Data & Knowledge Engineering 68 (2009) 1112–1127

Contents lists available at ScienceDirect

Data & Knowledge Engineering

journal homepage: www.elsevier.com/locate/datak

On mining multi-time-interval sequential patterns

Ya-Han Hu [a], Tony Cheng-Kui Huang [b,*], Hui-Ru Yang [c], Yen-Liang Chen [c]

[a] Department of Information Management, National Chung Cheng University, 168, University Rd., Min-Hsiung, Chia-Yi, Taiwan, ROC
[b] Department of Business Administration, National Chung Cheng University, 168, University Rd., Min-Hsiung, Chia-Yi, Taiwan, ROC
[c] Department of Information Management, National Central University, Chung-Li 320, Taiwan, ROC

ARTICLE INFO

Article history:
Received 13 April 2008
Received in revised form 18 May 2009
Accepted 18 May 2009
Available online 23 May 2009

Keywords:
Data mining
Knowledge discovery
Sequential pattern
Time-interval
Multi-time-interval

ABSTRACT

Sequential pattern mining is essential in many applications, including computational biology, consumer behavior analysis, web log analysis, etc. Although sequential patterns can tell us what items are frequently to be purchased together and in what order, they cannot provide information about the time span between items for decision support. Previous studies dealing with this problem either set time constraints to restrict the patterns discovered or define time-intervals between two successive items to provide time information. Accordingly, the first approach falls short in providing clear time-interval information while the second cannot discover time-interval information between two non-successive items in a sequential pattern. To provide more time-related knowledge, we define a new variant of time-interval sequential patterns, called multi-time-interval sequential patterns, which can reveal the time-intervals between all pairs of items in a pattern. Accordingly, we develop two efficient algorithms, called the MI-Apriori and MI-PrefixSpan algorithms, to solve this problem. The experimental results show that the MI-PrefixSpan algorithm is faster than the MI-Apriori algorithm, but the MI-Apriori algorithm has better scalability in long sequence data.

1. Introduction

Data mining extracts implicit, previously unknown, and potentially useful information from databases [12]. The discovered information and knowledge are useful for various applications, including market analysis, decision support, fraud detection, and business management. Many approaches have been proposed to extract information, and mining sequential patterns is one of the most important [9,12,16]. Mining sequential patterns was first introduced in the mid-1990s, showing that patterns occur frequently in a sequence database [2]. A typical example of a sequential pattern is when a customer returns to buy a scanner and a microphone after buying a computer.

Although sequential patterns can tell us what items are frequently bought together and in what order, they cannot provide information about the time span between items for further decision support. The former researches addressed this issue by setting time constraints on the patterns; however, they fail to gain the time-interval information between different pairs of items precisely.

To remedy the problems stated above, Chen et al. [6] generalized the mining problem into discovering *time-interval sequential patterns*, which reveal not only the order of items, but also the time-intervals between successive items. For example, we can find patterns like (a) after buying a laser printer, a customer returns to buy a scanner in two months and then a CD burner in six months, or (b) a customer revisits website A within a week and website B after three weeks.

* Corresponding author. Tel.: +886 5 2720411x34319; fax: +886 5 2720564.
E-mail address: bmahck@ccu.edu.tw (T.Cheng-Kui Huang).

0169-023X/$ - see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.datak.2009.05.003

Initially, we assume that the intervals are defined as ti_0: (within 1 day), ti_1: (between 1 day and 8 days), ti_2: (between 9 days and 16 days), and ti_3: (over 17 days). A mapping between letters and items' names at a video rental store is given as: (a, *The Lord of the Rings: The Fellowship of the Ring*), (b, *The Lord of the Rings: The Two Towers*), (c, *The Lord of the Rings: The Return of the King*), and (d, *The Chronicles of Narnia: The Lion, The Witch and The Wardrobe*). An example of a time-interval sequential pattern is $\langle a, ti_1, b, ti_1, c, ti_1, d\rangle$, meaning that a is rented first, then after an interval of ti_1, b, then after an interval of ti_1, c, and finally, after an interval of ti_1, d is rented.

Although the *single-time-interval* [6] approach is an improvement from the above problem, the time-intervals discovered in patterns may not be sufficient for further decision support. Even though it is useful to determine when to take the next step, we are unable to determine when to take the next h steps, where $h \geqslant 2$. For example, although the pattern $\langle a, ti_1, b, ti_1, c, ti_1, d, ti_1, a\rangle$ indicates that the time-interval between items a and b is ti_1, we are unclear to know which time-interval is between items a and d. Without this piece of information, although we observe that a customer rents item a, we have no idea when he or she will come to rent item d. When a customer rents video a, according to the pattern above, the possible period that he or she would rent again the next a is (4,32), which overlaps with intervals from ti_1 to ti_3. In this wide range of time, it is difficult to act in advance. If action is taken too soon or too late, the effects may seriously deteriorate or the action may even become totally useless. Gaining more precise time-interval information, therefore, becomes a significant challenge of mining time-interval sequential patterns.

In this paper, we introduce the *multi-time-interval sequential pattern*, which reveals the time-intervals between all pairs of items in the pattern. For example, we may have a multi-time-interval sequential pattern $\langle a, ti_1, b, (ti_2, ti_1), c, (ti_3, ti_2, ti_1), d\rangle$. Here, the list of intervals (ti_3, ti_2, ti_1) before item d means the intervals between items a, b, and c and item d are ti_3, ti_2 and ti_1, respectively. For the video rental example, we draw the three different kinds of patterns in Fig. 1. As shown in the figure, both the single- and multi-time-interval sequential patterns reveal the order and time-intervals of items, but only the latter can more show the time-intervals between all pairs of items. Two algorithms are developed to discover multi-time-interval sequential patterns from databases. The MI-Apriori algorithm is developed by modifying the I-Apriori algorithm [6], while the MI-PrefixSpan algorithm is developed from the I-PrefixSpan algorithm [6]. The bodies of the I-Apriori algorithm and the I-PrefixSpan algorithm are modified from those of the Apriori algorithm [2] and the PrefixSpan algorithm [28], respectively.

This paper is organized as follows. In Section 2, we review the related work. Then, we formally define the problem in Section 3. The two algorithms are developed in Section 4; Section 4.1 introduces the MI-Apriori algorithm and Section 4.2 the MI-PrefixSpan algorithm. Section 5 provides experiments that evaluate the efficiencies and scalabilities of our algorithms. Finally, conclusions are drawn in Section 6.

2. Related work

The problem of mining sequential patterns was first introduced in the mid-1990s, which consists of discovering the set of subsequences that occur frequently in a sequence database [2]. Since its introduction [2], sequential pattern mining has drawn a great deal of attention from academic researchers as well as practitioners. As a result of its success, extensions have been proposed in various directions, including: (1) other variants of sequential patterns, including maximum patterns [2], similar patterns [1,20,31], cyclic patterns [14,15,23], continuous patterns [34], traversal patterns [10,27], multidimensional patterns [35], hybrid patterns [5], and nonambiguous temporal patterns [33], (2) improved methods for mining sequential

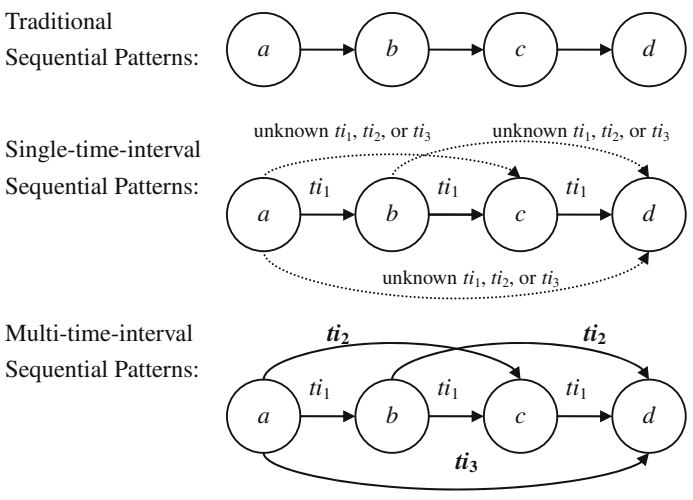


Fig. 1. The three different kinds of patterns from the video rental example.

patterns [17,28,36], (3) constraint-based sequential pattern mining [13,19], (4) mining sequential patterns in web or e-commerce applications [29], (5) parallel [26], incremental methods [8,21,37], and change detection [32] for mining sequential patterns, (6) sequence classification [11], and (7) mining fuzzy sequential patterns [4,7].

Although the discovered sequential patterns can reveal what items are frequently bought together and in what order they appear, they cannot tell us the time span between successive items. Early works addressed this problem in two ways: (a) by setting time constraints, specified by users, to restrict the discovered patterns, and (b) by defining time-intervals between successive items to provide time information. We will describe these two approaches and address their deficiencies in detail.

Srikant and Agrawal [30] extended the definition of the sequential pattern mining problem to handle time constraints, and a new algorithm, called GSP, was proposed. To apply the algorithm, we need to specify the maximum time gap (*max-gap*), the minimum time gap (*min-gap*), and the sliding time window size (*window-size*) in advance. The sequential pattern discovered is thus a sequence of windows, each of which includes a set of items. Items in the same window are bought in the same period. For example, if *max-gap* = 30, *min-gap* = 0, and *window-size* = 7, a sequential pattern $\langle(a,b),(c,d)\rangle$ means that items a and b were purchased within 7 days of each other, items c and d also were purchased within 7 days of each other, and the time-interval between itemsets (a,b) and (c,d) is more than 7 days but within 30 days. Other researchers followed this concept and proposed more efficient algorithms [3,22,24,25]. Joshi et al. [18] presented a universal formulation of sequential patterns, which unifies and generalizes most previous studies in generalized sequential pattern mining. The formulation they proposed can be made identical to most existing formulations that involve time constraints.

