IEICE TRANS. INF. & SYST., VOL.E96–D, NO.8 AUGUST 2013

PAPER

Finding Interesting Sequential Patterns in Sequence Data Streams via a Time-Interval Weighting Approach

Joong Hyuk CHANG[†], *Nonmember and* **Nam Hun PARK**[†a)], *Member*

SUMMARY The mining problem over data streams has recently been attracting considerable attention thanks to the usefulness of data mining in various application fields of information science, and sequence data streams are so common in daily life. Therefore, a study on mining sequential patterns over sequence data streams can give valuable results for wide use in various application fields. This paper proposes a new framework for mining novel interesting sequential patterns over a sequence data stream and a mining method based on the framework. Assuming that a sequence with small time-intervals between its data elements is more valuable than others with large time-intervals, the novel interesting sequential pattern is defined and found by analyzing the time-intervals of data elements in a sequence as well as their orders. The proposed framework is capable of obtaining more interesting sequential patterns over sequence data streams whose data elements are highly correlated in terms of generation time.

key words: *time-interval weight, weighted sequential pattern, time-interval sequential pattern, time-interval sequence data stream, data stream*

1. Introduction

Since data has been increasingly taking the form of continuous data streams rather than finite stored data sets in various application fields, the database research community has begun focusing its attention on processing over data streams [1]. The data stream is a massive unbounded sequence of data elements continuously generated at a rapid rate [2]. In data stream processing, each data element should be examined at most once to analyze the data stream, and the memory usage for data stream analysis should be restricted finitely although new data elements are continuously generated in a data stream. Moreover, newly generated data elements should be processed as fast as possible to produce the up-to-date analysis result of a data stream so that it can be instantly utilized upon request [2]. To satisfy these requirements, data stream processing sacrifices the correctness of its analysis result by allowing some error.

Considering the changes in the form of data generated in realworld application fields, many research have been actively performed to find various kinds of knowledge embedded in data streams. They mainly focus on efficient mining of frequent itemsets [3]–[5] and sequential patterns [6], [7] over data streams, which have been proven to be useful in conventional data mining for a finite data set. In addition, mining algorithms [3], [8] have also been proposed to efficiently reflect the changes of data streams over time into their mining results. However, they have been targeted on finding naively interesting patterns such as frequent patterns and simple sequential patterns, which are found intuitively, taking no interest in mining novel interesting patterns that express the characteristics of target data streams better. In recent computing application fields generating data streams, a data stream generally takes a more complex form with additional information such as generation times of data elements. Therefore, it can be a valuable research topic in the field of mining data streams to define novel interesting patterns and develop a mining method finding the novel patterns, which will be effectively used to analyze recent data streams.

Sequential pattern mining aims to discover interesting sequential patterns in a sequence database, and it is one of the essential data mining tasks widely used in various application fields such as Web access pattern analysis, customer purchase pattern analysis, and DNA sequence analysis. In many of the previous researches on sequential pattern mining problems, sequential patterns and items in a sequential pattern have been considered uniformly. However, they have different weights in real world applications, and thus more interesting sequential patterns can be found when their different weights are considered in sequential pattern mining. Based on this observation, weighted sequential pattern mining [9]–[12] has recently been proposed and actively studied. In weighted sequential pattern mining, the weight of information is used in finding interesting sequential patterns, which is derived from its quantitative information and value in a real world application. For example, in a retail database, the quantum and price of an item being sold can be considered as its weight.

For a sequence or a sequential pattern, not only the generation order of data elements but also their generation times and time-intervals are important because they can help to get more valuable sequential patterns. In [13] and [14], several sequential pattern mining algorithms have been presented which consider a time-interval between two successive items in a sequential pattern. However, they simply consider a time-interval between two successive data elements as an item. If the importance of sequences in a sequence database is differentiated based on the time-intervals in the sequences, more interesting sequential patterns can be found.

The following is an example of a set of time-interval sequences generated from the purchasing history in a computer store.

Manuscript received November 8, 2012.
Manuscript revised March 19, 2013.
[†]The authors are with Anyang University, Korea.
a) E-mail: nmhnpark@anyang.ac.kr (Corresponding author)
DOI: 10.1587/transinf.E96.D.1734

[*Customer_A*]
a laser printer → (1 month after) → a scanner → (1 month after) → a CD burner
[*Customer_B*]
a laser printer → (6 months after) → a scanner → (3 months after) → a CD burner

The sequences consist of the same items and their orders are the same in both customers, but the time-intervals between the items are different. Therefore, they may appear to be the same if only the order of items is considered, but they would be totally different if the time-intervals were itemized. However, it is better to consider them as having the same sequence with a different weight, since it is closer to a real world situation. In the above example, the sequence by *customer_A* can be considered more important than the sequence by *customer_B* since the former has relatively smaller time-intervals than the latter.

In general a sequence with small time-intervals between its data elements is more valuable than others with large time-intervals. Motivated by this observation, this paper proposes a new framework for mining novel interesting sequential patterns over time-interval sequence data streams and a mining method based on the new framework. First, a technique to get the weight of a sequence in a time-interval sequence data stream is presented, which is derived from the time-intervals of items in the sequence. Based on the weight of a sequence, a novel interesting sequential pattern of a time-interval weighted sequential pattern is defined, and a framework for finding the patterns in a time-interval sequence data stream is presented. In addition, adapting the proposed framework to the conventional method of mining sequential patterns over a data stream, this paper proposes a mining method of novel interesting sequential patterns over a time-interval sequence data stream, which can find time-interval sequential patterns over the data stream in a short time with a small memory.

The rest of this paper is organized as follows: Section 2 gives a brief summary of related work including our previous work on mining sequential patterns over data streams, and the definition of a time-interval sequence data stream and a problem discussed in this paper are described in Sect. 3. Section 4 presents novel interesting sequential patterns for mining time-interval sequence data streams, which are based on time-intervals of data elements in a data stream. A mining method to get the novel interesting sequential patterns over a time-interval sequence data stream is also presented in the section. In Sect. 5, the effectiveness of the novel interesting sequential patterns and the efficiency of the proposed mining method are verified through a series of experiments. Finally, Sect. 6 concludes this paper.

