Concept Shift Detection for Frequent Itemsets from Sliding Windows over Data Streams*

Jia-Ling Koh and Ching-Yi Lin

Department of Computer Science and Information Engineering
National Taiwan Normal University
Taipei, Taiwan
Email: jlkoh@csie.ntnu.edu.tw

Abstract. In a mobile business collaboration environment, frequent itemsets analysis will discover the noticeable associated events and data to provide important information of user behaviors. Many algorithms have been proposed for mining frequent itemsets over data streams. However, in many practical situations where the data arrival rate is very high, continuous mining the data sets within a sliding window is unfeasible. For such cases, we propose an approach whereby the data stream is monitored continuously to detect any occurrence of a concept shift. In this context, a “concept-shift” means a significant number of frequent itemsets in the up-to-date sliding window are different from the previously discovered frequent itemsets. Our goal is to detect the notable changes of frequent itemsets according to an estimated changing rate of frequent itemsets without having to perform mining of the frequent itemsets at every time point. Consequently, for saving the computing costs, it is triggered to discover the complete set of new frequent itemsets only when any significant change is observed. The experimental results show that the proposed method detects concept shifts of frequent itemsets both effectively and efficiently.

Keywords: Frequent Itemsets, Data Streams, Change Detection.

1 Introduction

In recent times, data stream mining has received more attention due to the increasing number of streaming applications. Among the different applications, the mobile environment is a major source of data streams. Unlike mining static databases, mining data streams presents many new challenges. Since the amount of data over a data stream has no theoretical boundary, it is unrealistic to keep the entire stream. Accordingly, the traditional methods of mining static datasets using multiple scans are unfeasible. Additionally, a great deal of processing power is required to keep up with the high data arrival rate.

* This work was partially supported by the R.O.C. N.S.C. under Contract No. 97-2221-E-003-007 and 97-2631-S-003-003.

Frequent itemset mining is well recognized as a fundamental problem in important data mining tasks such as finding sequential patterns [13], [14], clustering [15], and classification [10]. Since the problem was first identified by Agrawal et al. [1], many efficient algorithms for mining frequent itemsets on static databases have been proposed over the last decade [5], [6]. In a mobile business collaboration environment, frequent itemsets analysis will discover the noticeable associated events or data to provide important information of user behaviors.

Many algorithms have been proposed for mining frequent itemsets over data streams [3]. Lossy-counting is the most popular approach among landmark-model algorithms for mining frequent itemsets over data streams [11]. Given an error tolerance parameter ε, the Lossy-counting algorithm prunes patterns with support being less than ε from the pool of monitored patterns such that the required memory usage is reduced. Consequently, the frequency of a pattern is estimated by compensating the maximum number of times that the pattern could have occurred before being inserted into the monitored patterns. It has been proven that no false dismissal occurs with the Lossy-counting algorithm and that the error of the estimated frequency is guaranteed not to exceed a given error tolerance parameter.

In addition to the restriction on memory requirement introduced earlier, the time sensitivity issue is another important concern when mining frequent itemsets over data streams. It is likely that the embedded knowledge in a data stream will change quickly over time. In order to provide the most recent trend of a data stream, many algorithms have been proposed for mining frequent itemsets by adopting the sliding window model. Under the assumption that exactly one transaction arrives during each time unit, [2] defines the current sliding window to consist of the most recent w transactions in a data stream given a window size of w. Consequently, the recently frequent itemsets were defined to be the frequent itemsets mined from the current sliding window. In addition to counting the occurrence of a pattern in the new transaction, the count of a pattern in the oldest transaction has to be removed from the maintained data structure when the window is sliding. In [9], it was assumed that a block of varying transactions (zero or more transactions) is inputted into the data stream at each time unit. Accordingly, the frequent itemsets were discovered from the most recent w blocks of transactions. For each block of transactions, the frequent itemsets in the block were found and all possible frequent itemsets in the sliding window were collected in a PFP (Potential Frequent-itemset Pool) table. For each newly inserted pattern, the maximum number of its previous appearances was estimated. Moreover, a discounting table was constructed to provide approximate counts of the expired data items.

