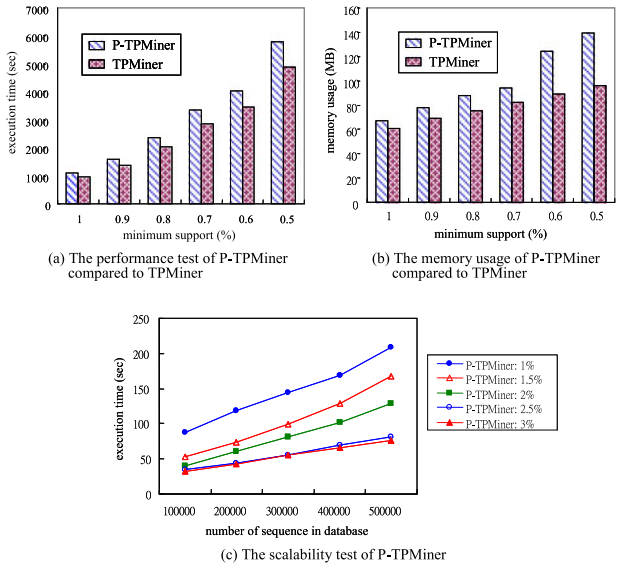
Fig. 4. Runtime performance results on dataset *D*100k – *C*40 – *N*10k.

All of the experiments indicate that, even with extremely low support and a large number of temporal patterns, TPMiner is still efficient and outperforms the state-of-the-art algorithms.

6.3 Memory Usage and Scalability Studies

In this section, we compare the memory usage of five algorithms (TPMiner, ARMADA, H-DFS, IEMiner, and TPrefixSpan) using the synthetic dataset *D*100k–*C*40–*N*10k. Fig. 5a shows that TPMiner is not only more efficient but also more stable in memory usage than the other four algorithms. For example, when the minimum support threshold is reduced to 0.5 percent, TPMiner consumes about 3.4 times less memory than ARMADA, more than 7.1 times less than IEMiner, more than 13 times less than TPrefixSpan, and about 21 times less than H-DFS. This result also explains why, in the previous performance tests when the support threshold becomes extremely low, TPMiner is still efficient and outperforms the state-of-the-art algorithms. Based on our analysis, TPMiner only requires memory space for storing the sequences, plus a set of header tables and pseudo projection tables to construct the projected databases. Although TPrefixSpan is also designed based on PrefixSpan, it still consumes memory space to hold the generated candidate sequences because of the complex relationships among intervals. Both IEMiner and H-DFS require memory space for storing the candidate sequences at each level. When the minimum support threshold drops, the set of candidate sequences grows quickly, resulting in an upsurge of memory consumption. In summary, the performance and memory studies indicate that TPMiner has the best overall performance among the algorithms tested. The memory usage analysis shows the efficient memory consumption of TPMiner and is part of the reason why other algorithms become slow, since the candidate sequences may consume a huge amount of memory.

Second, we discuss the scalability of TPMiner. Fig. 5b shows the results of the scalability test of TPMiner, with the database size growing from 100 to 500K sequences, and with different minimum support threshold settings. Here,