2. Related Work

To extract different types of knowledge embedded in data streams, various algorithms [3]–[8] have been actively proposed. These algorithms mainly target on minimizing the memory usage and processing time to get their mining results in data streams. For mining sequential patterns, the number of patterns to be considered in a mining process is very large, and it takes quite a long time to get its mining result. Therefore, it is very hard to do mining sequential patterns efficiently in data streams whose data elements are continuously generated at a rapid rate. Previous researches [6], [7] on mining sequential patterns over data streams have focused on getting their mining results efficiently in a short time and a small memory using basic information such as the frequency of a data element, but they have not been able to support getting interesting sequential patterns by considering additional information such as the generation time of a data element.

In particular, to find frequent sequential patterns efficiently over a data stream, the eISeq [6] method has been proposed in our previous work. In the eISeq method, the effect of the information in an old sequence on the current mining result is diminished by decaying the old occurrence count of a sequential pattern as time goes by. In other words sequences are differentiated by their generation times. However, to concentrate on the mining problem of interesting sequential patterns in a time-interval sequence data stream, they are not differentiated by the generation time in this paper as in most conventional data stream mining approaches.

For mining sequential patterns in sequence databases, various algorithms for mining simple sequential patterns [15]–[19] and weighted sequential patterns [9]–[11] have been actively proposed so far. In most of the weighted sequential pattern mining algorithms, they usually require pre-assigned weights, which are generally derived from the quantitative information of items and its importance in a real world application. In addition, there have been several studies on mining sequential patterns considering time-interval and gap information between items in a sequence. In recent, a mining method for finding time-interval weighted sequential patterns is proposed [12], but it focuses on a stored finite data set. So that, it cannot be used for mining data streams efficiently.

Chen et al. [13], [14] have proposed sequential pattern mining algorithms for a sequence database with single items and their corresponding time-intervals, but the algorithm just considers time-interval information between two successive items as an item. Pei et al. [20] and Ji et al. [21] have proposed constrained sequential pattern mining algorithms, which use time-interval and gap information as a constraint. In these algorithms, time-interval and gap information are used only to confine the mining result of sequential patterns. Consequently, they are unable to support getting a mining result of weighted sequential patterns.

3. Preliminaries

3.1 A Time-Interval Sequence Data Stream

Conventional sequential pattern mining considers the order of data elements of a sequence in general, so that a sequence

in a sequential data stream is represented as an ordered list of data elements [6]. However, a time-interval sequence data stream discussed in this paper has generation time information for each data element in the data stream, and is defined as follows:

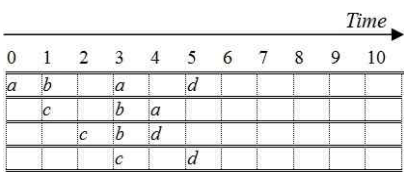
i) Let $I = \{i_1, i_2, \ldots, i_n\}$ be a set of current **items**, which have been used as a unit of information of an application domain.

ii) A **sequence** S is an ordered list of items and its **time stamp list** $TSL(S)$ is an ordered list of corresponding time stamps of the items, which stand for the time when the items occur. They are denoted as $S = <s_1, s_2, \ldots, s_l>$ and $TSL(S) = <t_1, t_2, \ldots, t_l>$, respectively, where s_j is an item and t_j is the time stamp of s_j for $1 \leq j \leq l$. In addition, the relationship $t_{j-1} \leq t_j$ for $2 \leq j \leq l$ is satisfied. In a sequence, if items occur at the same time, they are ordered alphabetically. The *length* of S, $|S|$, is the number of items that form the sequence, and a sequence with n items is called an *n-sequence*. A sequence $\alpha = <a_1, a_2, \ldots, a_n>$ is called a *subsequence* of another sequence $\beta = <b_1, b_2, \ldots, b_m>$, and β is a *super-sequence* of α, if there exist integers $1 \leq j_1 < j_2 < \ldots < j_n \leq m$ such that $a_1 = b_{j1}$, $a_2 = b_{j2}$, $\ldots$, $a_n = b_{jn}$. Each sequence has a unique sequence identifier *SID*. A sequence generated at the k^{th} turn is denoted by S_k and its transaction identifier *SID* is k.

iii) When a new sequence S_k is generated, the current **time-interval sequence data stream** $TiDS_k$ is composed of all sequences that have ever been generated so far, i.e., $TiDS_k = <S_1, S_2, \ldots, S_k>$, and the *total number of sequences* in $TiDS_k$ is called its size and denoted by $|TiDS|_k$. In the rest of this paper, a sequence data stream means a time-interval sequence data stream, if not specified otherwise.

Figure 1 shows an example time-interval sequence data stream, $TiDS_4$, having 4 sequences subsequently generated, and their *SID*s are 1, 2, 3, and 4, respectively.

A sequence is represented as an ordered list of items in this paper, while it is represented as an ordered list of itemsets in practice. However, the new representation of sequences described herein is in fact a typical one. A sequence in the previous format can be transformed to the new format by sorting all the items first by time and then alphabetically. Likewise, a sequence in the new format can be transformed into the traditional format by first combining items that occur at the same time into an item set and then sorting these item sets by time [13]. Moreover, a sequence in the new format itself is capable of capturing some of the most important and popular sequences, such as Web-logs, DNA sequences, and documents [21].



SID	Sequence	Time stamp list
1	<a, b, a, d>	<0, 1, 3, 5>
2	<c, b, a>	<1, 3, 4>
3	<c, b, d>	<2, 3, 4>
4	<c, d>	<3,5>

Fig. 1 A time-interval sequence data stream $TiDS_4$.

3.2 Problem Statements

In general, when a sequence S_k is currently generated in a sequence data stream $TiDS_k$, the current count $C_k(s)$ of a sequential pattern s is the number of sequences that contain the sequential pattern among the k sequences. Likewise, the current support $Supp_k(s)$ of a sequential pattern s is the ratio of its current count $C_k(s)$ over $|TiDS|_k$.

In order to find interesting sequential patterns over a sequence data stream, a term of weighted sequential patterns is used in this paper. That is, in sequence data stream mining, the weight of a sequence in a sequence data stream is considered, and it is derived from the time-intervals of items and used to get the count of a sequence and the size of a sequence data stream. In mining of weighted sequential patterns on the basis of a time-interval of an item, the weighted count of a sequential pattern A in a sequence data stream $TiDS_k$ is the sum of weights of sequences in $TiDS_k$ containing A. Likewise, its weighted support is the ratio of its weighted count over the sum of the weights of all sequences in $TiDS_k$.