The above approach deals with time information by setting time constraints on the patterns. Two problems, however, arise immediately. First, the time-interval information is not explicitly shown in the patterns. Second, all items were restricted by the same time constraints; it does not allow for the possibility that different time-intervals may exist between different pairs of items. Chen et al. [6], therefore, remedied the former to discover time-interval sequential patterns. The new type pattern can reveal not only the order of items, but the time-interval between successive items. Although the first problem is resolved by Chen et al., the second one is still pending to require solving by someone. To the best of our knowledge, no researchers take care of the latter, and we study it in this paper. This study, therefore, can address the void to advance the research of mining time-interval sequential patterns.

3. Problem definition

In this section, we first define the problem of multi-time-interval sequential patterns mining in sequence databases.

A customer's data sequence is an ordered list of itemsets with time stamps. Therefore, a sequence A can be represented as $\langle(a_1,t_1),(a_2,t_2),(a_3,t_3),\ldots,(a_n,t_n)\rangle$, where a_j is an item and t_j is its time stamp for $1 \leqslant j \leqslant n$, and $t_{j-1} \leqslant t_j$ for $2 \leqslant j \leqslant n$. If items occur at the same time in the sequence, they are ordered alphabetically. An item can occur at most once with the same time stamp, but can occur multiple times with the different time stamps.

Let t represent the time-interval between two items, and let T_l be the given constant for $1 \leqslant l \leqslant r-1$, where r represents the number of time-intervals. The time line is divided into r ranges as follows:

- ti_0 denotes the time-interval t satisfying $t = 0$.
- ti_1 denotes the time-interval t satisfying $0 < t \leqslant T_1$.
- ti_j denotes the time-interval t satisfying $T_{j-1} < t \leqslant T_j$ for $1 < j < r-1$.
- ti_r denotes the time-interval t satisfying $T_{r-1} < t \leqslant \infty$.

Let $TI = \{ti_0, ti_1, ti_2, \ldots, ti_{r-1}\}$ be a set of time-intervals, where each distinct object of the set is a complete and non-overlapping partition of the time domain, and let the inequality $ti_i \leqslant ti_j$ be true if $i \leqslant j$. Then, the multi-time-interval sequences can be defined as follows:

Definition 1. Let $I = \{i_1, i_2, \ldots, i_m\}$ be the set of items and $TI = \{ti_0, ti_1, ti_2, \ldots, ti_{r-1}\}$ be the set of time-intervals. A sequence $B = \langle b_1, \&_2, b_2, \&_3, \ldots, b_{s-1}, \&_s, b_s\rangle$ is a *multi-time-interval sequence* if:

1. $b_i \in I$ for $1 \leqslant i \leqslant s$.
2. $\&_i$ is a list of time-intervals, where its size is $i-1$ for $2 \leqslant i \leqslant s$. We can represent $\&_i$ as $(TR_{b_1b_i}, TR_{b_2b_i}, \ldots, TR_{b_{i-1}b_i})$, where $TR_{b_wb_i} \in TI$ denotes the time-interval between b_w and b_i for $1 \leqslant w \leqslant i-1$.

For ease of reference, we define an index system $\&_i(m,n) = (TR_{b_mb_i}, TR_{b_{m+1}b_i}, \ldots, TR_{b_nb_i})$ for $1 \leqslant m \leqslant n \leqslant i-1$. For example, in a multi-time-interval sequence $B = \langle b_1, (ti_1), b_2, (ti_2, ti_1), b_3, (ti_2, ti_1, ti_1), b_4\rangle$ we have the following multi-time-intervals:

(1) $\&_2(1,1) = (ti_1) = (TR_{b_1b_2})$ means that a time-interval between b_1 and b_2 is ti_1;
(2) $\&_3(1,2) = (ti_2, ti_1) = (TR_{b_1b_3}, TR_{b_2b_3})$ means that there are two time-intervals; one between b_1 and b_3 is ti_2 and the other between b_2 and b_3 is ti_1;
(3) $\&_4(1,3) = (ti_2, ti_1, ti_1) = (TR_{b_1b_4}, TR_{b_2b_4}, TR_{b_3b_4})$ means that there are three time-intervals; one between b_1 and b_4 is ti_2, another between b_2 and b_4 is ti_1, and the other between b_3 and b_4 is ti_1.

We also can acquire other multi-time-intervals, such as: $\&_4(2,3) = (ti_1, ti_1) = (TR_{b_2b_4}, TR_{b_3b_4})$ and $\&_3(1,1) = (ti_2) = (TR_{b_1b_3})$ by assigning different m and n. A multi-time-interval sequence satisfies the following property:

Property 1 (*Descending property*). *The time-intervals in* $\&_i$ *must satisfy* $TR_{b_ub_i} \geqslant TR_{b_vb_i}$ *for* $1 \leqslant u < v \leqslant i$.

In the example above, $\&_4 = (ti_2, ti_1, ti_1) = (TR_{b_1b_4}, TR_{b_2b_4}, TR_{b_3b_4})$ and the time-interval between b_1 and b_4 cannot be shorter than the time-interval between b_2 and b_4, and so on. Therefore, it satisfies $TR_{b_1b_4} \geqslant TR_{b_2b_4} \geqslant TR_{b_3b_4}$, i.e. $ti_2 \geqslant ti_1 \geqslant ti_1$.

Definition 2. Let $A = \langle (a_1,t_1),(a_2,t_2),(a_3,t_3),\ldots,(a_n,t_n)\rangle$ be a data sequence and $B = \langle b_1,\&_2,b_2,\&_3,\ldots,b_{s-1},\&_s,b_s\rangle$ be a multi-time-interval sequence. B is *contained* in A if there are integers $1 \leqslant j_1 < j_2 < \cdots < j_s \leqslant n$ in A satisfying the following:

1. $b_1 = a_{j_1}, b_2 = a_{j_2}, \ldots, b_s = a_{j_s}$.
2. $t_{j_i} - t_{j_k}$ is in the time interval of $\&_i(k,k)$ for $2 \leqslant i \leqslant s$ and $1 \leqslant k \leqslant i-1$.

For example, the multi-time-interval sequence $\langle a,(ti_1),b,(ti_2,ti_1),c,(ti_2,ti_1,ti_1),d\rangle$ is contained in data sequence $A = \langle (a,1),(b,3),(c,5),(d,6),(e,8)\rangle$, where ti_0: $t = 0$, ti_1: $0 < t \leqslant 3$, ti_2: $3 < t \leqslant 6$, ti_3: $6 < t \leqslant \infty$.

Definition 3. A multi-time-interval sequence $C = \langle c_1,\&'_2,c_2,\&'_3,\ldots,c_{j-1},\&'_j,c_j\rangle$ is a *subsequence* of multi-time-interval sequence $B = \langle b_1,\&_2,b_2,\&_3,\ldots,b_{s-1},\&_s,b_s\rangle$ if we can find indices $1 \leqslant i_1 < i_2 < \cdots < i_j \leqslant s$ in B satisfying the following:

1. $c_1 = b_{i_1}, c_2 = b_{i_2}, \ldots, c_j = b_{i_j}$.
2. $\&'_m = (TR_{b_{i_1}b_{i_m}}, TR_{b_{i_2}b_{i_m}}, \ldots, TR_{b_{i_{m-1}}b_{i_m}})$ for $2 \leqslant m \leqslant j$.

For example, a multi-time-interval sequence $C = \langle a,(ti_1),b,(ti_2,ti_1),d\rangle$ is a *subsequence* of multi-time-interval sequence $B = \langle a,(ti_1),b,(ti_2,ti_1),c,(ti_2,ti_1,ti_1),d\rangle$ because:

1. $b_{i_1} = a, b_{i_2} = b, b_{i_3} = d$, where $i_1 = 1$, $i_2 = 2$ and $i_3 = 4$.
2. $\&'_2 = (ti_1) = (TR_{ab})$ and $\&'_3 = (ti_2, ti_1) = (TR_{ad}, TR_{bd})$.

The total number of items in a multi-time-interval sequence is the *length* of the sequence. A multi-time-interval sequence C is a *k-subsequence* of B when the length of C is k and C is a subsequence of multi-time-interval sequence B. For example, consider a multi-time-interval sequence B: $\langle b,(ti_1),e,(ti_3,ti_2),c\rangle$, where $k = 3$. All of B's subsequences are $\{b,e,c,\langle b,(ti_1),e\rangle, \langle e,(ti_2),c\rangle,\langle b,(ti_3),c\rangle,\langle b,(ti_1),e,(ti_3,ti_2),c\rangle\}$. B's 1-subsequences are $\{b,e,c\}$, B's 2-subsequences are $\{\langle b,(ti_1),e\rangle,\langle e,(ti_2),c\rangle, \langle b,(ti_3),c\rangle\}$, and B's 3-subsequences are $\{\langle b,(ti_1),e,(ti_3,ti_2),c\rangle\}$.

A data sequence is represented by (sid,s), where sid is the identifier and s is a sequence. A sequential database S is formed from a set of data sequences (sid,s). For a given multi-time-interval sequence α, its support in database S is defined as follows:

$$support(\alpha) = \frac{\{(sid,s)|(sid,s) \in S \wedge \alpha \text{ is contained in } s\}}{|S|}.$$

A multi-time-interval sequence α is called a *multi-time-interval sequential pattern* or a *frequent multi-time-interval sequence* if the support of α is no less than *min_sup*. The total number of items in a multi-time-interval sequence s is the *length* of the sequence. A multi-time-interval sequence whose length is k is referred to as a *k-multi-time-interval sequence*.

