Mining Time-Interval Sequential Patterns Using Clustering Analysis

Hao-En Chueh[1], Nancy P. Lin[2]
[1]*Department of Information Management, Yuanpei University*
[2]*Department of Computer Sciences and Information Engineering, Tamkang University*
[1]*hechueh@mail.ypu.edu.tw,* [2]*nancylin@mail.tku.edu.tw*

Abstract*-A time-interval sequential pattern is a sequential pattern with the information about the time intervals between itemsets. Most algorithms of mining time-interval sequential patterns find the time intervals between itemsets by predefining some non-overlap time partitions, however, a predefined set of non-overlap time partitions cannot be suitable for every pair of successive itemsets. Therefore, in this paper, we present a new algorithm to mine time-interval sequential patterns without defining any time partitions in advance. The algorithm first adopts the clustering analysis to automatically generate the suitable time partitions for frequent occurring pairs of successive itemsets, and then uses these generated time partitions to extend typical algorithms to discover the time-interval sequential patterns. Our result of experiment verifies that this algorithm outperforms than the algorithms which use some predefined and non-overlap time partitions.*

Keywords: Data Mining, Sequential Pattern, Time Interval, Clustering Analysis.

1. Introduction

Data mining is the procedure of discovering hidden, useful, previously unknown information from databases. The existing techniques of data mining include association analysis, classification, clustering, etc, and one of the important techniques is mining sequential patterns. Mining sequential patterns first introduced by Agrawal and Srikant (1995) is the task of finding frequently occurring patterns related to time or other sequences from a sequence database [1]. An example of a sequential pattern is "A customer who bought a digital camera will return to buy an extra memory card". Mining sequential patterns is widely used in the field of retail business to assist in making various marketing decisions since many transaction records are stored as sequence data [3,5,7].

Many algorithms of mining sequential patterns have been proposed [1,4,6,9], and most of these algorithms only focus on finding the order of the itemsets, but ignore the time intervals between successive itemsets. An example of a sequential pattern with time intervals between successive itemsets is "A customer who bought a digital camera will buy an extra memory card within one month". In business field, actually, a sequential pattern which includes the time intervals between successive itemsets is more valuable than a traditional sequential pattern without any time information, because the time intervals between successive itemsets can offer useful information for businesses to sell the appropriate products to their customers at the right time. To this end, some researches start to propose algorithms to discover the sequential patterns with time intervals between itemsets, called time-interval sequential pattern [2].

To find the time intervals between every pair of successive itemsets, most proposed algorithms of mining time-interval sequential patterns usually predefine some non-overlap time partitions, and assume that the time interval between each pair of successive itemsets can match one time partition of this predefined set, but a predefined set of non-overlap time partitions, in fact, cannot be suitable for every pair of successive itemsets. Generating the suitable time partitions for every pair of successive itemsets directly from the real data is more reasonable. Accordingly, in this paper, we present a new algorithm to mine time-interval sequential patterns without predefining any time partitions. This algorithm tries to use the clustering analysis first to automatically generate the suitable time partitions between frequent occurring pairs of successive itemsets, and then uses these generated time partitions to extend typical algorithms to discover the time-interval sequential patterns.

The rest of this paper is organized as follows. Some researches related to time-interval sequential patterns are reviewed in section 2. The proposed time-interval sequential patterns mining algorithm is presented in section 3. A simple experiment is displayed in section 4. The conclusions are given

in section 5.

2. Time-Interval Sequential Patterns

The problem of sequential patterns mining which can be described as the task of discovering frequently occurring ordered patterns from a given sequence database was first introduced by Agrawal and Srikant in the mid 1990s [1].

A sequence is an ordered list of itemsets. Let $I = \{ i_1, i_2, ..., i_m \}$ be a set of items, $S = < s_1, s_2, ..., s_k >$ is a sequence, where $s_i \subseteq I$ is called an itemset. Length of a sequence means the number of itemsets in the sequence, and a sequence contains k itemsets is called a k-sequence. The support of a sequence S is denoted by supp (S) and means the percentage of total number of records containing sequence S in the sequence database. If supp (S) is greater than or equal to a predefined threshold, called minimal support, than sequence S is considered as a frequent sequence and called a sequential pattern.