When a sequence S_k is currently generated in a sequence data stream $TiDS_k$, a sequential pattern A is called an interesting sequential pattern in $TiDS_k$ for a given support threshold $minSupp$ ($0 < minSupp \leq 1$), if the weighted support of A is no less than the support threshold. Accordingly, for a given sequence data stream and a support threshold, the mining problem of interesting sequential patterns over the data stream is to find the complete set of all interesting sequential patterns whose weighted supports are no less than the threshold.

4. Interesting Sequential Pattern Mining over a Time-Interval Sequence Data Stream

A new term of interesting sequential patterns is presented in this section, which can be effectively used in analyzing time-interval sequential data streams. It is based on the weight of a sequence derived from the time-intervals between data elements of the sequence. In general, not only the generation order of data elements but also their generation times and time-intervals among them are important in sequential pattern mining. Therefore, the novel interesting sequential patterns can be usefully applied to analyzing the characteristics of real world applications generating data as a form of

sequence data streams. In addition, a mining method to find the novel interesting sequential patterns over a time-interval data stream is also presented.

4.1 Time-Interval Weight of a Sequence

For mining sequential patterns over a time-interval sequence data stream, the weight of a sequence in the data stream can be computed from the generation times of data elements in the sequence, which means the relative importance of the sequence in the sequence data stream. It is called the time-interval weight of the sequence.

To get the time-interval weight of a sequence in a sequence data stream, first the time-intervals in the sequence are found from the time stamps of items in the sequence. For a sequence $S = <s_1, s_2, \ldots, s_l>$ having its time stamp list $TSL(S) = <t_1, t_2, \ldots, t_l>$ in a sequence data stream, there exist $\{l \times (l-1)\}/2$ pairs of items in the sequence because it consists of l items, and the ***time-interval*** between two items s_i and s_j $(1 \le i < j \le l)$ in the sequence, i.e., TI_{ij}, is defined as follows:

$$TI_{ij} = t_j - t_i$$

For a sequence whose *SID* is 1 as shown in Fig. 1, there can be $4 \times (4-1)/2$ pairs of items, and the time-interval of each pair can have the following possible pairs of items as shown in Table 1.

The time-interval between a pair of items is a positive value with no limitation. Therefore, to fairly enumerate the time-intervals of different pairs of items in a sequence data stream, they need to be normalized. For this purpose, the time-interval weight for each pair of items in a sequence is found on the basis of its time-interval, and defined as in Definition 1.

Definition 1 Time-interval weight
Let u $(u > 0)$ be the size of **unit time** and δ $(0 < \delta < 1)$ be a **base number** to determine the amount of weight reduction per unit time u, for a sequence $S = <s_1, s_2, \ldots, s_l>$ and its time stamp list $TSL(S) = <t_1, t_2, \ldots, t_l>$, the time-interval weight of the time-interval TI_{ij} between two items s_i and s_j $(1 \le i < j \le l)$, i.e., $w(TI_{ij})$, is defined as follows:

i. General-scale weighting

$$w(TI_{ij}) = {}_{\delta}\left\lceil \frac{TI_{ij}}{u} \right\rceil = {}_{\delta}\left\lceil \frac{t_j - t_i}{u} \right\rceil$$

ii. Log-scale weighting:

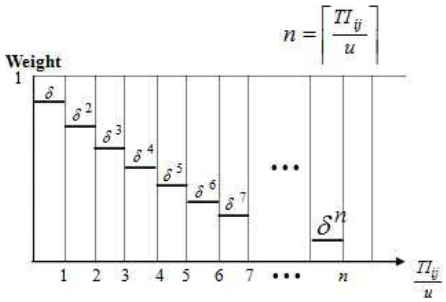
$$w(TI_{ij}) = {}_{\delta}\left\lceil \log_2\left(1 + \frac{TI_{ij}}{u}\right) \right\rceil = {}_{\delta}\left\lceil \log_2\left(1 + \frac{t_j - t_i}{u}\right) \right\rceil$$

■

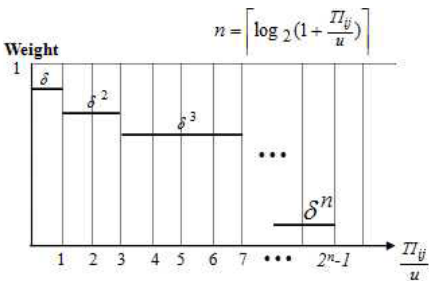
The smaller the values of δ and u are, the more sensitive a time-interval weight is to the increase of a time-interval. Among the two weighting functions shown in Definition 1, when the general-scale weighting function is applied, the time-interval weight of a pair of items is affected by its time-interval in general-scale. In other words, the weight decreases in general-scale as the time-interval increases as shown in Fig. 2 (a). In the case of the log-scale weighting function, the weight decreases in log-scale as the time-interval increases as shown in Fig. 2 (b). In the former case, all the pairs of items whose time-intervals, *TI*s, are in the range of $u \times i < TI \le u \times (i+1)$ have the same time-interval weight, while all the pairs of items whose time-intervals are in the range of $u \times 2^i < TI \le u \times 2^{(i+1)}$ have the same time-interval weight in the latter case, where u denotes the size of unit time and $i = 1, 2, 3, \ldots$.

Table 1 Possible pairs of items.

1st Item	*a*	*a*	*a*	*b*	*b*	*a*
2nd Item	*b*	*a*	*d*	*a*	*d*	*d*
Time-interval	1	3	5	2	4	2



(a) A general-scale weighting function



(b) A log-scale weighting function

Fig. 2 Time-interval weighting functions.

The time-interval weight of a sequence is computed from the time-intervals of pairs of items in the sequence. For a sequence $S = <s_1, s_2, \ldots, s_l>$ and its time stamp list $TSL(S) = <t_1, t_2, \ldots, t_l>$, the time-interval weight of the sequence is found as in Definition 2 considering the time-intervals in the sequence.