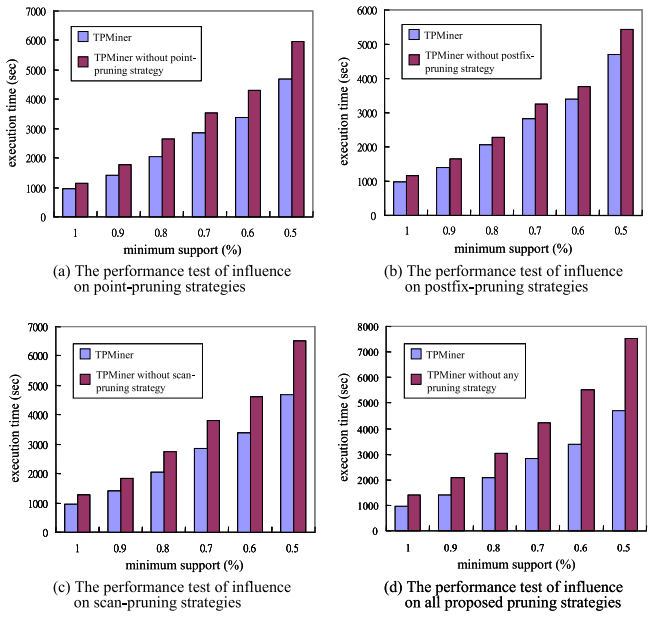


(a) The performance test of P-TPMiner compared to TPMiner

(b) The memory usage of P-TPMiner compared to TPMiner

(c) The scalability test of P-TPMiner

Fig. 6. Experiments of performance, memory usage and scalability of P-TPMiner.



(a) The performance test of influence on point-pruning strategies

(b) The performance test of influence on postfix-pruning strategies

(c) The performance test of influence on scan-pruning strategies

(d) The performance test of influence on all proposed pruning strategies

Fig. 7. Experiments of influence on proposed pruning strategies.

we use the dataset *C*40–*N*10k (the average length of the sequence is 40 and the number of event intervals in the database is 10,000) with varying database size. As the size of the database increases and minimum support decreases, the processing time of TPMiner increases since the number of frequent patterns also increases. As can be seen, TPMiner is linearly scalable with different minimum support thresholds. When the number of generated temporal patterns is large, the runtime of TPMiner still increases linearly with different database sizes.

6.4 Experiments on Mining Probabilistic Temporal Patterns

The main difference between TPMiner and P-TPMiner is including probability function or not. In this section, we discuss the execution time, memory usage, and scalability of P-TPMiner to show the additional cost for mining probabilistic temporal patterns. Fig. 6a gives the comparison of the execution time of P-TPMiner and TPMiner using the synthetic dataset *D*100k–*C*40–*N*10k with different minimum support thresholds. We can observe that mining probabilistic temporal patterns constantly consumes 16.7-19.5 percent more execution time than mining classical temporal patterns. This is because that the complexity of calculating probability function is based on the time information of interval in database. Fig. 6b shows the results of memory usage of P-TPMiner using dataset *D*100k–*C*40–*N*10k with different minimum support thresholds. From the experiments, we can see that mining probabilistic temporal pattern cost a constant workload depending on the number of intervals in a database.

Fig. 6c shows the results of the scalability test of P-TPMiner using the dataset *C*40–*N*10k with varying database size from 100 to 500K, and with different minimum support thresholds. As the size of the database increases and minimum support decreases, the processing time of P-TPMiner increases. Clearly, we can see that P-TPMiner is linearly scalable with different minimum support threshold and with different database size.

6.5 Influence of the Proposed Pruning Strategies

To reflect the enhancement of the proposed pruning methods, we measure TPMiner with three pruning strategies and without a pruning strategy in terms of its time performance. With endpoint representation, TPMiner utilizes three pruning strategies, scan-pruning, point-pruning, and postfix-pruning, to reduce the search space and enhance the performance. The experiment is performed on the dataset *D*100k–*C*40–*N*10k, which contains 100,000 event sequences (the average length of sequence is 40 and the number of events is 10,000).

As shown in Fig. 7a, the point-pruning strategy can improve the performance of TPMiner from 25.3 to 28.1 percent. Because of removing non-qualified slices before database projection, point-pruning can efficiently speedup the execution time. The impact of the postfix-pruning strategy is presented in Fig. 7b. As can be seen from the graph, when TPMiner does not adopt the postfix-pruning strategy, the execution time is about 11 percent slower than TPMiner on average. We can find that the postfix-pruning strategy can improve the performance of TPMiner by effectively eliminating all useless slices for temporal pattern discovery. Fig. 7c shows the improvement of the scan-pruning strategy. We can see that the scan-pruning method constantly ameliorates the performance of TPMiner by about 30 percent. Since we only need to scan from the beginning to the stop position in each sequence in the database, the workload of support counting is significantly reduced. Fig. 7d depicts the influence of the three proposed pruning strategies. We can observe that TPMiner is consistently about 46 percent faster when the pruning strategies are applied than when no pruning strategy is adopted. Consequently, the proposed pruning strategies not only effectively reduce the searching space, but also efficiently improve the performance of TPMiner.