Many algorithms have been proposed to mine sequential pattern [1,4,6,9], and most algorithms only focus on finding the frequently occurring order of the itemsets, but ignore the time intervals between itemsets. The time intervals between itemsets, in fact, can offer useful information for businesses to sell the appropriate products to their customers at the right time. Due to the value of the time intervals between itemsets, some researchers start to propose algorithms to mine various sequential patterns with time information between itemsets recently [2,8,10,11].

Srikant et al. [10] utilize three restrictions, the maximum interval $(max-interval)$, the minimum interval $(min-interval)$, and the time window size $(window-size)$ to find sequential patterns related to time intervals, and the discovered pattern is like $((A,B),(C,D))$, where (A,B) and (C,D) are called subsequences of $((A,B),(C,D))$. The $max-interval$ and the $min-interval$ are respectively used to indicate the maximal and minimal interval within subsequence. The $window-size$ is used to indicate the interval among subsequences. For example, let the $max-interval$ is set to 10 hours, the $min-interval$ is set to 3 hours, and $window-size$ is set to 24 hours. That is, the time interval between A and B lies in [2,10], the time interval between C and D also lies in [2,10], and the time interval between (A,B) and (C,D) lies in [0,24].

Mannila et al. [8] use a window width (win) to find frequent episodes in sequences of events, and the discovered episode is like (A,B,C). Assume that the win is set to 3 days, then the episode (A,B,C) means that, in 3 days, A occurs first, B follows, and C happens finally.

Wu et al. [11] utilize a window (d) as well to find the sequential pattern likes (A,B,C), such that, in a sequential pattern, the interval between adjacent events is within the window d. Assume d is set to 5 hours, then the discovered pattern (A,B,C) means that A occurs first, B follows, and C happens finally and the interval between A and B, and between B and C is within 5 hours.

Chen et al. [2] use a predefined set of non-overlap time partitions to discover potential time-interval sequential patterns, and the discovered pattern is like (A,I_0,B,I_2,C), where I_0, I_2 belong to the non-overlap set of time partitions. Assume that, in the set of time partitions, I_0 denotes the time interval t satisfying $0 \le t \le 1$ day; I_2 denotes the time interval t satisfying $3 < t \le 7$ days, and then the pattern (A,I_0,B,I_2,C) means that A, B and C happen in this order, and the interval between A and B is within 1 day , and the interval between B and lies between 3 days and 7 days.

Although these preceding discussion researches can discover the sequential patterns with the time information between itemsets by using a or some predefined and fixed time partitions, the patterns whose intervals between itemsets lie outside these predefined time ranges cannot be found yet.

Therefore, this work presents a new algorithm to mine time-interval sequential patterns without defining any time partitions in advance, and the main concept of this algorithm is to generate the suitable time partitions directly from the real data. This algorithm first uses the clustering analysis to automatically generate the suitable time partitions for frequent occurring pairs of successive itemsets, and then adopts these generated time partitions to extend typical algorithms to mine the sequential patterns with time intervals between every pair of successive itemsets.

Details of the proposed time-interval sequential mining patterns algorithm are presented in the following section.

3. The Proposed Algorithm

The proposed time-interval sequential patterns mining algorithm is presented in this section. This

algorithm uses clustering analysis to automatically generate the suitable time partitions directly from the real data for the frequent occurring pairs of successive itemsets, and then uses these generated time partitions to extend the typical algorithms to discover the sequential patterns with time intervals between every pair of successive itemsets. Steps of the algorithm are described as follows.

Notation:

$I = \{ i_1, i_2, ..., i_m \}$: The set of items.

$S_i = < s_1, s_2, ..., s_n >$: A sequence, where each $s_k \subseteq I$.

$D = \{ S_1, S_2, ..., S_k \}$: The set of sequences.

$\text{supp}(S_i)$: The support of S_i .

$min - supp$: The minimal support threshold.

CS_k : The set of candidate k-sequences.

FS_k : The set of frequent k-sequences.

$CTIS_k$: The set of candidate time-interval k-sequences.

$FTIS_k$: The set of frequent time-interval k-sequences.

Algorithm:

Step 1: Producing FS_1 . Each itemset can be regarded as a candidate 1-sequence. A candidate 1-sequence whose support is greater than or equal to $min - supp$ is a frequent 1-sequence, and the set of all frequent 1-sequences is FS_1 .

Step 2: Producing CS_2 . For any two frequent 1-sequences s_1 and s_2, where $s_1, s_2 \in FS_1$ and $s_1 \neq s_2$, then we can generate 2 candidate 2-sequences $< s_1, s_2 >$ and $< s_2, s_1 >$ that belong to CS_2 .