Definition 2 Time-interval weight of a sequence
For a sequence $S = <s_1, s_2, \ldots, s_l>$ and its time stamp list $TSL(S) = <t_1, t_2, \ldots, t_l>$, the time-interval weight of the sequence, i.e., $W(S)$, is defined as follows:

$$W(S) = \begin{cases} \frac{1}{N} \sum_{i=1}^{|S|-1} \sum_{j=i+1}^{|S|} w(TI_{ij}), \text{ where } N = \frac{|S|(|S|-1)}{2} & (l \geq 2) \\ 1 & (l = 1) \end{cases}$$

■

For a sequence S_1 in Fig. 1, the time-interval weight of the sequence, $W(S_1)$, is as follows:

$$\begin{aligned} N &= 4 \times (4-1)/2 = 6 \\ W(S_1) &= 1/6\{w(TI_{12}) + w(TI_{13}) + w(TI_{14}) + w(TI_{23}) \\ &\quad + w(TI_{24}) + w(TI_{34})\} \\ &= 1/6\{w(1) + w(3) + w(5) + w(2) + w(4) + w(2)\} \\ &= 1/6\{w(1) + w(2) \times 2 + w(3) + w(4) + w(5)\} \end{aligned}$$

Therefore, when the general-scale weighting function $w(TI) = \delta^{\lceil TI/u \rceil}$ with $\delta = 0.9$ and $u = 1$ is applied, the value of $W(S_1)$ is found to be 0.749. Under the same condition, the weights of the other sequences in Fig. 1 are found to be $W(S_2) = 0.813$, $W(S_3) = 0.870$, and $W(S_4) = 0.810$, respectively. In addition, the appearance of the sequence $<a, b>$ in S_3 is considered more important than that in S_1 because the time-interval weight of S_3 is greater than that of S_1.

4.2 Time-Interval Weighted Support of a Sequential Pattern

The sequential pattern evaluation by support has been generally based on simple counting. Contrary to the classical sequential pattern mining, however, this paper proposes a novel interesting sequential pattern of a *time-interval weighted sequential pattern* which is based on the *time-interval weighted support* of a sequential pattern. In this section, the evaluation process of the time-interval weighted support of a sequential pattern is presented in detail.

The time-interval weighted support of a sequential pattern in a sequence data stream is found by using a time-interval weight of a sequence described in Sect. 4.1. For a sequence data stream $TiDS_k$ consisting k sequences, the ***time-interval weighted support of a sequential pattern X*** in the sequence data stream, i.e., *TW-Supp*(X), is defined as follows:

$$TW\text{-}Supp(X) = \frac{\sum_{S:(X \subseteq S) \wedge (S \in TiDS_k)} W(S)}{\sum_{S:S \in TiDS_k} W(S)}$$

Accordingly, a novel interesting sequential pattern of a time-interval weighted sequential pattern can be defined. Given a support threshold *minSupp* ($0 < minSupp \leq 1$), a sequential pattern X is a ***time-interval weighted sequential pattern*** if *TW-Supp*(X) is no less than the threshold, i.e., $TW\text{-}Supp(X) \geq minSupp$.

Table 2 shows the supports of several sequential patterns derived from the sequence data stream in Fig. 1 when the general-scale weighting function $w(TI) = \delta^{\lceil TI/u \rceil}$ with $\delta = 0.9$ and $u = 1$ is applied. Three sequential patterns $<b, a>$, $<c, b>$, and $<c, d>$ have the same support in simple support counting, but the time-interval weighted support of $<b, a>$ is less than those of the others because it appears in the sequences whose time-interval weights are relatively smaller such as the sequence S_1. In addition, if the support threshold for mining sequential patterns on the data stream is set to 0.5, all of them can be interesting sequential patterns in the classical sequential pattern mining, but the sequence $<b, a>$ cannot be an interesting sequential pattern in mining time-interval weighted sequential patterns since its time-interval weighted support is less than the threshold.

Table 2 Change of supports (Simple support vs. Time-interval weighted support).

Sequential pattern	*Simple support*	*TW-support*
<b, a>	0.500	0.482
< c, b>	0.500	0.519
<c, d>	0.500	0.518

4.3 TWDS Method

For a sequence data stream, to find interesting sequential patterns of time-interval weighted sequential patterns over the data stream, a ***TWDS*** (**T**ime-interval **W**eighted sequential pattern mining over a sequence **D**ata **S**tream) method is presented in this section. Although it is based on the eISeq method, it also has an additional operation to get the time-interval weight of a sequence in a sequence data stream from the time-intervals of data elements in the sequence. In addition, a count updating operation and a sequential pattern insertion operation are merged into one operation in the *TWDS* method. It can help to reduce the number of times to traverse a monitoring tree, and then the processing time for each sequence can decrease. For a sequence data stream, when a sequence is newly generated in the sequence data stream, the time-interval weight of the sequence is first computed before a series of operations to process the sequence are performed. In every operation for the sequence such as a parameter updating operation, a count updating & sequential pattern insertion operation, and a time-interval weighted sequential pattern selection operation, the time-interval weight of the sequence is considered. For a sequence data stream, the time-interval sequential pattern selection operation is not necessary to be performed in every sequence, but it is performed only when the up-to-date set of time-interval weighted sequential patterns in the current sequence data stream is requested. A force-pruning operation is similar to that in the eISeq method, but a time-interval support is used to decide whether a sequential pattern, i.e., its corresponding node in a monitoring tree, should be pruned or not. Details of the *TWDS* method can be summarized as follows and presented in Fig. 3.

For a data stream and a given minimum support *minSupp*, the ***TWDS*** method finds the complete set of interesting sequential patterns of time-interval weighted sequential patterns over the data stream. The method examines each sequence in a sequence data stream one by one without any candidate generation. Among all the sequential patterns in each sequence of a sequence data stream, only those sequen-

Input: A time-interval sequence data stream $TiDS_k$; A support threshold *minSupp*; A significant support threshold $Supp_{sig}$;
Output: A complete set of interesting sequential patterns, i.e., time-interval weighted sequential patterns, ISP_k;

ML : A monitoring tree that maintains a set of significant sequential patterns

$ML = \emptyset$;
for each *new sequence* S_k in $TiDS_k$ **{**

// Getting the time-interval weight $W(S_k)$ of S_k=<s_1, s_2, …, s_l> from its time stamp list $TSL(S_k)$=<t_1, t_2, …, t_l>
$W(S_k) = \frac{1}{N} \sum_{i=1}^{l-1} \sum_{j=i+1}^{l} w(TI_{ij})$, where $N = \frac{l(l-1)}{2}$ and $TI_{ij} = t_j - t_i$;

// Parameter updating
The total number of sequences in the current sequence data stream is updated considering $W(S_k)$;