6.6 Real-World Dataset Analysis

In addition to synthetic datasets, we have also performed an experiment on real-world datasets [5], [10], [17] to compare

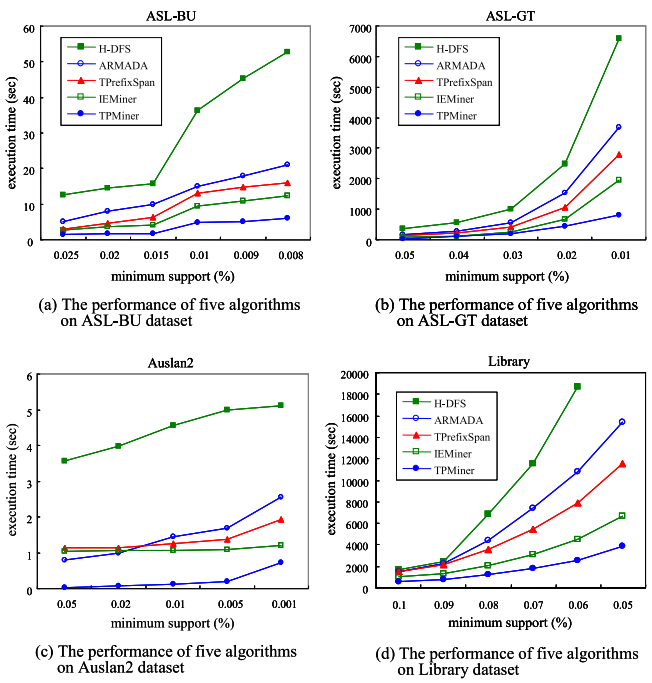
TABLE 5
Five Real Databases [5], [10], [17]

database	intervals	labels	sequences
ASL-BU	18,250	154	441
ASL-GT	89,247	47	3,493
Auslan2	900	12	200
Library	549,071	206,844	28,339
REDD	5,364	90	137

the performance of the algorithms and indicate the applicability of the conventional and probabilistic temporal pattern mining. We use five datasets for evaluation, as shown in Table 5. Notice that only the REDD [10] dataset includes the time information on interval occurrence; the remaining datasets (ASL-BU, ASL-GT, Auslan2 [17] and Library [5]) do not include time information.

We first compare TPMiner with the other mining algorithms: ARMADA, H-DFS, IEMiner and TPrefixSpan for performance testing on the ASL-BU, ASL-GT, Auslan2, and Library datasets to show the efficiency of the proposed algorithm. Fig. 8 shows the execution time of the five algorithms on all real datasets with varying minimum support thresholds. Obviously, all experiments indicate that even with extremely low support, TPMiner is still efficient and outperforms all of the other mining algorithms, particularly with large datasets, such as Library. As can be seen from Fig. 8d, when the minimum support drops down to 0.05 percent, the runtime of TPMiner is about 2.1 times faster than IEMiner, more than 3.6 times faster than ARMADA, about 4.7 times faster than TPrefixSpan, and H-DFS do not terminate.

Then, to demonstrate the practicability of the temporal patterns, we applied the TPMiner algorithm to the book lending dataset [5] to extract the readers' behaviors. This type of information would be more helpful than



(a) The performance of five algorithms on ASL-BU dataset

(b) The performance of five algorithms on ASL-GT dataset

(c) The performance of five algorithms on Auslan2 dataset

(d) The performance of five algorithms on Library dataset

Fig. 8. Experimental results on the ASL-BU, ASL-GT, Auslan2, and Library datasets.