Example 1. Consider the sequential database shown in Fig. 2 with $TI = \{ti_0, ti_1, ti_2, ti_3\}$, where ti_0: $t = 0$, ti_1: $0 < t \leqslant 3$, ti_2: $3 < t \leqslant 6$, ti_3: $6 < t \leqslant \infty$. The 3-multi-time-interval sequence $(b,(ti_1),e,(ti_3,ti_2),c)$ is *contained* in sequences 10, 20, and 30. Therefore, its support is 75%. If minimum support (*min_sup*) is 50%, then $\langle b,(ti_1),e,(ti_3,ti_2),c\rangle$ is a *multi-time-interval sequential pattern* in the sequence database.

Property 2 (Anti-monotonocity property). *If a multi-time-interval sequence is frequent, so are all of its subsequences. Accordingly, if a multi-time-interval sequence is not frequent, then its super sequence will not be either.*

sid	Sequence
10	<(*a*, 1), (*b*, 3), (*c*, 3), (*a*, 5), (*e*, 5), (*c*, 10)>
20	<(*d*, 5), (*a*, 7), (*b*, 7), (*e*, 7), (*d*, 8), (*e*, 8), (*c*, 14), (*d*, 15)>
30	<(*a*, 8), (*b*, 8), (*e*, 11), (*d*, 12), (*b*, 13), (*c*, 13), (*c*, 16)>
40	<(*b*, 15), (*f*, 15), (*e*, 16), (*b*, 17), (*c*, 17)>

Fig. 2. A sequence database.

4. Algorithms

Two efficient algorithms are developed for mining multi-time-interval sequential patterns from databases. Modifying the I-Apriori algorithm develops the first algorithm, the MI-Apriori algorithm. Modifying the I-PrefixSpan algorithm develops the second algorithm, the MI-PrefixSpan algorithm. Sections 4.1 and 4.2 introduce these two algorithms.

4.1. The MI-Apriori algorithm

The algorithm is shown in Fig. 3, where L_k denotes the set of all *frequent* k-multi-time-interval sequences, and C_k denotes the set of all *candidate* k-multi-time-interval sequences. The algorithm proceeds in phases, where the kth phase determines L_k from C_k. In the first phase, L_1 is found from C_1, which is generated by listing all distinct items in the database. Then, sequentially scanning the database determines the supports of all sequences in C_1 and produces L_1 as well. The kth phase is to generate L_k. To accomplish this, C_k is derived by applying the function *MI-Apriori_gen* (L_{k-1}). After obtaining C_k, we get L_k by scanning the database and deleting all infrequent multi-time-interval sequences.

Although the body of the MI-Apriori algorithm is similar to that of the Apriori algorithm, they are different in two ways: (1) the method to generate C_k and (2) the method for computing their supports.

4.1.1. Candidate generation method

We will first discuss the generation of C_2, and then discuss the generation of C_k, where $k > 2$.

The generation of C_2 can be accomplished by joining L_1 with L_1. Since the first item and the second item in C_2, say b and c, may have various time-interval relations, pairs for all possible time-interval relations must be generated. For example, suppose (b) and (c) are in L_1 and $TI = \{ti_0, ti_1, ti_2\}$. Then the following candidates exist in $C_2 = \{\langle b,(ti_0),b\rangle, \langle b,(ti_1),b\rangle, \langle b,(ti_2),b\rangle, \langle c,(ti_0),c\rangle, \langle c,(ti_1),c\rangle, \langle c,(ti_2),c\rangle, \langle b,(ti_0),c\rangle, \langle b,(ti_1),c\rangle, \langle b,(ti_2),c\rangle, \langle c,(ti_0),b\rangle, \langle c,(ti_1),b\rangle, \langle c,(ti_2),b\rangle\}$. In summary, C_2 can be expressed as $L_1 \times TI \times L_1$, where $\times$ denotes joining.

Next, consider the generation of C_k. Let $B = \langle b_1, \&_2, b_2, \&_3, \ldots, b_{k-1}, \&_k, b_k\rangle$ be a frequent k-multi-time-interval sequence in L_k. If two subsequences of B that take the first and the last item away from B exist in L_{k-1}, then the joining of these two subsequences will be a candidate for L_k. Let there be two $(k-1)$-sequences: $l_1 = \langle b_1, \&_2, b_2, \&_3, \ldots, b_{k-2}, \&_{k-1}, b_{k-1}\rangle$ and $l_2 = \langle b_2, \&'_3, b_3, \&'_4, \ldots, b_{k-1}, \&'_k, b_k\rangle$. If l_1 and l_2 satisfy the relation of $\&'_m = \&_m(2, m-1)$ where $3 \leqslant m \leqslant k-1$, then the joining of l_1 and l_2 will produce $\langle b_1, \&_2, b_2, \&_3, \ldots, b_{k-1}, \&'', b_k\rangle$, where $\&'' = (TR_{b_1 b_k}, \&'_k(2, k-1))$. Note that there may exist infeasible time-intervals on generating $(TR_{b_1 b_k})$, which are unnecessary to be included in C_k. Here, two methods are considered in this phase to remove them. First, the *descending property* indicates that the time-intervals in each $\&_i$ must keep descending. Candidates without satisfying this property would not be generated in C_k. Second, the *time-interval information matrix* is built to store the possible time-intervals when two time-intervals are combined. To determine the new time-interval $(TR_{b_1 b_k})$, we can regard $(TR_{b_1 b_k})$ as $(TR_{b_1 b_{k-1}} + TR_{b_{k-1} b_k})$. By checking the time-interval information matrix, we can determine the possible time-intervals for $TR_{b_1 b_k}$. In other words, if an interval does not overlap with $(TR_{b_1 b_{k-1}} + TR_{b_{k-1} b_k})$, it is impossible for that interval to be a time-interval for $(TR_{b_1 b_k})$. For example, the combination of ti_1 $(0 < t \leqslant 3)$ and ti_1 $(0 < t \leqslant 3)$ is $(0 < t \leqslant 6)$, which overlaps with ti_1 $(0 < t \leqslant 3)$ and ti_2 $(3 < t \leqslant 6)$, but disjoints with ti_0 $(t = 0)$ and ti_3 $(6 < t \leqslant \infty)$. Fig. 4 shows the time-interval information matrix constructed for the time-intervals in Example 1.

Below, we give an example to explain the whole process of the generation of C_k $(k > 2)$.

Example 2. Suppose we have $TI = \{ti_0, ti_1, ti_2, ti_3\}$ and $\langle b,(ti_1),c,(ti_1,ti_1),d\rangle$ and $\langle c,(ti_1),d,(ti_2,ti_1),e\rangle$ in L_3. Then, in candidate generation phase, there exist at most four possible candidates: $\langle b,(ti_1),c,(ti_1,ti_1),d,(ti_0,ti_2,ti_1),e\rangle$, $\langle b,(ti_1),c,(ti_1,ti_1),d,$

```
L_1 = Find_1 − frequent − item(S);
    For(k = 2; L_{k-1} ≠ null; k++){
        C_k = MI − Apriori_gen(L_{k−1}, TI); //generate all possible candidates
        For each sequence c ∈ C_k {
            For each sequence s ∈ S{
                If (c contained in s) then{
                c.count++;
                }
            }
        }
    L_k = {c ∈ C_k | c.count ≥ min_sup}
    }
return ∪ L_k;
```

Fig. 3. The *MI-Apriori* algorithm.

$ti_0(t=0)$	ti_0	-	-	-
$ti_1(0<t\leq 3)$	ti_1	ti_1, ti_2	-	-
$ti_2(3<t\leq 6)$	ti_2	ti_2, ti_3	ti_3	-
$ti_3(6<t\leq\infty)$	ti_3	ti_3	ti_3	ti_3
	$ti_0(t=0)$	$ti_1(0<t\leq 3)$	$ti_2(3<t\leq 6)$	$ti_3(6<t\leq\infty)$

Fig. 4. Time-interval information matrix.

$(ti_1,ti_2,ti_1),e\rangle$, $\langle b,(ti_1),c,(ti_1,ti_1),d,(ti_2,ti_2,ti_1),e\rangle$, and $\langle b,(ti_1),c,(ti_1,ti_1),d,(ti_3,ti_2,ti_1),e\rangle$. We first consider the descending property. We find that two of them can be excluded and the remaining ones are $\langle b,(ti_1),c,(ti_1,ti_1),d,(ti_2,ti_2,ti_1),e\rangle$ and $\langle b,(ti_1),c,(ti_1,ti_1),d,(ti_3,ti_2,ti_1),e\rangle$. Next, through checking the time-interval information matrix (see Fig. 4) we find $\langle b,(ti_1),c,(ti_1,ti_1),d,(ti_3,ti_2,ti_1),e\rangle$ is infeasible becauseti_3 does not overlap with $(TR_{bd}+TR_{de})=ti_1+ti_1$. Now, only $\langle b,(ti_1),c,(ti_1,ti_1),d,(ti_2,ti_2,ti_1),e\rangle$ can be considered in C_4.

After we generated candidates in the join phase, finally, the *anti-monotonocity property* is used to further prune infeasible candidates. If a multi-time-interval sequence B in L_k is frequent, then so are all of B's subsequences. Therefore, we can eliminate a candidate in C_k if any of its $(k-1)$-subsequences are not in L_{k-1}.

4.1.2. Support counting method

After the candidate set of k-multi-time-interval sequences is found, an immediate problem is how to compute their supports. To accomplish this, a tree structure, called a candidate tree, is used. The candidate tree is similar to the hash tree adopted in previous research [2]. The major difference is that the previous approach connects each tree branch with an item name, while in our approach two components are attached: an item name and a multi-time-interval value. Besides this difference, the traversal of the tree and the computation of support counts are all similar. For brevity, we omit the details.

4.2. The MI-PrefixSpan algorithm

This subsection proposes the second algorithm for mining multi-time-interval sequential patterns by modifying the I-PrefixSpan algorithm. Before introducing the proposed algorithm, the definitions of prefix, projection, postfix, and projected database are given.