// Count updating & Sequential pattern insertion
for each *sequential pattern* $s \subseteq S_k$ **{**
if (its corresponding node with an entry (*cnt*, *cnt_r*, *tid, tid_r*) is in *ML*) **{**
// A corresponding node of *s* means the node that maintains its count information in *ML*
if (*tid*<*k*) **{**
The *cnt* and *tid* of the entry are updated subsequently considering $W(S_k)$;
if (*cnt* / $|TiDS|_k$) < $Supp_{sig}$
The corresponding entry is pruned from *ML*; // *s* is regarded as an insignificant one
if ($s \subseteq R(S_k)$) **&&** (*tid_r* < *k*) // $R(S_k)$ is a remaining-sequence of S_k
The *cnt_r* and *tid_r* of the entry are updated subsequently considering $W(S_k)$;
}
}
else {
If *s* is a significant sequential pattern, its corresponding node with an entry (*cnt*, *cnt_r*, *tid, tid_r*) is inserted into *ML*, and the values of *cnt*, *cnt_r*, *tid* and *tid_r* in the entry are initialized as described in [6] considering $W(S_k)$;
}
}

// Time-interval weighted sequential pattern selection
$ISP_k = \emptyset$;
for all sequential pattern *s* whose corresponding node is in *ML*
if *TW-support*(*s*) ≥ *minSupp*
$ISP_k = ISP_k \cup \{s\}$; // *s* is a time-interval weighted sequential pattern
}

Fig. 3 *TWDS* method.

tial patterns that should be monitored closely are maintained in the main memory in a lexicographic tree structure [22] called a *monitoring tree*. In the method, a sequential pattern is called a *significant sequential pattern* when its weighted support is greater than or equal to a predefined *significant support* $\boldsymbol{Supp}_{sig}$ ($0 < Supp_{sig} < minSupp$), and only those significant sequential patterns are maintained in the main memory.

Every node in a monitoring tree contains an item, and it denotes a sequential pattern composed of the items in the nodes of its path from the root. Each node maintains an entry (*cnt*, *cnt_r*, *tid, tid_r*) for its corresponding sequential pattern. The *cnt* is the count of the sequential pattern in the current data stream D_k. The *cnt_r* is the remaining count of the sequential pattern that represents the number of sequences whose remaining-sequences are a super-sequence of the sequential pattern in the current data stream D_k. For a sequence S_k, a *prefix-item* $P(S_k)$ and a *remaining-sequence* $R(S_k)$ of the sequence are defined as follows [6]: A prefix-item $P(S_k)$ is the first item of the sequence S_k, and a remaining-sequence $R(S_k)$ is a sub-sequence that is composed of all the items of S_k except $P(S_k)$. The *tid* denotes the sequence identifier of the latest sequence that is a super-sequence of the sequential pattern. The *tid_r* denotes the sequence identifier of the latest sequence whose remaining-sequence is a super-sequence of the sequential pattern. The other components are the same as those in the general lexicographic tree structure.

When a new sequence S_k is generated in the current data stream D_k, the following operations, except a selection operation of frequent sequential patterns and a force-pruning operation, are performed in sequence to reflect the

information of the new sequence S_k on a monitoring tree. A selection operation of frequent sequential patterns is performed only when the up-to-date set of frequent sequential patterns in the current data stream is requested, and a force-pruning operation is usually performed periodically or when the current size of a monitoring tree reaches a pre-defined threshold value.

- ***Getting the time-interval weight***: The time-interval weight of the sequence S_k is computed from the time-intervals of data elements in the sequence.
- ***Parameter updating***: The total number of sequences in the current data stream is updated.
- ***Count updating & Sequential pattern insertion***: For each sequential pattern s that appears in the new sequence S_k, if its corresponding node with an entry (*cnt*, *cnt_r*, *tid, tid_r*) is in the monitoring tree and it is not traversed yet by the new sequence S_k, the *cnt* and *tid* of the entry are updated subsequently. If the sequential pattern s is a subsequence of the remaining-sequence $R(S_k)$ of the new sequence S_k and the remaining count of the entry is not updated yet by the new sequence S_k, i.e., $tid\_r < k$, the *cnt_r* and *tid_r* of the entry are subsequently updated. When the updated support of a sequential pattern in the monitoring tree becomes less than $Supp_{sig}$, the sequential pattern is regarded as an insignificant one, and it is pruned from the monitoring tree. On the other hand, for each sequential pattern s that appears in the new sequence S_k, if its corresponding node is not in the monitoring tree but s is a significant sequential pattern, its corresponding node with an entry (*cnt*, *cnt_r*, *tid, tid_r*) is inserted to the monitoring tree. The values of *cnt*, *cnt_r*, *tid* and *tid_r* in the entry are initialized as described in [6].
- ***Frequent sequential pattern selection***: All the currently frequent sequential patterns in the monitoring tree, whose current supports are no less than *minSupp*, are found by traversing all the paths of the monitoring tree as in the conventional mining methods based on a lexicographic tree structure.
- ***Force-pruning***: All the insignificant sequential patterns in a monitoring tree can be pruned together by examining the current support of every sequential pattern in the monitoring tree.

5. Performance Evaluation

To evaluate the effectiveness and efficiency of the proposed method, five data sets listed in Table 3 are used in this paper, and each data set is derived from a corresponding base data set generated by the IBM data generator [15]. The IBM data generator is widely used to generate data sets for performance evaluation of a sequential pattern mining algorithm in the fields of data mining and information systems. In all our experiments, the sequences of each data set are looked up one by one in sequence to simulate the environment of an online data stream.

Table 3 Data sets.