TABLE 6
Some Temporal Patterns Discovered from the Library Dataset [5]

PID	temporal patterns	support
1	"The Know-It-All" overlaps "The Curious Incident of the Dog in the Night-time"; "The Hitchhiker's guide to the galaxy" overlaps "The Restaurant at the End of the Universe"	163 (0.57%)
2	"The Know-It-All" overlaps "The Curious Incident of the Dog in the Night-time"; "Le Cosmicomiche"; "The One Hundred Years of Solitude"	43 (0.15%)
3	"The Homed Man" overlaps "Corazón tan blanco"; "Ut og stjaele hester"	109 (0.38%)
4	"Magic Toyshop"; "Wise Children" overlaps "Nights at the Cirus"; "Burning Your Boats"	97 (0.34%)

conventional sequential patterns for reader recommendation. Table 6 illustrates several temporal patterns (part of the mining results) discovered from the Library dataset. From patterns 1 and 2, we could find an interesting result. Readers who borrow two books, "*The Know-It-All*" and "*The Curious Incident of the Dog in the Night-time*," at different times may borrow different books next. This example shows the practicability of temporal pattern mining. We can perceive that temporal patterns promise a more expressive result from extracting correlations among event data than conventional sequential patterns.

Moreover, in several applications, users may want to know not only the usage correlations among appliances, but also the time information. This is also the motivation for mining probabilistic temporal patterns. To show the practicability of such patterns, we apply P-TPMiner to the appliance usage dataset [10] to discover the residents' usage patterns including usage time and duration estimation. Table 7 illustrates several occurrence-probabilistic temporal patterns (a portion of the mining results) discovered from the REDD dataset. Each endpoint in a pattern includes the occurrence probability function, which describes the usage distribution within a day (24 hours).

TABLE 7
Some Occurrence-Probabilistic Temporal Patterns Discovered from the REDD Dataset [10]

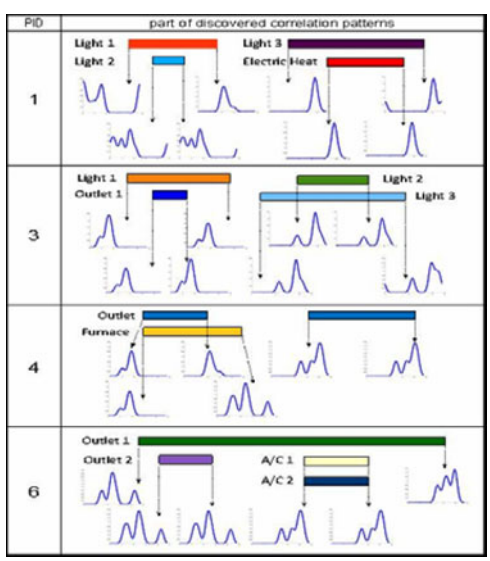
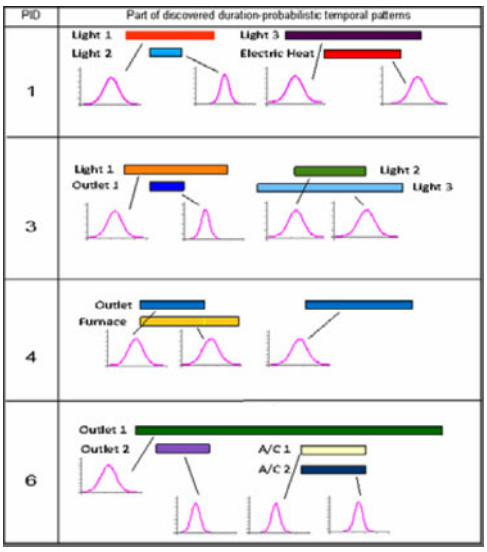


TABLE 8
Some Duration-Probabilistic Temporal Patterns Discovered from the REDD Dataset [10]



By using an occurrence-probabilistic temporal pattern, given a time query, a user can find the relation among intervals and the probability of the occurrence time. Table 8 lists several duration-probabilistic temporal patterns (a portion of the mining results) discovered from the REDD dataset. Each duration probability function describes the duration distribution of an interval. Users can clearly know the probability of interval duration with the duration-probabilistic temporal pattern.