Definition 4. Let $A=\langle(a_1,t_1),(a_2,t_2),(a_3,t_3),\ldots,(a_n,t_n)\rangle$ be a data sequence. A multi-time-interval sequence $B=\langle b_1,\&_2,b_2,\&_3,\ldots,b_{m-1},\&_m,b_m\rangle$ $(m\leqslant n)$ is a *prefix* of A if and only if:

1. $b_i=a_i$ for $1\leqslant i\leqslant m$ and
2. t_i-t_k is in the time-interval of $\&_i(k,k)$ for $2\leqslant i\leqslant m$ and $1\leqslant k\leqslant i-1$.

For example, $\langle a,(ti_1),b,(ti_1,ti_0),c\rangle$ is a prefix of $A=\langle(a,1),(b,3),(c,3),(a,5),(e,5),(c,10)\rangle$. A prefix whose length is k is referred to as a *k-prefix*.

Definition 5. Let $A=\langle(a_1,t_1),(a_2,t_2),(a_3,t_3),\ldots,(a_n,t_n)\rangle$ be a data sequence and $B=\langle b_1,\&_2,b_2,\&_3,\ldots,b_{s-1},\&_s,b_s\rangle$ $(s\leqslant n)$ be a multi-time-interval sequence contained in A. Let $i_1<i_2<\cdots<i_s$ be the indices of the elements in A that match the elements of B. A data subsequence $A':\langle(a'_1,t'_1),(a'_2,t'_2),(a'_3,t'_3),\ldots,(a'_p,t'_p)\rangle$ of A, where $p=s+n-i_s$, is called a projection of A with respect to B if and only if:

1. B is a prefix of A' and
2. the last $n-i_s$ elements of A' are the same as the last $n-i_s$ elements of A.

For example, if a data sequence $A=\langle(a,1),(b,3),(c,3),(a,5),(e,5),(c,10)\rangle$ is projected with $B=\langle a\rangle$, two different projections A' are obtained. The first is $\langle(a,1),(b,3),(c,3),(a,5),(e,5),(c,10)\rangle$ and the second is $\langle(a,5),(e,5),(c,10)\rangle$.

The example above indicates that projecting a data sequence A may produce more than one projection A' with different positions. To differentiate these different projections from the same source, the tag $[sid,p,t]$ is attached to each projected sequence, where *sid* is the identifier of the sequence, p is the list of positions that matches the elements of A, and t is the list of times corresponding to positions in p.

Therefore, if a data sequence $A=\langle(a,1),(b,3),(c,3),(a,5),(e,5),(c,10)\rangle$ is projected with $B=\langle a\rangle$, two different projections A' are obtained:

- [sid,(1),(1)]: $\langle(a,1),(b,3),(c,3),(a,5),(e,5),(c,10)\rangle$.
- [sid,(4),(5)]: $\langle(a,5),(e,5),(c,10)\rangle$.

Similarly, if A is projected with $B = \langle a,(ti_1),b,(ti_1,ti_0),c\rangle$, one projection A' is obtained, [sid,(1,2,3),(1,3,3)]: $\langle(a,1),(b,3),(c,3),(a,5),(e,5),(c,10)\rangle$. The prefix B occurs in A at a list of positions: (1,2,3).

Definition 6. Let $A' = \langle(a'_1,t'_1),(a'_2,t'_2),\ldots,(a'_p,t'_p)\rangle$ be the projection of A with respect to a prefix $B = \langle b_1,\&_2,b_2,\&_3,\ldots,b_{s-1},\&_s,b_s\rangle$. Then $\lambda = \langle(a'_{s+1},t'_{s+1}),(a'_{s+2},t'_{s+2}),\ldots,(a'_p,t'_p)\rangle$ is called the postfix of A with respect to prefix B.

For instance, if a data sequence $A = \langle(a,1),(b,3),(c,3),(a,5),(e,5),(c,10)\rangle$ is projected with $B = \langle a\rangle$, two different postfixes are [sid,(1),(1)]: $\langle(b,3),(c,3),(a,5),(e,5),(c,10)\rangle$ and [sid,(4),(5)]: $\langle(e,5),(c,10)\rangle$. Similarly, if A is projected with $B = \langle a,(ti_1),b,(ti_1,ti_0),c\rangle$, the postfix is [sid,(1,2,3),(1,3,3)]: $\langle(a,5),(e,5),(c,10)\rangle$.

The *MI-PrefixSpan* algorithm solves the problem using a divide and conquer strategy. Initially, α = *null* is set. For a given α-project database $S|_\alpha$, the algorithm first finds all frequent 1-patterns. For each frequent 1-pattern in $S|_\alpha$, such as γ, append γ to α to form a multi-time-interval sequential pattern α', and then construct the α'-project database $S|'_\alpha$. Recursively finding frequent multi-time-interval sequences in $S|_{\alpha'}$ finally yields all frequent multi-time-interval sequences in S. The MI-PrefixSpan algorithm is shown in Fig. 5.

Although the MI-PrefixSpan algorithm looks similar to the PrefixSpan algorithm, there are three major differences: (1) how to construct the projected database $S_{\alpha'}$, (2) how to enumerate the candidate patterns of α', and (3) how to compute the supports of these candidate patterns. First, we consider how to construct α'-project database $S|_{\alpha'}$. Let $\alpha' = \langle\alpha,\&,\gamma\rangle$, where α is a frequent pattern that we previously used to project the database, γ denotes a frequent item in $S|_\alpha$, and & denotes a multi-time-interval between α and γ. Let A in $S|_\alpha$ be the postfix of s in S with respect to α, and let λ in $S|_{\alpha'}$ be the postfix of s with respect to prefix α'. Then we can obtain λ by projecting A with respect to γ, because $\alpha' = \langle\alpha,\&,\gamma\rangle$. Therefore, the new projected database $S|_{\alpha'}$ can be constructed by reading every sequence A in $S|_\alpha$ and inserting the corresponding sequence λ into $S|_{\alpha'}$. The algorithm for constructing α'-project database $S|_{\alpha'}$ from $S|_\alpha$ is shown in Fig. 6.

Example 3. Consider the sequence database shown in Fig. 2 with $TI = \{ti_0,ti_1,ti_2,ti_3\}$, ti_0: $t = 0$, ti_1: $0 < t \leqslant 3$, ti_2: $3 < t \leqslant 6$, ti_3: $6 < t \leqslant \infty$, and with *min_sup* = 50%. If $\alpha = \langle a\rangle$, the α–project database $S-_{\langle a\rangle}$ is:

- [10,(1),(1)]: $\langle(b,3),(c,3),(a,5),(e,5),(c,10)\rangle$.
- [10,(4),(5)]: $\langle(e,5),(c,10)\rangle$.
- [20,(2),(7)]: $\langle(b,7),(e,7),(d,8),(e,8),(c,14),(d,15)\rangle$.
- [30,(1),(8)]: $\langle(b,8),(e,11),(d,12),(b,13),(c,13),(c,16)\rangle$.

Let $\alpha' = \langle a,(ti_1),e\rangle$. Then we can construct the α'-project database $S|_{\langle a,(ti_1),e\rangle}$ by projecting $S|_{\langle a\rangle}$ with respect to $\langle e\rangle$. The results are given below.

- [20,(2,6),(7,8)]: $\langle(c,14),(d,15)\rangle$.
- [30,(1,3),(8,11)]: $\langle(d,12),(b,13),(c,13),(c,16)\rangle$.

Next, we consider how to determine the candidate patterns and count their supports. Since the multi-time-interval relationships of items are complicated, we use a table, named *table*, to record the supports. Let $\alpha = \langle b_1,\&_2,b_2,\&_3,\ldots,b_{k-1},\&_k,b_k\rangle$. Then we construct the *table*, where each column corresponds to a frequent item γ in $S|_\alpha$ and each row corresponds to a multi-time-interval relationship in $\&_{k+1}(1,k) = (TR_{b_1b_{k+1}},TR_{b_2b_{k+1}},\ldots,TR_{b_kb_{k+1}})$. The table allows us to identify all multi-time-interval candidate patterns of $\alpha' = \langle\alpha,\&_{k+1}(1,k),\gamma\rangle$ and record their supports. Similar to the generation

Subroutine MI - PrefixSpan(α, k, $S|_\alpha$)
Paramenter : α : a multi - time - interval sequential pattern
k : the length of α
γ : a frequent item
Methods :
Scan $S|_\alpha$ one time
→ If $k = 0$, then find all frequent items in $S|_\alpha$.
For every frequent item γ, append γ to α as α'.
Output all α'.
→ If $k > 0$, then construct the *Table* for α.
⇒ Find all frequent items γ in $S|_\alpha$
⇒ Generate all kinds of possible multi - time - intervals $\&_{k+1}$.
For every cell $Table(\&_{k+1},\gamma) \geq min\_sup$, append $(\&_{k+1},\gamma)$ to α as α'.
Output all α'.
For each α', construct α' – projected database, and call MI - PrefixSpan (α', $k+1$, $S|_{\alpha'}$).

Fig. 5. The *MI-PrefixSpan* algorithm.

Paramenter : γ : a frequent item
k : the length of α
p : denotes a list of positions when α occurs.
s_i : the ith position in sequence s.
$time$: the time of s_i
t : a list of times correspond ing to p.
t_j : the time of the j - th position in p.
$[sid, p, t]$: is the tags attached to sequence s in $S|_\alpha$
method :
For each sequence A with tag $[sid, p, t]$ in $S|_\alpha$
For ecah position s_i in A where γ occurs,
If $time - t_j$ is in the time interval of $\&_{k+1}(j, j)$ for $1 \le j \le k$, then
$p = p + (i)$
$t = t + (time)$
Let λ be the postfix of A with respect to prefix γ.
Insert sequence λ with tag $[sid, p, t]$ into $S|_{\alpha'}$.

Fig. 6. Construction of α'-projected database $S|_{\alpha'}$.

of C_k in the MI-Apriori algorithm, the descending property and the time-interval information matrix are used to screen out infeasible candidates before the support counting process.