Data set	Base data set	# of items	Time-interval range
SDS_1M	*D*1000K.*C*10.*T*1.*S*10.*I*1	1000	0-1000
SDS_AB	*D*500K.*C*10.*T*1.*S*10.*I*1	1000	0-1000, 2000-3000
SDS_100K_1	*D*100K.*C*10.*T*1.*S*10.*I*1	1000	0-1000
SDS_100K_2			1000-2000
SDS_100K_3			2000-3000

The base data sets generated by the IBM data generator do not have any generation time information. Therefore, to use the data sets in the experiments for the proposed *TWDS* method, a corresponding generation time has to be assigned to each data element in the data sets. For this purpose, several approaches such as the approach using a probability distribution function and that using a randomization function can be considered. However, there is little relationship between the type of the approach and the performance of the proposed method, and the randomization function approach was used in this paper. For data set *SDS_1M*, the difference in generation time between two successive data elements in a sequence is in the range of 0–1000 milliseconds. The data set *SDS_AB* is composed of two consecutive subparts *part_A* and *part_B*. *Part_A* is a set of sequences generated by a set of items *set_A,* and *part_B* is a set of sequences generated by a set of items *set_B*. The two subparts are generated by the same method described in [15], but there is no common item between *set_A* and *set_B*. The time-interval between two successive data elements in a sequence is in the range of 0–1000 milliseconds in *part_A*, while it is in the range of 2000–3000 milliseconds in *part_B*. That is, the sequences in *part_B* have relatively larger time-intervals than those in *part_A*. The data sets *SDS_100K_1*, *SDS_100K_2* and *SDS_100K_3* are derived from the same base data set, but they have different time-intervals between two successive data elements in a sequence. In other words, the time-intervals between two successive data elements in a sequence are in the range of 0–1000 milliseconds, 1000–2000 milliseconds, and 2000–3000 milliseconds, respectively. Details of each data set such as a base data set and the range of a time-interval are listed in Table 3. For each base data set, five numbers denote the number of customers in the data set (D, in K), the average number of transactions per customer (C), the average number of items per transaction (T), the average length of maximal sequences (S), and the average length of transactions within the maximal sequences (I), respectively.

All the experiments were performed on a 2.8 GHz Pentium machine with 1 GB main memory running on Linux, and all the programs were implemented in C. In all the experiments, a significant support $Supp_{sig}$ was set to 30% of a support threshold *minSupp*, and a force-pruning operation was performed in every 1000 sequences.

Figures 4 and 5 show the number of sequential pat-

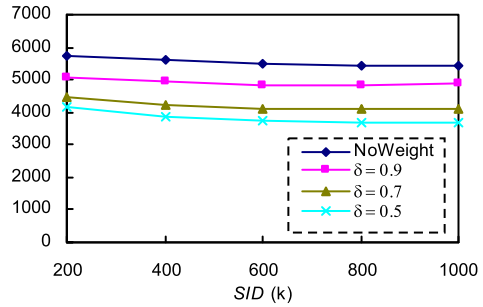


Fig. 4 Number of patterns in function of δ (u = 500, *minSupp* = 0.001).

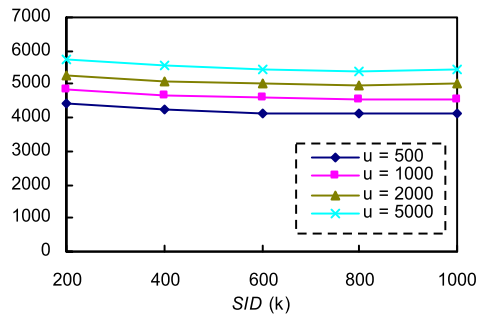


Fig. 5 Number of patterns in function of u (δ = 0.7, *minSupp* = 0.001).

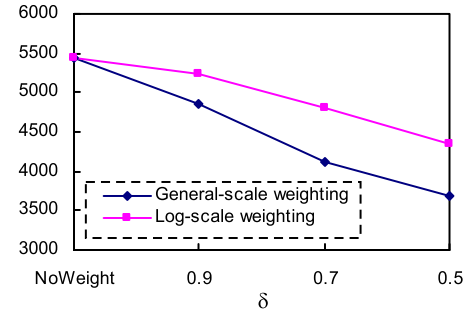


Fig. 6 Comparison of weighting functions (u = 500, *minSupp* = 0.001).

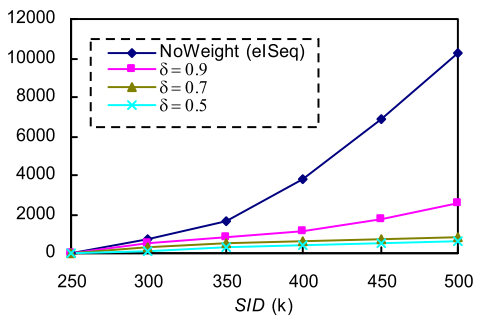


Fig. 7 Number of patterns derived from *part_B* of *SDS_AB* (u = 500, *minSupp* = 0.0005).

terns for the data set *SDS_1M* to compare the performance of the *TWDS* method by varying the values of its parameters. In this experiment, a support threshold was set to 0.001. The series of generated sequences is divided into 5 intervals, each of which consists of 200000 sequences. Figure 4 shows the number of sequential patterns in function of δ for each interval. The line named *NoWeight* shows the case of $\delta = 1.0$ which denotes the number of sequential patterns found by the eISeq method. In this case, all the sequences in a sequence data stream have the same weight regardless of the time-intervals of data elements in each sequence, so it denotes the number of sequential patterns found in mining sequential patterns based on simple support counting. Among the sequential patterns found in mining sequential patterns based on simple support counting, several sequential patterns with relatively large time-intervals were not found in a resulting set found by the *TWDS* method. Therefore, the number of sequential patterns found in the case of $\delta < 1$ is less than that in the case of *NoWeight*. Moreover, the number of patterns decreases as the value of δ becomes smaller. Figure 5 shows the number of sequential patterns in function of u. Similarly in the case of δ in Fig. 4, the number of patterns decreases as the value of u becomes smaller because it is more sensitive to the increase of a time-interval as the values of δ or u become smaller.

Figure 6 shows the changes in the number of sequential patterns for the weighting functions presented in Sect. 4.1. The values of *minSupp* and u are set to 0.001 and 500 milliseconds, respectively. The reduction rate of a time-interval weight with respect to the increase of a time-interval is relatively large in general-scale weighting, compared to log-scale weighting. That is, when the general-scale weighting function is applied, there may be a more number of sequential patterns considered as less interesting because of larger time-intervals, compared to the case when the log-scale weighting function is applied. Consequently, as shown in this figure, the number of patterns in the case of general-scale weighting is less than that in the case of log-scale weighting.