Step 3: Producing FS_2 . A candidate 2-sequence whose support is greater than or equal to $min - supp$ is a frequent 2-sequence, and the set of all frequent 2-sequences is FS_2 .

Step 4: Finding the set of all the suitable time partitions for each frequent 2-sequences of FS_2 . For any frequent 2-sequence of $FS_2 < s_p, s_q >$, all the time intervals between s_p and s_q appears in D are listed in increasing order, then the following clustering steps are used to find the suitable time partitions of all these intervals.

Step 4(a): Assume that $T(1, z) = [t_1, t_2, \cdots, t_z]$ is the increasingly ordered list of the time intervals of $< s_p, s_q >$. Let $T < s_p, s_q >= \{ T(1, z) \}$ be the set of all the suitable time partitions of $< s_p, s_q >$. The first step of the clustering analysis is to find the maximal difference between two adjacent intervals from all partitions of $T < s_p, s_q >$, and then divide the partition with the maximal difference into 2 partitions. At the beginning, only one partition is in $T < s_p, s_q >$, therefore, the chosen partition is $T(1, z)$. Assume that the different between t_i and t_{i+1} is maximal, than $T(1, z)$ is divided into $T(1, i)$ and $T(i+1, z)$, where $T(1, i) = [t_1, \cdots, t_i]$, $T(i+1, z) = [t_{i+1}, \cdots, t_z]$.

Step 4(b): The second step is to calculate the support of $< s_p, s_q >$ that respectively includes time intervals within these two partitions. If the support of $< s_p, s_q >$ that includes time intervals within $T(1, i)$ is greater than or equal to $min - supp$, $T(1, i)$ is a suitable time partition of $< s_p, s_q >$, and then $T(1, i)$ is reserved, otherwise $T(1, i)$ is deleted. Similarly, if the support of $< s_p, s_q >$ that includes time intervals within $T(i+1, z)$ is greater than or equal to $min - supp$, $T(i+1, z)$ is a suitable time partition, and then $T(1, i)$ is reserved, otherwise $T(1, i)$ is deleted. Thus, the chosen partition is placed by the reserved partitions. If no partition is reserved, this chosen partition is considered as non-dividable. If all of the differences between two adjacent intervals in a partition are equal, this partition is considered as non-dividable as well.

Step 4(c): Repeating step 4(a) and step 4(b), until all time partitions in $T < s_p, s_q >$ are non-dividable.

Step 5: Producing $FTIS_2$. Each 2-sequence of FS_2 is extended by all its suitable time partitions to form $FTIS_2$. Let $T < s_p, s_q >= \{ T^1, T^2, \cdots, T^R \}$ is the set of all the suitable time partitions of the sequence $< s_p, s_q >$, then $T < s_p, T^i, s_q >, i = 1 \cdots R$, is a frequent time-interval 2-sequences.

Step 6: Producing $CTIS_k, k \geq 3$. For any two frequent time-interval (k-1)-sequences S_1 and S_2, where $S_1 = < s_{1,1}, T_{1,1}, s_{1,2}, \cdots, s_{1,k-2}, T_{1,k-2}, s_{1,k-1} >$, $S_2 = < s_{2,1}, T_{2,1}, s_{2,2}, \cdots, s_{2,k-2}, T_{2,k-2}, s_{2,k-1} > \in FTIS_{k-1}$; $s_{1,2} = s_{2,1}, s_{1,3} = s_{2,2}, \cdots, s_{1,k-1} = s_{2,k-2}$; $T_{1,2} = T_{2,1}, T_{1,3} = T_{2,2}, \cdots, T_{1,k-2} = T_{2,k-3}$, then we can generate a candidate time-interval k-sequences $S_1 = < s_{1,1}, T_{1,1}, s_{1,2}, \cdots\cdots, s_{1,k-2}, T_{1,k-2}, s_{1,k-1}, T_{2,k-2}, s_{2,k-1} >$.

Step 7: Producing $FTIS_k, k \geq 3$. A candidate

time-interval k-sequence whose support is greater than or equal to $min-supp$ is a frequent time-interval k-sequence, and the set of all frequent time-interval k-sequences is $FTIS_k$.

Step 8: Repeating step 6 and step 7, until no next $CTIS_k$ can be generated.