6.7 Discussion of the Quality of the Discovered Patterns from Real-World Datasets

Now we discuss the quality of the discovered patterns from real datasets [5], [10]. We adopt the evaluation of predictive accuracy of the temporal patterns in [29] to show the quality of the discovered temporal patterns. Let R_{train} and R_{test} be the set of discovered patterns from the training set and the set of all patterns appearing at least once in the testing set, respectively. Given a temporal patter p in R_{train}, $Support_{test}(p)$ is defined as the support of p in the testing set. Suppose the length of p is k, $Prefix(p)$ is the prefix $(k-1)$-pattern of p. The predictive accuracy [29] is defined as,

$$PA(p) = \begin{cases} \frac{Support_{test}(p)}{Support_{test}(Prefix(p))}, & if\ p \in R_{test}, \\ 0, & if\ p \notin R_{test}. \end{cases} \quad (8)$$

The value of $PA(p)$ indicates how we can predict the final outcome based on the $(k-1)$-prefix of p. Then, we define the average predictive accuracy and predictive number of R_{Train} as follows,

$$avg\_PA(R_{train}) = \frac{\sum_{p \in R_{train}} PA(p)}{|R_{train}|}, \qquad PN(R_{train}) = |\mathrm{R}_{train} \cap \mathrm{R}_{test}|. \quad (9)$$

In this section, we use $avg\_PA(Rt_{Train})$ (i.e., averaging all $PA(p)$ in R_{train}) and $PN(R_{train})$ (i.e., number of generated patterns from R_{train} appear in R_{test}) to measure the quality of generated patterns. Obviously, the higher values of $avg\_PA(R_{train})$ and $PN(R_{train})$ mean better quality of the discovered patterns in the training set.

TABLE 9
Average Predictive Accuracy and Number of Generated Patterns from the Library Dataset [10]

Library Dataset [10]		
min_sup	$avg\_PA(R_{train})$	$PN(R_{train})$
0.1%	27.4%	254
0.09%	26.3%	336
0.08%	26.9%	393
0.07%	28.2%	418
0.06%	27.2%	556
0.05%	21.1%	1,073

For the Library dataset [5], we take the records of the first two years to construct the training set and use the data of the last year to build the testing set. Table 9 shows the average predictive accuracy with varying *min_sup* from 0.1 to 0.05 percent. We can see that when *min_sup* decreases, $PN(R_{train})$ increases. Clearly, this is because the number of generated patterns also increases. However, this increasing in number of generated patterns does not imply that the average predictive accuracy also increases. We can observe that the $avg\_PA(R_{train})$ is about 26.9 percent with varying *min_sup*.

Since each endpoint in a probabilistic temporal pattern includes a distribution function, when measuring the quality of the pattern, we modify the predictive accuracy in (8) as follows. Given an occurrence-probabilistic temporal pattern q in R_{train},

$$PA(q) = \begin{cases} \frac{Support_{test}(q)}{Support_{test}(Prefix(q))} \times Sim_{test}(q), if\ q \in R_{test} \\ 0, \qquad\qquad if\ q \notin R_{test} \end{cases}, \quad where\ Sim_{test}(q) = \frac{1}{\sum_{each\ ei \in q} KS_{test}(e_i)} \times k, \quad KS_{test}(e_i) = \sup_x |f_{ei,R_{train}}(x) - f_{ei,R_{test}}(x)|, Sup\ is\ the\ supremum\ function, and\ k\ is\ the\ number\ of\ total\ endpoints\ in\ q. \quad (10)$$

When evaluating the predictive accuracy of a probabilistic temporal pattern, we use the Kolmogorov-Smirnov test (KS test) to measure the similarity of two distribution functions. With each endpoint e_i in q, we calculate the maximum distance between the occurrence probability functions f_{ei} in R_{train} (i.e., $f_{ei,Rtrain}$) and f_{ei} in R_{test} (i.e., $f_{ei,Rtest}$). Likewise, if q is a duration-probabilistic temporal pattern, we replace the f_{ei} in (10) with g_e. For each interval e, we calculate the maximum distance between the duration probability functions g_e in R_{train} (i.e., $g_{e,Rtrain}$) and g_e in R_{test} (i.e., $g_{e,Rtest}$).