Detecting changes in a data stream is an important area of research and has many applications. A method for the detection and estimation of changes according to data distributions was proposed in [7]. Although many studies have provided efficient techniques to support the incremental computation of frequent itemsets over data streams, it is not trivial for users to discover changes in patterns from the obtained results when the amount of frequent itemsets is large. A naïve solution to this problem can be implemented in two phases: (1) using a pattern mining algorithm to discover frequent itemsets; and (2) matching the new set of frequent itemsets with the old set. As stated in [12], in many practical situations where the data arrival rate is very high, it is unfeasible to continuously mine the data set within a sliding window. For such cases, we propose an approach whereby the data stream is monitored continuously to detect any occurrence of a concept shift. In this context, “concept-shift” means that a significant number of frequent itemsets in the up-to-date sliding window are different from the previously discovered frequent itemsets. Our goal is to detect the notable changes in frequent itemsets according to an estimated changing rate of frequent itemsets without having to perform frequent itemsets mining at every time point. Consequently, for saving the computing costs, it is triggered to discover the complete set of new frequent itemsets only when any significant change is observed.

In a sliding window model, when the window slides, the previously discovered frequent itemsets which are no longer applicable within the new window can be obtained by monitoring their change in frequency over windows. However, the difficulty lies with incrementally discovering the newly generated frequent itemsets without keeping their previous counts. To handle such a challenge, this paper applies the power-law distribution of supports for itemsets to estimate the amount of newly generated frequent itemsets. An algorithm, named **F**requent itemsets **C**hange **D**etection method (FCDT), is proposed to monitor the changing ratios of frequent itemsets over a data stream updated per **T**ransaction. Besides, the proposed method is extended to develop the algorithm FCDB for monitoring a data stream updated per **B**atch. The experimental results show that FCDT and FCDB can detect concept shifts of frequent itemsets both effectively and efficiently. The execution time of FCDT and FCDB is significantly less than the time required to mine the complete set of frequent itemsets over a sliding window at each time point. Furthermore, the information of frequency change for frequent itemsets is provided from the maintained data structures.

The remaining sections of this paper are organized as follows. The problem of concept-shift detection for frequent itemsets is defined in Section 2. The proposed data structures and monitoring method of the FCDT algorithm are introduced in Section 3 and 4, respectively. The performance study is reported in Section 5, which shows the effectiveness and efficiency of the proposed method. Finally, in Section 6, we conclude this paper and discuss directions for our future studies.

2 Problem Definition

Let $I = \{i_1, i_2, \ldots, i_m\}$ denote the set of items in the specific application domain where a transaction is composed of a set of items in *I*. A ***basic block*** is a set of transactions arriving within a fixed unit of time. A data stream, $DS = [B_1, B_2, \ldots)$, is a continuous sequence of basic blocks, where each basic block B_i, associated with a time identifier *i*, consists of a various number of transactions $\{T_1^i, T_2^i, \ldots, T_{i_k}^i\}$($i_k \geq 0$). A special case occurs when there is exactly one transaction in each basic block, which is categorized into a data stream being *updated per transaction* [3]. The general case is called a data stream being *updated per batch*. Let $DS(i,j)$ denote the set of transactions collected from basic blocks of *DS* from time *i* to *j*. Under a predefined window size *w*, the ***sliding window*** at time *t*, denoted as SW_t, corresponds to $DS(t-w+1, t)$. The number of transactions in SW_t is denoted as $|SW_t|$. Supposing that *t'* denotes current time, the corresponding window $SW_{t'}$ is named the ***current sliding window.***