According to these steps, an experiment using a simple sequence database will be displayed in the next section.

4. Experiment

In this section, we use the sequence database shown as in Table 1 to discover the time-interval sequential patterns. In Table 1, *Id* denotes the record number of a sequence, and each sequence is represented as $<(s_1,t_1),(s_2,t_2),\cdots,(s_n,t_n)>$, where s_i denotes an itemset, and t_i denotes the time stamp that s_i occurs; $min-supp$ is set as 0.3.

Table 1: A sequence database.

Id	Sequence
01	(s_5,8), (s_4,15), (s_6,20)
02	(s_1,2), (s_3,7), (s_2,11), (s_6,18)
03	(s_2,3), (s_1,4), (s_3,7), (s_6,16), (s_7,19)
04	(s_1,2), (s_2,8), (s_6,10), (s_7,15)
05	(s_5,4), (s_6,16), (s_1,20), (s_3,24)
06	(s_7,7), (s_1,13), (s_5,18), (s_2,25), (s_6,28)
07	(s_5,4), (s_1,8), (s_3,12), (s_6,16), (s_7,20)
08	(s_1,3), (s_5,6), (s_2,9), (s_4,18), (s_6,21)
09	(s_2,5), (s_1,10), (s_3,15), (s_6,20), (s_7,25)
10	(s_6,2), (s_7,8), (s_5,12), (s_2,17)

First, we need to calculate the supports of all itemsets to produce FS_1. Supports of all itemsets are shown in Table 2. Here, we can obtain $FS_1 = \{s_1, s_2, s_3, s_5, s_6, s_7\}$.

Table 2: Supports of itemsets.

itemset	supp
s_1	0.8
s_2	0.7
s_3	0.5
s_4	0.2
s_5	0.6
s_6	1
s_7	0.6

Next, CS_2 is generated by jointing $FS_1 \times FS_1$; Supports of the sequences in CS_2 are calculated and shown in Table 3. Therefore, we obtain $FS_3 = \{<s_1,s_2>,<s_1,s_3>,<s_1,s_6>,<s_1,s_7>,<s_2,s_6>,<s_2,s_7>,<s_3,s_6>,<s_3,s_7>,<s_5,s_2>,<s_5,s_6>,<s_6,s_7>\}$.

Table 3: Supports of sequences in CS_2.

CS_2	supp	CS_2	supp
<s_1,s_2>	0.4	<s_5,s_1>	0.2
<s_1,s_3>	0.5	<s_5,s_2>	0.3
<s_1,s_5>	0.2	<s_5,s_3>	0.2
<s_1,s_6>	0.7	<s_5,s_6>	0.5
<s_1,s_7>	0.4	<s_5,s_7>	0.1
<s_2,s_1>	0.2	<s_6,s_1>	0.1
<s_2,s_3>	0.2	<s_6,s_2>	0.1
<s_2,s_5>	0.0	<s_6,s_3>	0.1
<s_2,s_6>	0.6	<s_6,s_5>	0.1
<s_2,s_7>	0.3	<s_6,s_7>	0.5
<s_3,s_1>	0.0	<s_7,s_1>	0.1
<s_3,s_2>	0.1	<s_7,s_2>	0.2
<s_3,s_5>	0.0	<s_7,s_3>	0.0
<s_3,s_6>	0.4	<s_7,s_5>	0.2
<s_3,s_7>	0.3	<s_7,s_6>	0.1

For each frequent 2-sequence of FS_2, all its time intervals are recorded and listed in increasing order (Table 4).

Table 4: Time intervals of the sequences of FS_2.

FS_2	time intervals
<s_1,s_2>	T<s_1,s_2>= {6, 9, 12}
<s_1,s_3>	T<s_1,s_3>= {3, 4, 5}
<s_1,s_6>	T<s_1,s_6> = {8, 10, 12, 16, 18}
<s_1,s_7>	T<s_1,s_7>= {12, 13, 15 }
<s_2,s_6>	T<s_2,s_6>={2, 3, 7, 12, 13, 15}
<s_2,s_7>	T<s_2,s_7>= {7, 16, 20}
<s_3,s_6>	T<s_3,s_6>= {4, 5, 9, 11 }
<s_3,s_7>	T<s_3,s_7>= {8, 10, 12}
<s_5,s_2>	T<s_5,s_2> = {3, 5, 7}
<s_5,s_6>	T<s_5,s_6> = {10, 12, 15}
<s_6,s_7>	T<s_6,s_7> = {3, 4, 5, 6 }

According to the step 4 described in the above section, the set of all suitable time partitions for

each sequences of FS_2 are obtained as in Table 5. Next, each 2-sequence of FS_2 is extended by all its suitable time partitions to form $FTIS_2$ (Table 6).