Then, For the REDD dataset [10], we take the usage records of the first two months to construct the training set and use the data of the last month to build the testing set. Table 10 shows the average predictive accuracy with varying *min_sup* from 0.5 to 0.1 percent. We can see that when *min_sup* decreases, $PN(R_{train})$ increases, but the value of $avg\_PA(Rt_{Train})$ decreases. We could have best avg_PA for occurrence- and duration-probabilistic temporal patterns with *min_sup* = 0.4 and 0.3 percent, respectively.

TABLE 10
Average Predictive Accuracy and Number of Generated Patterns from the REDD Dataset [5]

REDD Dataset [5]			
min_sup	*avg_PA* (R_{train}) (occurrence-function)	*avg_PA* (R_{train}) (duration-function)	$PN(R_{train})$
0.5%	4.7%	13.2%	62
0.4%	9.1%	11.5%	69
0.3%	3.4%	15.2%	88
0.2%	3.1%	11.6%	143
0.1%	0.2%	6.9%	211

7 Conclusion

Mining temporal patterns from time interval-based data is an important subfield in data mining. However, the complex relationships among intervals increase the difficulty of designing an efficient algorithm. In this paper, two new representations, endpoint representation and endtime representation, are proposed to remedy this critical issue. Based on these two representations, we propose three types of patterns: temporal pattern, occurrence-probabilistic temporal pattern, and duration-probabilistic temporal pattern, to describe the correlation among intervals and the probability of the occurring time and duration of each interval. Moreover, the algorithms TPMiner and P-TPMiner, are developed to efficiently discover temporal pattern and probabilistic temporal patterns, respectively. We also propose several pruning techniques to effectively reduce the search space. The experimental studies indicate that TPMiner and P-TPMiner are efficient and practical. Both the runtime and memory usage of TPMiner outperforms those of the state-of-the-art algorithms. In addition, P-TPMiner is able to effectively discover both the occurrence-probabilistic temporal patterns and the duration-probabilistic temporal patterns. There are several interesting issues that could be studied further. Mining closed temporal patterns could use compact results to preserve the same expressive power as temporal pattern mining. Incremental temporal pattern mining is also an important issue which has many real-world applications.

Acknowledgments

Yi-Cheng Chen was supported by the Ministry of Science and Technology of Taiwan, Project No. MOST 104-2221-E-032-037-MY2