Example 4. Suppose $\alpha = \langle a,(ti_1),e\rangle$, and $\{c,d\}$ are frequent items γ in $S|_{\langle a,(ti_1),e\rangle}$. Since $TI = \{ti_0, ti_1, ti_2, ti_3\}$ and there are two time-intervals in $\&_3(1,2) = (TR_{b_1b_3}, TR_{b_2b_3})$, we have a total of $4 \times 4 = 16$ rows in the table. We construct the table shown in Fig. 7a, and see that there are 32 candidate patterns of α'. We first consider the descending property, and the *table* shown in Fig. 7a can be reduced to Fig. 7b. Table cells with "–" denote candidates that have been removed. By applying the time-interval information matrix, secondly, the final candidate set is shown in Fig. 7c.

Next, we consider how to count the supports of the cells in *table*. Cell $(\&_{k+1}, \gamma)$ of *table* records the number of transactions in $S|_\alpha$ that contain item γ and the time differences between this item and the items preceding α match $\&_{k+1}$. After every sequence has been processed, the counts in the table can be obtained and the frequent cells can be identified. If the cell $(\&_{k+1}, \gamma)$ of *table*is frequent, we can add $(\&_{k+1}, \gamma)$ to α to form a multi-time-interval sequential pattern α'. Furthermore, we can construct the α'-projected database $S|_{\alpha'}$. By repeating this procedure, we can find all the multi-time-interval patterns in S.

Example 5. The *table* for $\alpha = \langle a,(ti_1),e\rangle$ is shown in Fig. 7c and the $\langle a,(ti_1),e\rangle$-projected database $S|_{\langle a,(ti_1),e\rangle}$ is as follows:

- [20,(2,6),(7,8)]: $\langle(c,14),(d,15)\rangle$.
- [30,(1,3),(8,11)]: $\langle(d,12),(b,13),(c,13),(c,16)\rangle$.

To determine the supports, these two sequences should be processed.

- In [20,(2,6),(7,8)]: $\langle(c,14),(d,15)\rangle$, c occurs in position 1 ($p_i = 1$) and the time is 14 ($time = 14$). $Time - t_1 = 14 - 7 = 7$ is in the time-interval $\&_3(1,1) = ti_3$ and $time - t_2 = 14 - 8 = 6$ is in the time-interval $\&_3(2,2) = ti_2$. Therefore, we add 1 to the cell of *table* $((ti_3, ti_2), c)$ and go to the next sequence with a different *sid*.

Table	c	d
(ti_0, ti_0)		
(ti_0, ti_1)		
(ti_0, ti_2)		
(ti_0, ti_3)		
(ti_1, ti_0)		
(ti_1, ti_1)		
(ti_1, ti_2)		
(ti_1, ti_3)		
(ti_2, ti_0)		
(ti_2, ti_1)		
(ti_2, ti_2)		
(ti_2, ti_3)		
(ti_3, ti_0)		
(ti_3, ti_1)		
(ti_3, ti_2)		
(ti_3, ti_3)		

(a)

Table	c	d
(ti_0, ti_0)		
(ti_0, ti_1)	–	–
(ti_0, ti_2)	–	–
(ti_0, ti_3)	–	–
(ti_1, ti_0)		
(ti_1, ti_1)		
(ti_1, ti_2)	–	–
(ti_1, ti_3)	–	–
(ti_2, ti_0)		
(ti_2, ti_1)		
(ti_2, ti_2)		
(ti_2, ti_3)		
(ti_3, ti_0)	–	–
(ti_3, ti_1)		
(ti_3, ti_2)		
(ti_3, ti_3)		

(b)

Table	c	d
(ti_0, ti_0)	–	–
(ti_0, ti_1)	–	–
(ti_0, ti_2)	–	–
(ti_0, ti_3)	–	–
(ti_1, ti_0)		
(ti_1, ti_1)		
(ti_1, ti_2)	–	–
(ti_1, ti_3)	–	–
(ti_2, ti_0)	–	–
(ti_2, ti_1)		
(ti_2, ti_2)		
(ti_2, ti_3)		
(ti_3, ti_0)	–	–
(ti_3, ti_1)	–	–
(ti_3, ti_2)		
(ti_3, ti_3)		

(c)

Fig. 7. (a) Original *table*, (b) after considering the descending property, and (c) after considering the time-interval information matrix.

Table	c	d
(ti_1, ti_0)	0	0
(ti_1, ti_1)	0	0
(ti_2, ti_1)	1	1
(ti_2, ti_2)	0	0
(ti_3, ti_2)	2	0
(ti_3, ti_3)	0	1

Fig. 8. The *table* for $\langle a,(ti_1),e\rangle$ with supports.

- In [30,(1,3),(8,11)]: $\langle(d,12),(b,13),(c,13),(c,16)\rangle$, c occurs in position 3 ($p_i = 3$) and position 4 ($p_i = 4$). When $p_i = 3$, $time - t_1 = 13 - 8 = 5$ is in the time-interval $\&_3(1,1)=ti_2$ and $time - t_2 = 13 - 11 = 2$ is in the time-interval $\&_3(2,2) = ti_1$. So, we add 1 to the cell of *table* $((ti_2,ti_1), c)$. When $p_i = 4, time - t_1 = 16 - 8 = 8$ is in the time-interval $\&_3(1,1)=ti_3$ and $time - t_2 = 16 - 11 = 5$ is in the time-interval $\&_3(2,2) = ti_2$. So, we add 1 to the cell of *table* $((ti_3,ti_2),c)$. Finally, we go to the next sequence with a different *sid*.

After scanning all of sequence s_i in $S|_{\langle a,(ti_1),e\rangle}$, we can construct *table* for α with supports as shown in Fig. 8.

5. Empirical evaluation

We used several synthetic datasets and two real-life datasets to evaluate the performance of our algorithms. Seven algorithms, including the Apriori [2], the PrefixSpan [28], the I-Apriori [6], the I-PrefixSpan [6], the MI-Apriori, the MI-PrefixSpan, the MI-PrefixSpan with a pseudo projection technique, and post-processing algorithms, were implemented via Java language and tested on a Pentium IV-2.0G Windows 2000 system with 1024 MB of main memory and with JBuilder (7.0 version) as the Java execution environment.

The pseudo projection technique can be applied for efficiently saving both time and space. The main idea is that instead of generating numerous physical projections in main memory, one can register the index of the projected position with its sequence identifier in the sequence. Through these indexes, we can easily divide the searching space and then retrieve all necessary information for finding sequential patterns. Notice that this approach is a memory-based technique, which means it is efficient only when the original database or the projected database can fit in main memory.

The post-processing algorithm first generates all possible candidate multi-time-interval patterns from traditional sequential patterns discovered by the PrefixSpan algorithm. After that, with one additional DB scan, we can discover all multi-time-interval sequential patterns. Without creating a novel algorithm, the post-processing algorithm can still discover all multi-time-interval sequential patterns. Since this may generate many infeasible candidates, we also consider both the descending property and the time-interval information matrix. Because the post-processing algorithm determines the supports of all candidate patterns in a single stage, the anti-monotonocity property cannot be adopted here.

5.1. Synthetic datasets

We apply the synthetic data generation algorithm in [6] to generate synthetic datasets. Each data sequence contains a sequence of itemsets. However, we assign different time values to the items in different itemsets but the same time values to those in the same itemsets. A value w is drawn from a Poisson distribution with mean T_I for each customer. The drawn value w represents the average time-interval between successive itemsets in the sequence of a particular customer. After that, the intervals of successive itemsets of this customer are determined by repeatedly drawing values from a Poisson distribution with mean w.

Table 1 lists the parameters used in the simulation; the first eight parameters are classical parameters used in previous research [2], but the last parameter, T_I, is a new parameter created for the problem considered here. In the simulation, some parameters are fixed: $N = 10000$, $N_S = 5000$, $N_I = 25{,}000$, $T_I = 10$, and $|D| = 250{,}000$. All the intervals in successive itemsets are initially collected to determine the set *TI*. These values are separated into five equal depth intervals ti_1, ti_2, ti_3, ti_4 and ti_5, which each contains roughly the same amount of data. The resulting set is $TI = \{ti_0, ti_1, ti_2, ti_3, ti_4, ti_5\}$, where ti_0: $t = 0$. The eight datasets in Table 2 are used for comparison.

The first comparison is executed based on the first dataset in Table 2, where the *min_sup* varies from 2.5% to 1.5%. Figs. 9 and 10 summarize the run time of these seven algorithms with the following results:

1. When the *min_sup* decreases, the performance of an Apriori-like algorithm, such as the GSP, I-Apriori, and MI-Apriori, is inferior to a PrefixSpan-like algorithm. Intuitively, we may think that the algorithms for finding multi-time-interval patterns are the most time-consuming, followed by those for finding single-time-interval patterns, and finally those for finding patterns without intervals. It is worth noting that the results did not match our expectations, especially when the

Table 1
Parameters.

$\|D\|$	Number of customers (hundred thousand)
$\|C\|$	Average number of transactions per customer
$\|T\|$	Average number of items per transaction
$\|S\|$	Average length of maximal potentially large sequences
$\|I\|$	Average size of itemsets in maximal potentially large sequences
N_S	Number of maximal potentially large sequences (fixed: 5000)
N_I	Number of maximal potentially large itemsets (fixed: 25,000)
N	Number of items (fixed: 10000)
T_I	Average length of time intervals (fixed: 10)

Table 2
Datasets.