Table 5: The suitable time partitions of sequences of FS_2.

FS_2	suitable time partitions
$<s_1,s_2>$	$T<s_1,s_2>= \{ T^1_{1,2}=[6,12] \}$
$<s_1,s_3>$	$T<s_1,s_3>= \{ T^1_{1,3}=[3,5] \}$
$<s_1,s_6>$	$T<s_1,s_6> = \{ T^1_{1,6}=[8,12],\ T^2_{1,6}=[16,18] \}$
$<s_1,s_7>$	$T<s_1,s_7>= \{ T^1_{1,7}=[12,15] \}$
$<s_2,s_6>$	$T<s_2,s_6>=\{ T^1_{2,6}=[2,7],\ T^2_{2,6}=[12,15] \}$
$<s_2,s_7>$	$T<s_2,s_7>= \{ T^1_{2,7}=[16,20] \}$
$<s_3,s_6>$	$T<s_3,s_6>= \{ T^1_{3,6}=[4,11] \}$
$<s_3,s_7>$	$T<s_3,s_7>= \{ T^1_{3,7}=[8,12] \}$
$<s_5,s_2>$	$T<s_5,s_2> = \{ T^1_{5,2}=[3,7] \}$
$<s_5,s_6>$	$T<s_5,s_6> = \{ T^1_{5,6}=[10,12] \}$
$<s_6,s_7>$	$T<s_6,s_7> = \{ T^1_{6,7}=[3,6] \}$

Table 6 : $FTIS_2$
$<s_1, T^1_{1,2}, s_2>$
$<s_1, T^1_{1,3}, s_3>$
$<s_1, T^1_{1,6}, s_6>, <s_1, T^2_{1,6}, s_6>$
$<s_1, T^1_{1,7}, s_7>$
$<s_2, T^1_{2,6}, s_6>, <s_2, T^2_{2,6}, s_6>$
$<s_2, T^1_{2,7}, s_7>$
$<s_3, T^1_{3,6}, s_6>$
$<s_3, T^1_{3,7}, s_7>$
$<s_5, T^1_{5,2}, s_2>$
$<s_5, T^1_{5,6}, s_6>$
$<s_6, T^1_{6,7}, s_7>$

$CTIS_3$, the set of candidate time-interval 3-sequences is generated by jointing $FTIS_2 \times FTIS_2$. Supports of the sequences of $CTIS_3$ are calculated and shown in Table 7. A candidate time-interval 3-sequence whose support is greater than or equal to $min-supp$ is called as a frequent time-interval 3- sequence. Therefore, we can obtain the set of all the frequent time-interval 3-sequences, $FTIS_4 = \{ <s_1, T^1_{1,2}, s_2, T^1_{2,6}, s_6>, <s_1, T^1_{1,3}, s_3, T^1_{3,6}, s_6>, <s_1, T^1_{1,3}, s_3, T^1_{3,7}, s_7>, <s_1, T^1_{1,6}, s_6, T^1_{6,7}, s_7>, <s_3, T^1_{3,6}, s_6, T^1_{6,7}, s_7> \}$.

Table 7: Supports of sequences of $CTIS_3$

$CTIS_3$	supp
$<s_1, T^1_{1,2}, s_2, T^1_{2,6}, s_6>$	0.3
$<s_1, T^1_{1,2}, s_2, T^2_{2,6}, s_6>$	0.1
$<s_1, T^1_{1,2}, s_2, T^1_{2,7}, s_7>$	0.1
$<s_1, T^1_{1,3}, s_3, T^1_{3,6}, s_6>$	0.4
$<s_1, T^1_{1,3}, s_3, T^1_{3,7}, s_7>$	0.3
$<s_1, T^1_{1,6}, s_6, T^1_{6,7}, s_7>$	0.4
$<s_2, T^1_{2,6}, s_6, T^1_{6,7}, s_7>$	0.1
$<s_2, T^2_{2,6}, s_6, T^1_{6,7}, s_7>$	0.2
$<s_3, T^1_{3,6}, s_6, T^1_{6,7}, s_7>$	0.3
$<s_5, T^1_{5,2}, s_2, T^1_{2,7}, s_7>$	0
$<s_5, T^1_{5,6}, s_6, T^1_{6,7}, s_7>$	0.1