References

[1] R. Agrawal and R. Srikant, "Mining sequential patterns," in *Proc. 11th Int. Conf. Data Eng.*, 1995, pp. 3–14.
[2] J. Allen, "Maintaining knowledge about temporal intervals," *Commun. ACM*, vol. 26, no. 11, pp. 832–843, 1983.
[3] F. Chen, J. Dai, B. Wang, S. Sahu, M. Naphade, and C. Lu, "Activity analysis based on low sample rate smart meters," in *Proc. 17th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2011, pp. 240–248.
[4] J. Chen, "An up down directed acyclic graph approach for sequential pattern mining," *IEEE Trans Knowl. Data Eng.*, vol. 22, no. 7, pp. 913–928, Jul. 2010.
[5] Y. Chen, J. Jiang, W. Peng, and S. Lee, "An efficient algorithm for mining time interval-based patterns in large databases," in *Proc. 19th ACM Int. Conf. Inf. Knowl. Manage.*, 2010, pp. 49–58.
[6] Y. Chen, C. Chen, W. Peng, and W. Lee, "Mining correlation patterns among appliances in smart home environment," in *Proc. 18th Pacific-Asia Conf. Knowl. Discovery Data Mining, Adv. Knowl. Discovery Data Mining*, 2014, pp. 210–221 .
[7] F. Hoppner, "Finding informative rules in interval sequences," *Intell. Data Anal.*, vol. 6, no. 3, pp. 237–255, 2002.
[8] P. Kam and W. Fu, "Discovering temporal patterns for interval-based events," in *Proc. 2nd Int. Conf. Data Warehousing Knowl. Discovery*, 2000, vol. 1874, pp. 317–326.
[9] H. Kim, M. Marwah, M. Arlitt, G. Lyon, and J. Han, "Unsupervised disaggregation of low frequency power measurements," in *Proc. 11th SIAM Int. Conf. Data Mining*, 2011, pp. 747–758.
[10] J. Kolter and M. Johnson, "REDD: A public data set for energy disaggregation research," in *Proc. KDD Workshop Data Mining Appl. Sustainability*, 2011, pp. 1–6.
[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] M. Leleu, C. Rigotti, J. Boulicaut, and G. Euvrard, "Constraint-based mining of sequential patterns over datasets with consecutive repetitions," in *Proc. 7th Eur. Conf. Principles Practice Knowl. Discovery Databases*, 2003, pp. 303–314.
[13] Y. Li, J. Bailey, L. Kulik, and J. Pei, "Mining probabilistic frequent spatio-temporal sequential patterns with gap constraints from uncertain databases," in *Proc. 13th Int. Conf. Data Mining*, 2013, pp. 448–457.
[14] M. Lin and S. Lee, "Fast discovery of sequential patterns by memory indexing and database partitioning," *J. Inf. Sci. Eng.*, vol. 21, no. 1, pp. 109–128, 2005.
[15] B. Liu, Y. Yang, G. Webb, and J. Boughton, "A comparative study of bandwidth choice in kernel density estimation for naive Bayesian classification,"in *Proc. 13th Pacific-Asia Conf. Adv. Knowl. Discovery Data Mining*, 2009, pp. 302–313.
[16] F. Masseglia, F. Cathala, and P. Poncelet, "The PSP approach for mining sequential patterns," in *Proc. Eur. Conf. Principles Data Mining Knowl. Discovery*, 1998, vol. 1510, pp. 176–184.
[17] F. Morchen and D. Fradkin, "Robust mining of time intervals with semi-interval partial order patterns," in *Proc. SIAM Int. Conf. Data Mining*, 2010, pp. 315–326.
[18] F. Morchen and A. Ultsch, "Efficient mining of understandable patterns from multivariate interval time series," *Data Mining Knowl. Discovery*, vol. 15, no. 2, pp. 181–215, 2007.
[19] M. Muzammal and R. Raman, "Mining sequential patterns from probabilistic databases," in *Proc. 15th Pacific-Asia Conf. Adv. Knowl. Discovery Data Mining*, 2011, pp. 210–221.
[20] P. Papapetrou, G. Kollios, S. Sclaroff, and D. Gunopulos, "Discovering frequent arrangements of temporal intervals," in *Proc. Int. Conf. Data Mining*, 2005, pp. 354–361.
[21] D. Patel, W. Hsu, and M. Lee, "Mining relationships among interval-based events for classification," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2008, pp. 393–404.
[22] J. Pei, J. Han, B. Mortazavi-Asl, J. Wang, H. Pinto, Q. Chen, U. Dayal, and M. Hsum, "Mining sequential patterns by pattern-growth: The PrefixSpan approach," *IEEE Trans. Knowl. Data Eng.*, vol. 16, no. 10, pp. 1424–1440, Nov. 2004.
[23] R. Sadasivam and K. Duraiswamy, "Efficient method to discover interval-based sequential patterns," *J. Comput. Sci.*, vol. 9, no. 2, pp. 225–234, 2013.
[24] R. Srikant and R. Agrawal, "Mining sequential patterns: Generalizations and performance improvements," in *Proc. 5th Int. Conf. Extended Database Technol.*, 1996, pp. 3–17.
[25] S. van Schaik, D. Olteanu, and R. Fink, "ENFrame: A platform for processing probabilistic data," in *Proc. 17th Int. Conf. Extending Database Technol.*, 2014, pp. 355–366.
[26] R. Villafane, K. Hua, and D. Tran, "Knowledge discovery from series of interval events," *J. Intell. Inf. Syst.*, vol. 15, pp. 71–89, 2000.
[27] E. Winarko and J. F Roddick, "ARMADA – An algorithm for discovering richer relative temporal association rules from interval-based data," *Data Knowl. Eng.*, vol. 63, no. 1, pp. 76–90, 2007.
[28] A. Wong, D. Zhuang, G. Li, and E. Lee, "Discovery of closed patterns and noninduced patterns from sequences," *IEEE Trans. Knowl. Data Eng.*, vol. 24, no. 8, pp. 1408–1421, Aug. 2012.
[29] S. Wu and Y. Chen, "Mining nonambiguous temporal patterns for interval-based events," *IEEE Tran. Knowl. Data Eng.*, vol. 19, no. 6, pp. 742–758, Jun. 2007.