An itemset (or a pattern) is a set consisting of one or more items in *I*, that is, a non-empty subset of *I*. The number of elements in an itemset *p* is called the length of *p*. If the itemset *p* is a subset of transaction *T*, we say that *T* ***contains*** *p*. The number of transactions in SW_t which contain *e* is named the ***support count*** of *p* over *DS* at *t*, denoted as $SC_t(p)$. Given a user specified minimum support threshold between 0 and 1, denoted as *min_support*, an itemset *p* is called a ***frequent*** itemset at *t* over *DS* if $SC_t(p) \geq \lceil |SW_t| \times min\_support \rceil$. Otherwise, *p* is an ***infrequent*** itemset at *t*. In the following, $\lceil |SW_t| \times min\_support \rceil$ is known as a minimum support count threshold, which is denoted as *min_SC*.

Given two time identifiers *t* and *t'* (*t'*>*t*), let F_t and $F_{t'}$ denote the set of frequent itemsets at time *t* and *t'*, respectively. Accordingly, ($F_{t'}$-F_t) is the set of newly generated frequent itemsets at *t*' with respect to time *t*, which is denoted as $F_t^+(t')$. Additionally, $F_t^-(t')$ is defined as the set of itemsets which are frequent at *t* but become infrequent at *t'*, that is $F_t^-(t') = F_t - F_{t'}$. The changing ratio of frequent itemsets at *t'* with respect to *t*, denoted as $FChange_t(t')$ is defined by the following equation:

$$FChange_t(t') = \frac{|F_t^+(t')| + |F_t^-(t')|}{|F_t| + |F_t^+(t')|} . \tag{1}$$

The value of $FChange_t(t')$ is between 0 and 1. If $F_{t'}$ is identical to F_t, the changing ratio is 0. A value of 1 for the changing ratio occurs when $F_{t'}$ and F_t are disjoint.

Let *t* denote the time identifier when frequent itemsets are discovered completely from SW_t. The goal of concept shift detection is to discover the smallest *t'*, where *t'*> *t*, such that $FChange_t(t')$ is larger than a given threshold value δ. The main issue studied in this paper is to estimate the value of $FChange_t(t')$ without needing to perform complete mining of frequent itemsets at *t'*.

[Example 1]

Table 1. An example of data streams being updated per batch.

B_1	B_2	B_3	B_4	B_5	B_6	B_7	…
bcd cd	***abc bcde bc ab***	***bce e ae***	***abe***	***ab abcd***	***abd cde***	***b***	…

Table 1 shows an example of data streams being updated per batch. Suppose *min_support* is set to be 0.5 and window size *w* is 4. In this case, SW_4={***bcd, cd, ac, cde, bc, ab, bce, e, ae, be***} and $|SW_4|$ **= 10**; SW_6={***bce, e, ae, be, ab, acd, bd, cde***} and $|SW_6|$ = 8. Accordingly, F_4= {***b, c, e, bc***} and F_6= **{*a, b, e, ab*}**. Besides, $F_4^+(6)$={***a, ab***} and $F_4^-(6)$={***c, bc***}.The changing ratio of frequent itemsets at time 6 with respect to 4, denoted as $FChange_4(6)$, is (2+2)/(4+2) =2/3.

3 The Pattern Monitoring Tree

Given a time identifier *t*, for monitoring changes in frequent patterns starting at *t*, a tree structure called the ***pattern monitoring tree*** (*PM-tree*) is constructed to maintain both the frequent itemsets discovered in SW_t and candidates of newly generated frequent itemsets as time goes by. For a node N_p of a pattern *p* in the *PM-tree*, the following four fields of information are stored:

(1) $N_p.P$ stores the content of the pattern *p*.

(2) $N_p.count$ keeps the value of $SC_t(p)$-(min_SC–1) if *p* is in F_t. Otherwise, a cumulated count of *p* starting from *t* is recorded.

(3) $N_p.changing\_type$ denotes the type of status changing of *p*. The value 0 means that pattern *p* was found in F_t and remains a frequent pattern; the value 1 means that *p* was a frequent pattern but becomes infrequent; the value 2 means that *p* was an infrequent pattern but becomes frequent.