	Name	$\|D\|$	$\|C\|$	$\|T\|$	$\|S\|$	$\|I\|$
1	D2.50-C10-T2.50-S4-I1.25	2.50	10	2.50	4	1.25
2	D2.50-C10-T3.75-S4-I1.25	2.50	10	3.75	4	1.25
3	D2.50-C15-T2.50-S4-I1.25	2.50	15	2.50	4	1.25
4	D2.50-C10-T5.00-S4-I1.25	2.50	10	5.00	4	1.25
5	D2.50-C20-T2.50-S4-I1.25	2.50	20	2.50	4	1.25
6	D1.25-C10-T2.50-S4-I1.25	1.25	10	2.50	4	1.25
7	D3.75-C10-T2.50-S4-I1.25	3.75	10	2.50	4	1.25
8	D5.00-C10-T2.50-S4-I1.25	5.00	10	2.50	4	1.25

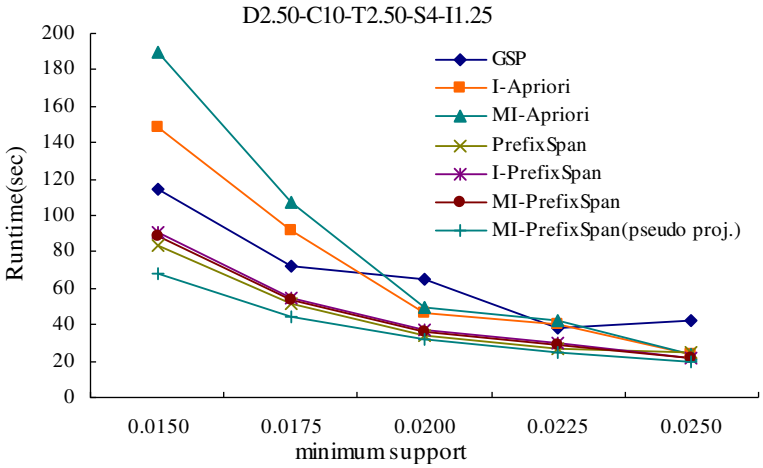


Fig. 9. The run time for dataset D2.50-C10-T2.50-S4-I1.25.

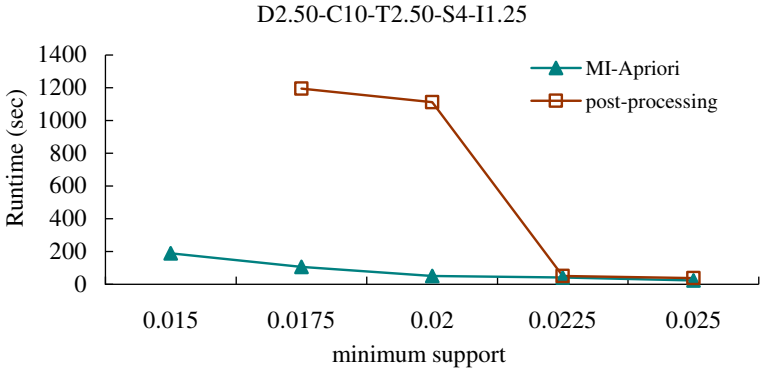


Fig. 10. The run times for the MI-Apriori algorithm and the post-processing algorithm.

min_sup increases. We observed (or We found) the most important factor influencing run time is not whether the algorithms or patterns are complicated or not, but whether we generate a large set of patterns, resulting in a longer processing time for these patterns. When the *min_sup* increases, multi-time-interval patterns are more likely to become infrequent, since they are more complicated than the other two kinds of patterns; this causes fewer candidate patterns and shorter run times.

2. Fig. 10 compares the run times of the MI-Apriori algorithm and the post-processing algorithm. The results show that the run time of the post-processing algorithm is much longer than that of the MI-Apriori algorithm, especially when the *min_sup* decreases. This phenomenon can be easily explained. Suppose we have a k-sequential pattern without time-intervals and there are r different types of time-intervals. From this pattern we can generate $r^{C_2^k}$ candidates for multi-time-interval patterns. This analysis indicates that the number of candidate multi-time-interval patterns grows exponentially with the number of traditional sequential patterns.

Next, the scalabilities of the six algorithms are compared. To this end, three tests are designed based on dataset D2.50-C10-T2.50-S4-I1.25. During each test, all parameters are fixed and the *min_sup* is set to 2.5%. However, in each test, a parameter is varied to see how the algorithms scale-up with the parameter. The first test varies the number of customers, $|D|$, from 125,000 to 500,000 (Fig. 11), the second varies the average number of data sequences per customer, $|C|$, from 10 to 20 (Fig. 12), and the final test varies the average number of items bought per transaction, $|T|$, from 2.5 to 5.0 (Fig. 13). The results indicate that the run times of all these algorithms grow almost linearly with respect to $|D|$. But the other two parameters, $|C|$ and $|T|$, have a stronger impact on the performance of the algorithms; they seem to scale up exponentially with $|T|$ and $|C|$. We also note that without the pseudo projection technique, the PrefixSpan-like algorithm becomes worse than the Apriori-like algorithm when $|T|$ and $|C|$ are increased. This is because the PrefixSpan-based algorithm needs to recursively build projected databases in main memory, which causes their performance to be more sensitive to the amount of main memory available and usage. When we increase the sequence length, more main memory is needed to store the compressed database. In turn, the operating system is forced to swap the data between the main memory and the hard disks, causing a sharp increase in the run time. Since the size of each dataset in our scalability tests can be fit into main memory, we also perform the

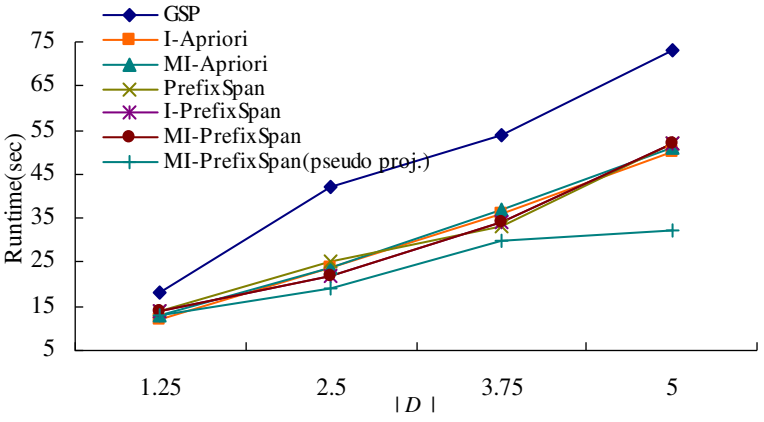


Fig. 11. Scale-up with $|D|$: number of customers.

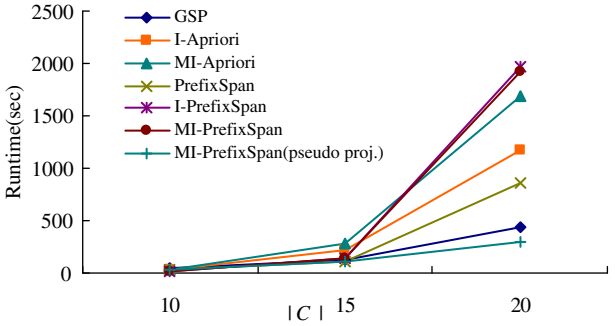


Fig. 12. Scale-up with $|C|$: average number of transactions per customer.

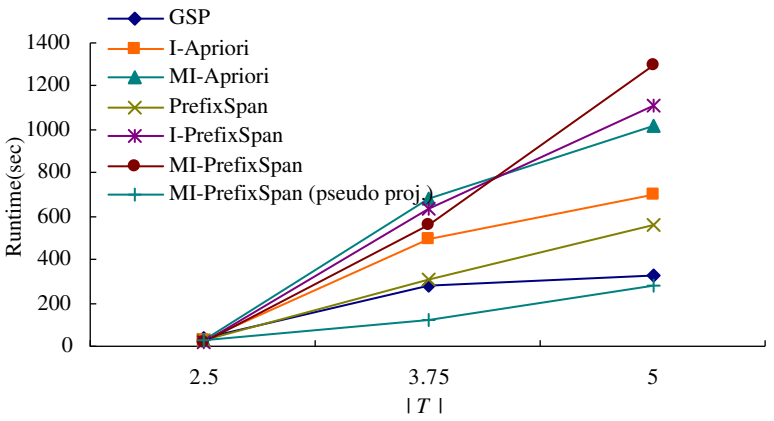


Fig. 13. Scale-up with $|T|$: average number of items per customer.

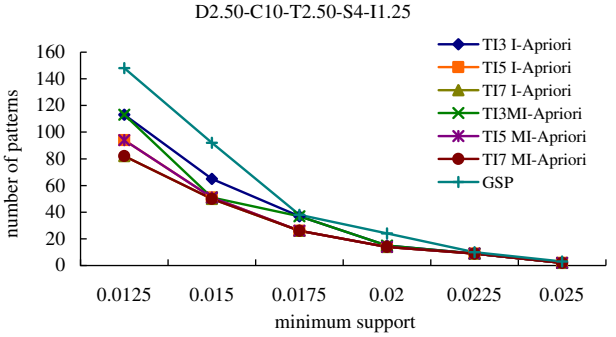


Fig. 14. The number of patterns versus pattern types.

pseudo projection technique on the MI-PrefixSpan algorithm. The results show that the method improves the efficiency of the MI-PrefixSpan algorithm, and both time and space are significantly reduced.