[30] S. Wu and Y. Chen, "Discovering hybrid temporal patterns from sequences consisting of point- and interval-based events," *Data Knowl. Eng.*, vol. 68, no. 11, pp. 1309–1330, 2009.
[31] J. Yang, W. Wang, and P. Yu, "InfoMiner: Mining surprising periodic patterns," *Data Mining Knowl. Discovery*, vol. 9, no. 2, pp. 189–216, 2004.
[32] J. Yang, W. Wang, P. S. Yu, and J. Han, "Mining long sequential patterns in a noisy environment," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2002, pp. 406–417.
[33] M. Yoshida, T. lizuka, H. Shiohara, and M. Ishiguro, "Mining sequential patterns including time intervals," *Proc. SPIE—Data Mining Knowl. Discovery: Theory, Tools, Technol. II*, vol. 4057, pp. 213–220, 2000.
[34] M. Zaki, "SPADE: An efficient algorithm for mining frequent sequences," *Mach. Learn.*, vol. 42, nos. 1/2, pp. 31–60, 2001.
[35] A. Zakour, S. Maabout, M. Mosbah, and M. Sistiaga, "Uncertainty interval temporal sequences extraction," in *Proc. Int. Conf. Inform. Syst. Technol. Manage.*, 2012, pp. 259–270.
[36] Z. Zhao, D. Yan, and W. Ng, "Mining probabilistically frequent sequential patterns in large uncertain databases," in *Proc. 15th Int. Conf. Extending Database Technol.*, 2012, pp. 74–85.

Yi-Cheng Chen received the BS degree in computer science and engineering from Yuan Ze University, Taiwan, in 2000, the MS degree in computer science and engineering from National Taiwan University of Science and Technology, Taiwan, in 2002, and the PhD degree in computer science, National Chiao Tung University, Taiwan, in 2012. He is an assistant professor in the Department of Computer Science and Information Engineering, Tamkang University, New Taipei City, Taiwan. His research interests include data mining, cloud computing, social network analysis, and bioinformatics.

Wen-Chih Peng received the BS and MS degrees from the National Chiao Tung University, Taiwan, in 1995 and 1997, respectively, and the PhD degree in electrical engineering from the National Taiwan University, Taiwan, in 2001. Currently, he is a professor in the Department of Computer Science, National Chiao Tung University, Hsinchu, Taiwan. His research interests include mobile data management and data mining.

Suh-Yin Lee received the BS degree in electrical engineering from National Chiao Tung University, Taiwan, in 1972, and the MS degree in computer science from University of Washington, in 1975, and the PhD degree in computer science from Institute of Electronics, National Chiao Tung University, Taiwan. Currently, she is a professor in the Department of Computer Science, National Chiao Tung University, Hsinchu, Taiwan. Her research interests include content-based retrieval, distributed multmedia information system, mobile computing, and data mining.