(4) $N_p.initial\_status$ denotes the initial status of *p* at time *t*. If *p* is a frequent pattern at *t*, a value of 1 is stored. Otherwise, a value of 0 is stored.

Let *t* denote the time identifier when frequent itemsets are discovered completely from SW_t, such as when the sliding window becomes full initially. For each pattern *p* in F_t, a corresponding node N_p is inserted into the *PM-tree* with $N_p.P=p$, $N_p.count$ = ($SC_t(p)$-(min_SC–1)), $N_p.initial\_status$ = 0, and $N_p.changing\_status$ =1. To speed up the process of pattern searching, the nodes in the *PM-tree* are organized as a prefix-tree according to their corresponding patterns.

Let *t'* denote the current time identifier. Since only frequent itemsets at time *t* are maintained in the *PM-tree*, when a pattern not in F_t appears at *t'* (*t'*>*t*), its support count within the sliding window $SW_{t'}$ is unknown. However, for each itemset *p*, it satisfies the following inequality:

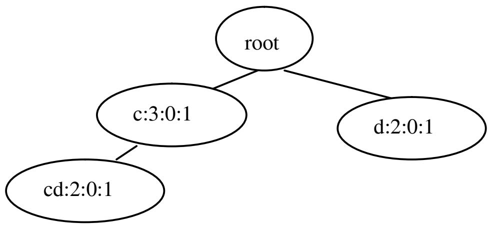
$$SC_{t'}(p) \le \min(\{SC_{t'}(I_i) \mid I_i \in p\}). \qquad (2)$$

Therefore, an auxiliary array, ***ICount***, is maintained to keep the support count of each single item at the current sliding window and to estimate the upper bound of the support count for a new pattern in the monitoring process.

[Example 2]

Table 2. An example of data streams being updated per transaction.

B_1	B_2	B_3	B_4	B_5	B_6	B_7	B_8	B_9	B_{10}	B_{11}	B_{12}	B_{13}	B_{14}	B_{15}	B_{16}
cd	bcd	ac	cde	bcd	ab	cde	cd	ab	ab	cd	ae	bc	ab	bce	e



ICount

a	*b*	*c*	*d*	*e*
5	5	8	7	3

Fig. 1. The constructed *PM-tree* and *ICount* table for the example.

Table 2 shows an example of data stream being updated per transaction. Suppose *min_support* is set to be 0.5 and window size *w* is 12. At time 12, the sliding window is full initially. The discovered frequent itemsets from SW_{12} is ***c***, ***cd***, and ***d***. The content of the constructed *PM-tree* and ***ICount*** is shown in Fig. 1.

4. The FCDT Algorithm

In this section, based on the constructed pattern monitoring tree, the FCDT algorithm is proposed to monitor the changing ratios of frequent itemsets over a data stream updated per transaction. The FCDB algorithm is extended from FCDT for solving the same problem over a data stream updated per batch.

4.1 The Pattern Monitoring Process

The pattern monitoring process starts from the next time point after the *PM-tree* is constructed. Initially, all the patterns in the *PM-tree* belong to F_t. During the pattern motoring process, in addition to maintaining the information of the support count and the changing status of patterns in the *PM-tree*, the newly inputted patterns are inserted into the tree if they become frequent according to their estimated support. However, each pattern which does not belong to F_t will be removed from the *PM-tree* when it becomes infrequent.

For estimating the changing ratio of frequent itemsets, two types of counters are used to record the amounts of the status changing patterns. The variable ***removenum*** is an integer, which is used to count the number of patterns in $F_t^-(t')$. Moreover, an integer array, ***addnum***, is used to maintain the number of newly generated candidates of frequent itemsets with respect to F_t according to their increased support counts.

There are two tasks in maintaining the changing information of patterns when the window slides: the first is to append the newly inputted transaction, and the second is to remove the expired transaction. These tasks are described in detail below.