Finally, we compare the total number of patterns and the number of patterns in different phases obtained from the three approaches: sequential patterns without intervals, those with single-time-intervals, and those with multi-time-intervals. In the experiment, intervals are partitioned into three, five, and seven intervals of equal depth (each with roughly the same amount of data). Accordingly, time-interval sets *TI*-3 = $\{ti_0, ti_1, ti_2, ti_3\}$, *TI*-5=$\{ti_0, ti_1, ti_2, ti_3, ti_4, ti_5\}$, and *TI*-7 = $\{ti_0, ti_1, ti_2, ti_3, ti_4, ti_5, ti_6, ti_7\}$ are obtained, where ti_0: $t = 0$. The tests are based on the dataset D2.50-C10-T2.50-S4-I1.25. During each test, all parameters are fixed and the *min_sup* varies from 1.25% to 2.5% to determine how the number of total patterns and number of patterns in different phases change with the *min_sup*. Fig. 14 shows the results. The traditional pattern is the highest, followed by the *TI*-3 of the single-time-interval, the *TI*-3 of the multi-time-interval, the *TI*-5 of the single-time-interval and of the multi-time-interval, and finally the *TI*-7 of the single-time-interval and of the multi-time-interval. We observed that the number of patterns in the *TI*-3 of the multi-time-interval is fewer than that in the *TI*-3 of the single-time-interval. Therefore, with the new approach some regularities that would be represented as single-time-interval sequential patterns will not be represented in the discovered collection of multi-time-interval patterns. Additionally, the results indicate that when we divide the time line into more intervals, the number of patterns decreases. This is because more intervals cause the supports of the candidate patterns to decrease, which in turn generates fewer patterns.

5.2. Real datasets

In this section, we want to observe how the multi-time-interval sequential patterns work in practice. Two real datasets, called SC-POS and TW-STOCK, are used in our experiments. The SC-POS dataset recorded sales data from a chain supermarket in Taiwan between 2001/12/27 and 2002/12/31. It contains 18,162 items and 102,601 customers' data-sequences. The length of a data-sequence ranges from 1 to thousands, but the majority of the data-sequences in the SC-POS dataset are quite long. It reveals that most members have long-term purchasing behavior and most purchases contain various items. The TW-STOCK dataset recorded historical weekly price/volume data of 30 technology stocks between 2006/1/1 and 2006/5/31 from Taiwan stock market. All the price/quantity data are discretized into intervals so that each discretized interval can be treated as an item. In TW-STOCK, it contains 11 different kinds of items and all data-sequences are equal-length.

We first use the SC-POS to compare the run times. The *min_sup* is varied from 2.0% to 3.0% and $TI = \{ti_0, ti_1, ti_2, ti_3, ti_4, ti_5\}$, where ti_0: $t = 0$, ti_1: $0 < t \leqslant 6$, ti_2: $6 < t \leqslant 12$, ti_3: $12 < t \leqslant 18$, ti_4: $18 < t \leqslant 20$, and ti_5: $20 < t \leqslant \infty$. In our test, we found that neither the original sequence database nor the projected database can be fit into main memory since the length of a sequence can be very long in supermarket data. The PrefixSpan-like algorithm is not applicable in this situation even though we employ the pseudo projection technique. Therefore, Fig. 15 only shows the run times of the I-Apriori and MI-Apriori algorithms.

Next, we discover three different kinds of patterns from the SC-POS, including traditional sequential patterns, single-time-interval sequential patterns, and multi-time-interval sequential patterns. The results are shown in Fig. 16. Here, we generalize the SC-POS dataset to a higher level of abstraction. The *min_sup* is set to 10% and $TI = \{ti_0, ti_1, ti_2, ti_3, ti_4, ti_5\}$, where ti_0: $t = 0$, ti_1: $0 < t \leqslant 20$, ti_2: $20 < t \leqslant 40$, ti_3: $40 < t \leqslant 60$, ti_4: $60 < t \leqslant 80$, and ti_5: $80 < t \leqslant \infty$.

As shown in Fig. 16, traditional approach only shows the order of the three items: *Canned Food*(*CF*), *Egg*(*E*), and*Natural Food*(*NF*) in a sequential pattern. Due to its lack of information, a single-time-interval approach is utilized and a pattern $\langle CF, ti_4, E, ti_1, NF\rangle$ is discovered. This means that after buying a *Canned Food*, the customer returned to buy an *Egg* in 60–80 days, and then a *Natural Food* in 0–20 days. We also see that there are five more single-time-interval patterns describing the relationship ($ti_1 \sim ti_5$) between items *Canned Food* and *Natural Food*. The question that arises immediately is what kind of time-interval the items *Canned Food* and *Natural Food* have in a single-time-interval pattern $\langle CF, ti_4, NF, ti_1, E\rangle$. The multiple-time-interval approach can easily solve this problem and all the time-intervals can be found.

Finally, we evaluate two types of sequential patterns (both single-time-interval and multi-time-interval) discovered from the TW-STOCK dataset. The *min_sup* is set to 40% and all time-intervals are set to 4 weeks. The patterns are illustrated in Fig. 17, where *P* denotes the percentages of week's range of a stock price (i.e. $\frac{weeklyhigh\text{-}weeklylow}{weeklyopen\text{-}price}$) and *V* denotes the degree of weekly volume. We can see that the multi-time-interval sequential pattern reveals all time-intervals between any two itemsets. Without this piece of information, it becomes unavailable to acquire the exact time-interval between two non-successive itemsets. Moreover, the possible range of a time-interval between two non-successive is getting larger while their

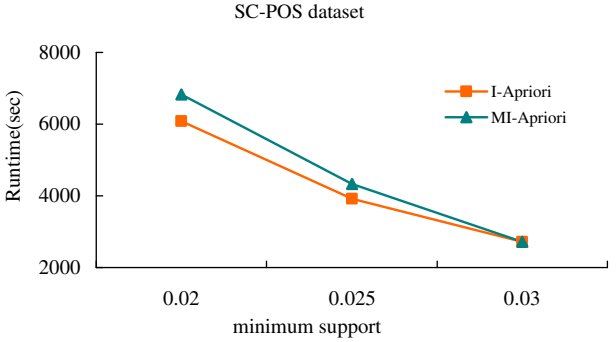
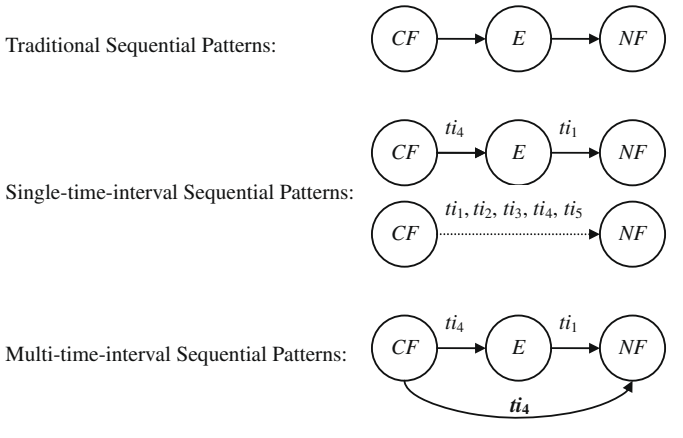


Fig. 15. The run times of I-Apriori and MI-Apriori.



ti_1: $0<t\leq20$, ti_2: $20<t\leq40$, ti_3: $40<t\leq60$, ti_4: $60<t\leq80$, and ti_5: $t>80$

Fig. 16. Three different kinds of patterns discovered from the SC-POS dataset.

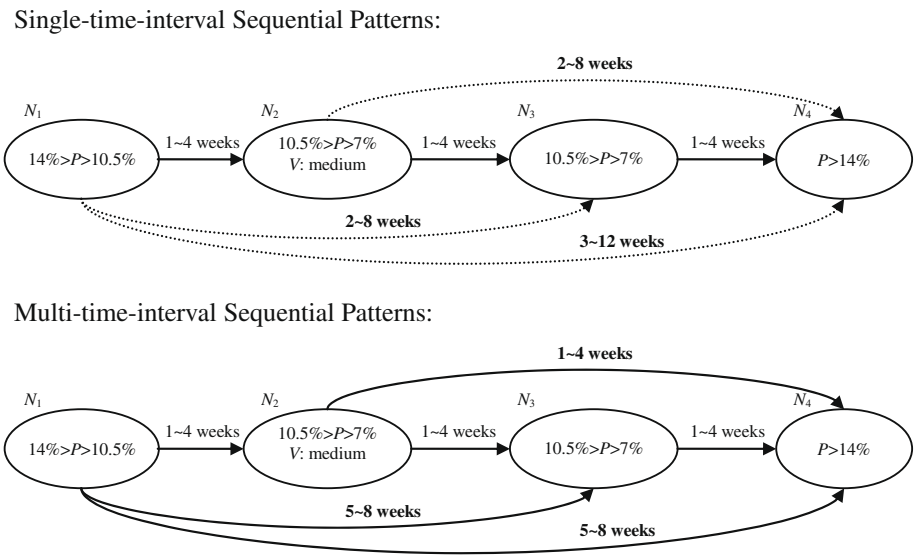


Fig. 17. Two types of pattern discovered from the TW-STOCK dataset.

distance is increasing. For example, the possible time-interval between itemset N_1 and N_3 is 2–8 weeks while the possible time-interval between N_1 and N_4 is 3–12 weeks. Hence, if the single-time-interval sequential patterns are very long, the possible range of time-interval will be very huge and patterns may become almost meaningless in understanding two non-successive itemsets.

6. Conclusion

This work presents a novel multi-time-interval sequential pattern mining approach. The multi-time-interval sequential pattern reveals not only the order of items but also time-intervals of items. The patterns allow us to realize what customers might buy in the next h steps, where $h \geqslant 2$, rather than only the next step. With this extended ability, we can realize long-term customer behaviors, helping us take appropriate actions beforehand.

Two algorithms have been developed in this paper: the MI-Apriori algorithm and the MI-PrefixSpan algorithm. The results of the performance analysis show if the data sequences are of short or medium length, the MI-PrefixSpan algorithm outperforms the MI-Apriori algorithm. But if most of the data sequences are long, the MI-Apriori algorithm performs much better than the MI-PrefixSpan algorithm. This result indicates that both algorithms are indispensable since each one is preferable under certain circumstances. The pseudo projection technique is well performed to improve both run time and memory usage of the MI-Prefixspan algorithm when the projected database can fit in main memory. The technique employing in [28] to advance their algorithm's performance drew the same conclusion as ours.