To verify the adaptability of the *TWDS* method for the change of time-intervals in a sequence data stream, the data set *SDS_AB* is used which can simulate the change of time-intervals in a sequence data stream over time. In this experiment, the values of *minSupp* and u are set to 0.0005 and 500 milliseconds, respectively. The series of generated sequences in *SDS_AB* is divided into 10 intervals, each of which consists of 50000 sequences. Figure 7 shows the number of sequential patterns derived from *part_B* of the data set whose time-intervals between two successive data elements are in the range of 2000–3000 milliseconds, and it shows last five intervals. As shown in this figure, in the line named *NoWeight* which denotes the number of sequential patterns found by the eISeq method, the number of sequential patterns derived from *part_B* greatly increases as the sequences are continuously generated despite the large time-intervals in *part_B*. However, in the other cases when the values of δ are less than 1.0, the number of sequential patterns derived from *part_B* is much less than that in the case of $\delta = 1.0$, even though they are increased as the sequences are continuously generated. The sequences in *part_B* have relatively large time-intervals between two successive data

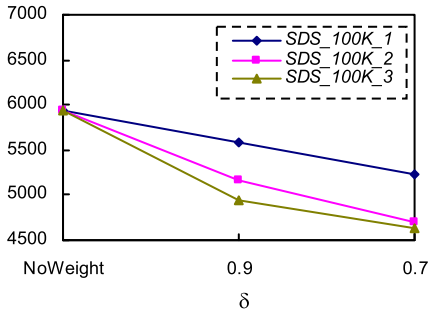


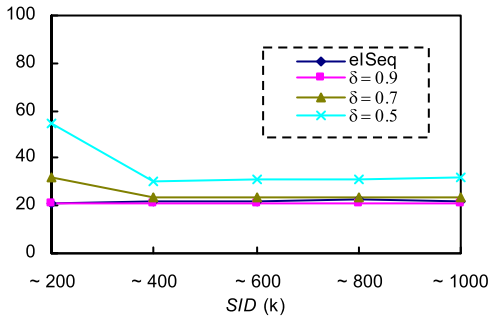
Fig. 8 Number of patterns for various data sets with different time-intervals (u = 1000, *minSupp* = 0.001).

elements; therefore, the sequential patterns appearing in the sequences have a relatively smaller time-interval weighted support, and many of them cannot be found in the resulting set of time-interval weighted sequential patterns.

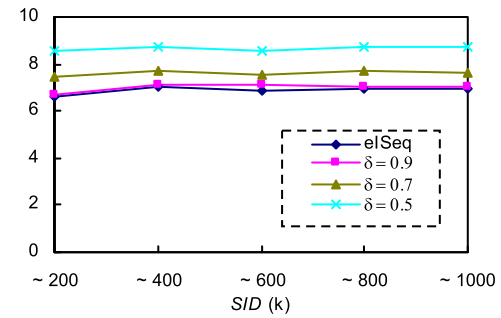
Figure 8 shows the number of sequential patterns found by the *TWDS* method on three data sets which are generated by the same base data set but the time-intervals between two successive data elements are in different ranges for one another. For this experiment, three data sets *SDS_100K_1*, *SDS_100K_2*, and *SDS_100K_3* are used, and the values of *minSupp* and u are set to 0.001 and 1000 milliseconds, respectively. In this figure, the number of sequential patterns for each data set and each value of δ is found when all the sequences in the data set are processed, i.e., right after the 100000th sequence is processed. For the same value of δ, more number of sequential patterns are found in the resulting set of time-interval weighted sequential patterns on *SDS_100K_1* since its time-intervals between two successive data elements are smaller than those of the other data sets. For the same data set, as described in Fig. 4, the number of sequential patterns in its resulting set decreases as the value of δ decreases.

To verify the basic performance of the *TWDS* method, its memory usage and processing time per sequence in its mining process are compared with those of the eISeq method that is one of the conventional mining methods for finding sequential patterns over data streams. For this purpose, the data set *SDS_1M* is used, and the values of *minSupp* and u are set to 0.001 and 500 milliseconds, respectively. The series of generated sequences is divided into 5 intervals, each of which consists of 200000 sequences.

Figure 9 (a) shows the memory usage of the proposed method. For each interval, the memory usage is represented by the maximum usage in the interval. Since only the significant sequential patterns are maintained by delayed-insertion and pruning operations, the memory usage remains almost the same although new sequences are continuously generated. On the other hand, for mining time-interval weighted sequential patterns by the *TWDS* method over a sequence data stream $TiDS_k$, the smaller the value of δ, the smaller the weight of each sequence in the data stream. Therefore, for a sequence data stream, the value of $|TiDS|_k$ decreases as the value of δ decreases, and then there



(a) Memory usage



(b) Processing time

Fig. 9 Basic performance of the *TWDS* method (u = 500, *minSupp* = 0.001).

emerges a greater possibility that the sequential pattern that appears in a small number of sequences can be a significant sequential pattern. Consequently, as the value of δ decreases, the number of significant sequential patterns to be maintained in memory increases, i.e., the memory usage of the *TWDS* method increases. For the first interval, the value of $|TiDS|_k$ is much less than those in the other intervals, so that the memory usage in the interval is greater than those in the other intervals for the same reason as aforementioned.

Figure 9 (b) shows the average processing time per sequence of the proposed method in each interval. The processing time per sequence is measured by a period from the generation of a new sequence to the end of a sequence insertion operation. As shown in this figure, the average processing time is less than 10 milliseconds. As the value of δ decreases, the memory usage of the *TWDS* method increases as shown in Fig. 9 (a), and then the processing time required to traverse a monitoring tree increases to estimate the weighted support of a new sequential pattern. As a result, the average processing time increases.

As shown in Fig. 9, the memory usage and the processing time of the *TWDS* method increase a little compared with those of the eISeq method. However, they remain small even if the target data stream is continuously expanded, so that it can get the mining result for mining data streams with a small memory in a very short time. Consequently, it can be useful for mining data streams.

6. Conclusions

For a sequence or a sequential pattern, the generation times and time-intervals are as important as the generation order of data elements. In sequential pattern mining, therefore, the time-interval information of data elements can help to get more valuable sequential patterns. To obtain more valuable sequential patterns, this paper analyzed the weight of a sequence based on the time-intervals between its data elements, differentiating the importance, i.e., the interestingness, of a sequence as well as that of a sequential pattern. Through this mechanism, more interesting sequential patterns can be selectively found in mining sequence data stream.

To develop a novel interesting sequential pattern of a time-interval weighted sequential pattern for mining sequence data streams, this paper presented a new technique to get the weight of a sequence in a sequence data stream. The weight is computed from the time-intervals of the items in the sequence. After defining the novel interesting sequential pattern of a time-interval weighted sequential pattern based on the weight of a sequence, a new framework to find the patterns in a sequence data stream was presented. In addition, a mining method for finding time-interval weighted sequential patterns over a sequence data stream was developed which can find its up-to-date mining result in a short time with a small memory over the sequence data stream.