1) Appending the newly inputted block $B_{t'}=\{T_{t'}\}$

For each item contained in $T_{t'}$, the corresponding entry in ***ICount*** is increased by 1. For each itemset *p* contained in $T_{t'}$, the *PM-tree* is searched to find the corresponding node N_p. If node N_p exists in the *PM-tree*, the following processing is required for updating the pattern changing counters according to the value of N_p.*initial_status*:

<1> N_p.*initial_status*=1: That is, *p* belongs to F_t. Thus, the value of N_p.*count* is increased by 1. If N_p.*changing_type* is 1, it means that pattern *p* has become infrequent in an earlier stage. In this case, the newly inputted transaction will make *p* return to being frequent if the value of N_p.*count* is equal to 1. Consequently, *removenum* is reduced by 1 and N_p.*changing_type* is set to 2. If N_p.*changing_type* is equal to 0 or 2, *p* remains a frequent pattern. For these two cases, the pattern changing counters remain unchanged.

<2> N_p.*initial_status* =0: That is, *p* does not belong to F_t but becomes frequent. The value of N_p.*count* will be updated from N_p.*count* to (N_p.*count*+1). Therefore, the counter ***addnum***[N_p.*count*+1] is increased and ***addnum***[N_p.*count*] is

reduced. After completing the update on ***addnum***, the value of N_p.*count* is increased by 1.

If a corresponding node of *p* does not exist in the *PM-tree*, it is indicated that *p* is not in F_t. The difficulty here is that the true support count of *p* in the sliding window is unknown since this pattern was infrequent at the previous time points. To address this problem, the value of $\mathbf{min}(\{SC_{t'}(I_k) \mid I_k \in p\})$, denoted as $U\_SC_{t'}(p)$, is obtained according to the counting information stored in ***ICount*** to get the upper bound of $SC_{t'}(p)$. Then, the support count of *p* is estimated to be the smaller value between *min_SC* and $U\_SC_{t'}(p)$. If the value is equal to *min_SC*, it is possible that *p* becomes a frequent itemset. Accordingly, the node N_p of *p* is inserted into the *PM-tree*, with $N_p.P=p$, $N_p.count$=1, $N_p.initial\_status$=0 and $N_p.changing\_type$=2. Additionally, ***addnum***[1] is increased by 1.

2) Removing the expired block $B_{t'-w}=\{T_{t'-w}\}$

For each item contained in $T_{t'-w}$, the corresponding entry in ***ICount*** is reduced by 1. For each itemset *p* contained in $T_{t'-w}$, if the corresponding node N_p exists in the *PM-tree*, the following processing is required for updating the pattern changing counters according to the value of $N_p.initial\_status$:

<1> $N_p.initial\_status$=1: That is, *p* is in F_t. The value of $N_p.count$ is reduced by 1. If $N_p.count$ is larger than 0, it means that pattern *p* remains a frequent pattern. Otherwise, the pattern becomes infrequent when $N_p.count$ is equal to 0. Accordingly, $N_p.changing\_type$ is checked to verify the previous type of status changing of *p*. If the value of $N_p.changing\_type$ is either 0 or 2, it indicates that *p* will become infrequent from frequent due to the removal of $T_{t'-w}$. Thus, the counter ***removenum*** is increased and $N_p.changing\_type$ is set to 1.

<2> $N_p.initial\_status$=0: The $N_p.count$ will be updated from $N_p.count$ to ($N_p.count$-*1*). If ($N_p.count$-1) is larger than 0, the counter ***addnum***[$N_p.count$] is reduced by 1 and ***addnum***[$N_p.count$-1] is increased by 1. After completing the update on ***addnum***, the value of $N_p.count$ is decreased by 1. Otherwise, if ($N_p.count$-1) is equal to 0, it means that the pattern *p* will become infrequent again. In addition to reducing the value of ***addnum***[$N_p.count$], the corresponding node N_p is removed from the *PM-tree*. According to the downward closure property of frequent itemsets, since *p* becomes infrequent, all the super-patterns of *p* must therefore be infrequent. It is implied that, for each descendant node of N_p, denoted as $N_{p'}$, which corresponds to a super-pattern of *p*, the support count was previously over-estimated. Therefore, for such a node $N_{p'}$, the counter ***addnum***[$N_{p'}.count$] is reduced by 1 and node $N_{p'}$ is removed from the tree to reduce the estimating error.