The set of candidate time-interval 4-sequences, $CTIS_4$, is generated by jointing $FTIS_3 \times FTIS_3$. Here, only one sequence, $<s_1, T^1_{1,3}, s_3, T^1_{3,6}, s_6, T^1_{6,7}, s_7>$, is generated. The support of the sequence $<s_1, T^1_{1,3}, s_3, T^1_{3,6}, s_6, T^1_{6,7}, s_7>$ is 0.3, thus $<s_1, T^1_{1,3}, s_3, T^1_{3,6}, s_6, T^1_{6,7}, s_7>$ is a frequent time-interval 4-sequences, and we obtain $FTIS_4 = \{<s_1, T^1_{1,3}, s_3, T^1_{3,6}, s_6, T^1_{6,7}, s_7>\}$. Because no next $CTIS_5$ can be generated, the algorithm stops here.

According to the experimental results, we can clearly see that the suitable time partitions for every pair of successive itemsets are different and overlap, therefore, for every pair of successive itemsets, it is more reasonable to generate the suitable time partitions directly from the real data when mining time-interval sequential patterns.

5. Conclusions

A sequential pattern with the time intervals between successive itemsets is more valuable than a traditional sequential pattern without any time information. Most existing time-interval sequential pattern mining algorithms reveal the time-interval between itemsets by predefining some non-overlap time partitions, but a predefined set of non-overlap time partitions, in fact, cannot be suitable for every

pair of successive itemsets.

In this paper, we present a new algorithm to mine time-interval sequential patterns without pre-defining any time partitions. This algorithm use the clustering analysis to automatically generate the suitable time partitions between frequent occurring pairs of successive itemsets, and then uses these generated time partitions to extend typical algorithms to discover the time-interval sequential patterns. The result of our experiment verifies that the concept of our algorithm is more reasonable than these algorithms which use some predefined and non-overlap time partitions.

References

[1] R. Agrawal, and R. Srikant, "Mining sequential patterns," *Proceedings of the International Conference on Data Engineering*, pp. 3–14, 1995.

[2] Y. L. Chen, M. C. Chiang, and M. T. Ko, "Discovering time-interval sequential patterns in sequence databases," *Expert Systems with Applications*, vol.25, no. 3, pp. 343–354, 2003.

[3] M. S. Chen, J. Han, and P. S. Yu, "Data mining: An overview from a database perspective," *IEEE Transactions on Knowledge and Data Engineering*, vol.8, no.6, pp. 866–883, 1996.

[4] M. S. Chen, J. S. Park, and P. S. Yu, "Efficient data mining for path traversal patterns," *IEEE Transactions on Knowledge and Data Engineering*, vol. 10, no. 2, pp. 209–221, 1998.

[5] M. H. Dunham, *Data mining, Introductory and Advanced Topics*, Pearson Education Inc., 2003.

[6] J. Han, G. Dong, and Y. Yin, "Efficient mining of partial periodic patterns in time series database," *Proceedings of 1999 International Conference on Data Engineering*, pp. 106–115, 1999.

[7] J. Han, and M. Kamber, *Data mining: Concepts and Techniques*, Academic Press, 2001.

[8] H. Mannila, H. Toivonen, and A. Inkeri Verkamo, "Discovery of frequent episodes in event sequences," *Data Mining and Knowledge Discovery*, vol. 1, no.3, pp. 259-289, 1997.

[9] J. Pei, J. Han, H. Pinto, Q, Chen, U. Dayal, and M.-C. Hsu, "PrefixSpan: Mining sequential patterns efficiently by prefix-projected pattern growth," *Proceedings of 2001 International Conference on Data Engineering*, pp. 215–224, 2001.

[10] R. Srikant, and R. Agrawal, "Mining sequential patterns: Generalizations and performance improvements," *Proceedings of the 5th International Conference on Extending Database Technology*, pp. 3–17, 1996.

[11] P. H. Wu, W. C. Peng, and M. S. Chen, "Mining sequential alarm patterns in a telecommunication database," *Proceedings of Workshop on Databases in Telecommunic-ations* (VLDB 2001), pp. 37-51, 2001.