Recently, various application fields generate data in the form of data streams. Especially, the time-interval sequence data streams are common in such daily activities as retailing, traveling, and E-commerce. Continuously generated transaction records of customers can be viewed as sequence data streams in the retailing business. In the field of a web-based service such as E-commerce, the member records and web-access logs can be viewed as sequence data streams. Consequently, the time-interval weighted sequential patterns presented in this paper can be effectively used in various professional fields, and the information obtained by the mining process can offer great benefits to the corporations and individuals related to the fields.

A promising direction for future research to make the proposed term and framework more useful is the optimal selection of parameters in the time-interval weighting function.

Acknowledgements

We would like to thank the editor of the 'IEICE Trans. on Inf. & Syst.' and anonymous reviewers for their constructive comments on an earlier version of this paper. This research was partially supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT & Future Planning (No. 2012R1A1B4000651), and was partially supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science Technology(NRF-2011-0025300).

References

[1] J. Kang, J.F. Naughton, and S.D. Viglas, "Evaluating window joins over unbounded streams," Proc. 19th Int'l Conf. on Data Engineering, pp.341–352, 2003

[2] M. Garofalakis, J. Gehrke, and R. Rastogi, "Querying and mining data streams: You only get one look," The Tutorial Notes of the 28th Int'l Conf. on Very Large Data Bases, 2002.

[3] J.H. Chang and W.S. Lee, "A sliding window method for finding recently frequent itemsets over online data streams," J. Information Science and Engineering, vol.20, pp.753–762, 2004.

[4] G. Mao, X. Wu, X. Zhu, G. Chen, and C. Liu, "Mining maximal frequent itemsets from data streams," J. Inf. Sci., vol.33, pp.251–262, 2007.

[5] J.X. Yu, Z. Chong, H. Lu, Z. Zhang, and A. Zhou, "A false negative approach to mining frequent itemsets from high speed transactional data streams," Inf. Sci., vol.176, pp.1986–2015, 2006.

[6] J.H. Chang and W.S. Lee, "Efficient mining method for retrieving sequential patterns over online data streams," J. Inf. Sci., vol.31, pp.420–432, 2005.

[7] Q. Huang and W. Ouyang, "Mining sequential patterns in data streams," Proc. 6th Int'l Symposium on Neural Networks, pp.865–874, 2009.

[8] C.-H. Lin, D.-Y. Chiu, Y.-H. Wu, and A.L.P. Chen, "Mining frequent itemsets from data streams with a time-sensitive sliding window," Proc. 5th SIAM Int'l Conf. on Data Mining, pp.68–79, 2005.

[9] S. Lo, "Binary prediction based on weighted sequential mining method," Proc. 2005 Int'l Conf. on Web Intelligence, pp.755–761, 2005.

[10] U. Yun, "WIS: Weighted interesting sequential pattern mining with a similar level of support and/or weight," ETRI J., vol.29, pp.336–352 2007.

[11] U. Yun, "A new framework for detecting weighted sequential patterns in large sequence databases," Knowledge-Based Systems, vol.21, pp.110–122, 2008.

[12] J.H. Chang, "Mining weighted sequential patterns in a sequence database with a time-interval weight," Knowledge-Based Systems, vol.24, pp.1–9, 2011.

[13] Y.-L. Chen, M.-C. Chiang, and M.-T. Ko, "Discovering fuzzy time-interval sequential patterns in sequence databases," IEEE Trans. Syst. Man Cybern. B, Cybern., vol.35, pp.959–972, 2005.

[14] Y.-L. Chen and T.C.-H. Huang, "Discovering time-interval sequential patterns in sequence databases," Expert Systems with Applications, vol.25, pp.343–354, 2003.

[15] R. Agrawal and R. Srikant, "Mining sequential patterns," Proc. 1995 Int'l Conf. on Data Engineering, pp.3–14, 1995.

[16] Y.-H. Hu, Y.-L. Chen, and K. Tang, "Mining sequential patterns in B2B environment," J. Inf. Sci., vol.35, pp.677–694, 2009.

[17] M.-Y. Lin, S.-C. Hsueh, and C.-W. Chang, "Fast discovery of sequential patterns in large databases using effective time-indexing," Inf. Sci., vol.178, pp.4228–4245, 2008.

[18] J. Pei, J. Han, B. Mortazavi-Asl, J. Wang, H. Pinto, Q. Chen, U. Dayal, and M.-C.- Hsu, "Mining sequential patterns by pattern-growth: The prefixspan approach," IEEE Trans. Knowl. Data Eng., vol.16, pp.1424–1440, 2004.

[19] M.J. Zaki, "SPADE: An efficient algorithm for mining frequent sequences," Mach. Learn., vol.42, pp.31–60, 2001.

[20] J. Pei, J. Han and W. Wang, "Mining sequential patterns with constraints in large databases," Proc. 2002 ACM Int'l Conf. on Information and Knowledge Management, pp.18–25, 2002.

[21] X. Ji, J. Bailey and G. Dong, "Mining minimal distinguishing subsequence patterns with gap constraints," Knowl. Inf. Syst., vol.11, pp.259–296, 2007.

[22] R.C. Agarwal, C.C. Aggarwal, and V.V.V. Prasad, "A tree projection algorithm for generation of frequent itemsets," J. Parallel Distrib.

Comput., vol.61, pp.350–371, 2001.

Joong Hyuk Chang received the B.S. and M.S. degree in Computer science from Yonsei University, Seoul, Korea, in 1996 and 1998, and also received the Ph.D. degree in Computer Science from Yonsei University in 2005. He was a post-doctoral research associate in the Department of Computer Science at the University of Illinois at Urbana-Champaign. He is currently a professor of Department of Computer & Information Technology at Daegu University, Daegu, Korea. He has been researching into mining and processing over data streams in-cluding Web data stream and ubiquitous data stream, anomaly detection system, data mining and knowledge in large-scale data sets, bioinformatics and database systems.

Nam Hun Park received the B.S., M.S. and Ph.D. degree in Computer Science from Yonsei University, Seoul, Korea, in 2000, 2002 and 2007. He was a post-Ph.D. at the Department of Computer Science, Worcester Polytech Institute, Worcester, MA. He is currently a professor of Department of Computer Science at Anyang University, Korea. His current interests include mining data streams.