This paper can be extended in several ways. First, we can apply fuzzy theory or rough set theory to partition the intervals, creating a situation where interval boundaries are no longer fixed but flexible. Second, a hierarchy of time-intervals can be developed so that a larger interval is formed from a set of smaller intervals. Using such an extension, multiple-level multi-time-interval sequential patterns can be determined from sequence data. Third, we can require that the generated patterns must comply with some given meta-forms. This would help us avoid many patterns that do not fit our expectations or applications.

Acknowledgements

The authors would like to thank the Editor and anonymous reviewers for their valuable comments to improve this paper. This research was supported by the National Science Council of the Republic of China under the Grants NSC 97-2410-H-309-020.

References

[1] R. Agrawal, C. Faloutsos, A. Swami, Efficient similarity search in sequence databases, in: Proceedings of the 4th International Conferecne on Foundations of Data Organization and Algorithms, 1993, pp. 69–84.
[2] R. Agrawal, R. Srikant, Mining sequential patterns, in: Proceedings of the 11th International Conference on Data Eng., March 1995, pp. 3–14.

[3] C. Antunes, A.L. Oliveira, Generalization of pattern-growth methods for sequential pattern mining with gap constraints, in: Proceedings of the 6th International Conference on Machine Learning and Data Mining, 2003, pp. 239–251.
[4] B.C.H. Chang, S.K. Halgamuge, Fuzzy sequence pattern matching in zinc finger domain proteins, in: Joint 9th IFSA World Congress and 20th NAFIPS International Conference, vol. 2, 2001, pp. 1116–1120.
[5] Y.L. Chen, S.S. Chen, P.Y. Hsu, Mining hybrid sequential patterns and sequential rules, Information Systems 27 (5) (2002) 345–362.
[6] Y.L. Chen, M.C. Chiang, M.T. Ko, Discovering time-interval sequential patterns in sequence databases, Expert Systems with Applications 25 (3) (2003) 343–354.
[7] Y.L. Chen, T.C.K. Huang, A novel knowledge discovering model for mining fuzzy multi-level sequential patterns in sequence databases, Data and Knowledge Engineering 66 (3) (2008) 349–367.
[8] L. Chang, T. Wang, D. Yang, H. Luan, S. Tang, Efficient algorithms for incremental maintenance of closed sequential patterns in large databases, Data and Knowledge Engineering 68 (1) (2009) 68–106.
[9] M.S. Chen, J. Han, P.S. Yu, Data mining: an overview from a database perspective, IEEE Transactions on Knowledge and Data Engineering 8 (6) (1996) 866–883.
[10] M.S. Chen, J.S. Park, P.S. Yu, Efficient data mining for path traversal patterns, IEEE Transactions on Knowledge and Data Engineering 10 (2) (1998) 209–221.
[11] T.P. Exarchos, M.G. Tsipouras, C. Papaloukas, D.I. Fotiadis, A two-stage methodology for sequence classification based on sequential pattern mining and optimization, Data and Knowledge Engineering 66 (3) (2008) 467–487.
[12] W.J. Frawley, G. Piatetsky-Shapiro, C.J. Matheus, Knowledge Discovery in Database: An Overview, AAAI/MIT Press, Cambridge, MA, 1991.
[13] M. Garofalakis, R. Rastogi, K. Shim, Mining sequential patterns with regular expression constraints, IEEE Transactions on Knowledge and Data Engineering 14 (3) (2002) 530–552.
[14] J. Han, G. Dong, Y. Yin, Efficient mining of partial periodic patterns in time series database, in: Proceedings of the 1999 International Conference on Data Engineering, 1999, pp. 106–115.
[15] J. Han, W. Gong, Y. Yin, Mining segment-wise periodic patterns in time-related databases, in: Proceedings of the 1998 International Conference on Knowledge Discovery and Data Mining, 1998, pp. 214–218.
[16] J. Han, M. Kamber, Data Mining: Concepts and Techniques, Academic Press, New York, 2001.
[17] J. Han, J. Pei, B. Mortazavi-Asl, Q. Chen, U. Dayal, M.-C. Hsu, Freespan: frequent pattern-projected sequential pattern mining, in: Proceedings of the International Conference on Knowledge Discovery and Data Mining, 2000, pp. 355–359.
[18] M.V. Joshi, G. Karypis, V. Kumar. A universal formulation of sequential patterns, Technical Report, Department of Computer Science, University of Minnesota, 1999.
[19] M. Leleu, C. Rigotti, J.-F. Boulicaut, G. Euvrard, Constraint-based mining of sequential patterns over datasets with consecutive repetitions, Lecture Notes in Computer Science 2838 (2003) 303–314.
[20] C. Li, P.S. Yu, V. Castelli, Hierarchyscan: a hierarchical similarity search algorithm for databases of long sequences, in: Proceedings of the 12th International Conference on Data Engineering, 1996, pp. 546–553.
[21] M.Y. Lin, S.Y. Lee, Incremental update on sequential patterns in large databases, in: Proceedings of the 10th IEEE International Conference on Tools with Artificial Intelligence, 1998, pp. 24–31.
[22] M.Y. Lin, S.Y. Lee, S.S. Wang, DELISP: efficient discovery of generalized sequential patterns by delimited pattern-growth technology, in: Proceedings of the 6th Pacific-Asia Conference on Knowledge Discovery and Data Mining, 2002, pp. 198–209.
[23] S. Ma, J.L. Hellerstein, Mining partially periodic event patterns with unknown periods, in: Proceedings of the 17th International Conference on Data Engineering, 2001, pp. 205–214.
[24] H. Mannila, H. Toivonen, A. Inkeri Verkamo, Discovery of frequent episodes in event sequences, Data Mining and Knowledge Discovery 1 (3) (1997) 259–289.
[25] F. Masseglia, P. Poncelet, M. Teisseire, Pre-processing time constraints for efficiently mining generalized sequential patterns, in: Proceedings of the 11th International Symposium on Temporal Representation and Reasoning (TIME'04), 2004, pp. 87–95.
[26] J.Z. Mohammed, Parallel sequence mining on shared-memory machines, Journal of Parallel and Distributed Computing 61 (3) (2001) 401–426.
[27] J. Pei, J. Han, B. Mortazavi-Asl, and H. Zhu, Mining Access Patterns Efficiently from Web Logs, Proc. 2000 Pacific-Asia Conf. Knowledge Discovery and Data Mining, pp. 396–407, 2000.
[28] J. Pei, J. Han, B. Mortazavi-Asl, J. Wang, H. Pinto, Q. Chen, U. Dayal, M.-C. Hsu, Mining sequential patterns by pattern-growth: the prefixspan approach, IEEE Transaction on Knowledge and Data Engineering 16 (11) (2004) 1424–1440.
[29] G. Song, G. Salvendy, A framework for reuse of user experience in web browsing, Behaviour and Information Technology 22 (2) (2003) 79.
[30] R. Srikant, R. Agrawal, Mining sequential patterns: generalizations and performance improvements, in: Proceedings of the 5th International Extending Database Technology, 1996, pp. 3–17.
[31] Z. Stejic, Y. Takamn, K. Hirota, Genetic algorithm-based relevance feedback for image retrieval using local similarity patterns, Information Processing and Management 39 (1) (2003) 1–23.
[32] C.Y. Tsai, Y.C. Shieh, A change detection method for sequential patterns, Decision Support Systems 46 (2009) 501–511.
[33] S.Y. Wu, Y.L. Chen, Mining nonambiguous temporal patterns for interval-base events, IEEE Transactions on Knowledge and Data Engineering 19 (6) (2007) 742–758.
[34] Y. Xiao, M.H. Dunham, Efficient mining of traversal patterns, Data and Knowledge Engineering 39 (2) (2001) 191–214.
[35] C.C. Yu, Y.L. Chen, Mining sequential patterns from multi-dimensional sequence data, IEEE Transactions on Knowledge and Data Engineering 17 (1) (2005) 136–140.
[36] M.J. Zaki, SPADE: an efficient algorithm for mining frequent sequences, Machine Learning Journal 42 (1/2) (2001) 31–60.
[37] M. Zhang, B. Kao, D. Cheung, C.L. Yip, Efficient algorithms for incremental update of frequent sequences, in: Proceedings of the 6th Pacific-Asia Conference on Knowledge Discovery and Data Mining, 2002, pp. 186–197.

Ya-Han Hu is currently an Assistant Professor of Department of Information Management at National Chung Cheng University, Taiwan. He received the Ph.D. degree in Information Management from National Central University of Taiwan in 2007. His current research interests include data mining and knowledge discovery, decision support systems, and EC technologies. His research has appeared in Decision Support Systems, and Journal of Information Science.

Tony Cheng-Kui Huang received the Ph.D. degree in Information Management from National Central University of Taiwan in 2006.

He is an Assistant Professor in the Department of Business Administration at National Chung Cheng University of Taiwan. His current research interests include data mining in business, decision support systems, information management, and soft computing. He has published papers in IEEE Transactions on Systems, Man and Cybernetics, Part B, Data & Knowledge Engineering, Fuzzy Sets and Systems, Expert Systems with Applications, and Computers and Education.

Hui-Ru Yang received her master's degree from National Central University of Taiwan. Her research interests are in data mining and information management.

Yen-Liang Chen is Professor of Information Management at National Central University of Taiwan. He received his Ph.D. degree in computer science from National Tsing Hua University, Hsinchu, Taiwan. His current research interests include data modeling, data mining, data warehousing and operations research. He has published papers in Operations Research, IEEE Transaction on Software Engineering, IEEE Transaction on Knowledge and Data Engineering, Decision Support Systems, Computers & OR, European Journal of Operational Research, Expert Systems with Applications, Information and Management, Information Processing Letters, Information Systems, Journal of Operational Research Society, and Transportation Research – Part B.