If a corresponding node of *p* in the *PM-tree* does not exist, it means that the pattern has not become a candidate of frequent itemsets. In this situation, no information of the pattern *p* is kept or updated.

4.2 Estimation Method of the Changing Ratio

In the pattern monitoring process described in the previous section, when a pattern *p* is added into the *PM-tree*, its true frequency in the sliding window is unknown. Due to limited information, the support count of *p* is estimated to be the smaller value between *min_SC* and $U\_SC_{t'}(p)$. However, in most cases, the estimated support count is higher than the actual one of a pattern. Accordingly, the amount of newly generated frequent itemsets, $|F_t^+(t')|$, is also over-estimated. For solving this problem, the power-law relationship appears in the distribution of supports of itemsets is used to adjust the value of $|F_t^+(t')|$.

In [4], the author identified that the power-law relationship appears in the distribution of supports of itemsets. Let s_i denote a value of the support count and f_i denote the number of itemsets with their support counts being equal to s_i. The power-law relationship is described by the following equation:

$$\log(f_i) = \theta\log(s_i)+\Omega. \tag{3}$$

In other words, the amount of patterns with a specific support can be estimated after the characteristics of the power-law relationship are extracted.

Among the frequent itemsets at *t*, which are known, the number of frequent itemsets with support count s_i is counted and denoted as f_i. By sampling (f_i, s_i) pairs of frequent itemsets, the parameters θ and Ω in the power-law relationship are extracted after solving the linear regression problem. Accordingly, when assigning a value *k* ($min\_SC > k \geq 1$) to s_i, the number of infrequent itemsets with support count being *k* is approximately estimated. An auxiliary array ***PLArray*** is used to store these values, where ***PLArray***[*k*] keeps the estimated value of the number of patterns with support count being equal to *k*.

During the pattern monitoring process, the ***addnum*** array cumulates the number of possible newly generated frequent itemsets according to their appearance frequency after time *t*. In other words, for each pattern which contributes one count in ***addnum*** [*k*], it appears *k* times after *t*. If it actually becomes a frequent itemset, its support count at *t* must be no less than (*min_SC -k*).

Let ***addnum_up***[*k*] denote the actual number of new frequent itemsets with appearance frequency *k* after time *t*. Accordingly, the value of ***addnum_up***[1] must be no more than both ***addnum***[1] and ***PLArray***[*min_SC*-1]. Thus, ***addnum_up***[1] is set to be the smaller value of ***addnum***[1] and ***PLArray***[*min_SC*-1]. In addition, a pattern with support (*min_SC*-1) at *t* will become frequent if its count increased 2 after *t*. Therefore, the value of ***addnum_up***[2] must be no more than ***addnum***[2] and (***PLArray***[*min_SC*-2]+(***PLArray***[*min_SC*-1]-***addnum_up***[1])). In general, the value of ***addnum_up***[*k*] is obtained according to the following equation:

$$\textbf{\textit{addnum\_up}}[k]=\textbf{\textit{min}}(\textbf{\textit{addnum}}[k], (\textbf{\textit{PLArray}}[SC_{min}-k]+\sum_{i=1}^{k-1} d_i)),$$

where d_i = (***PLArray***[SC_{min}-*i*]- ***addnum_up***[*i*]) if ***PLArray***[SC_{min}-*i*]> ***addnum_up***[*i*], otherwise, d_i =0. (4)

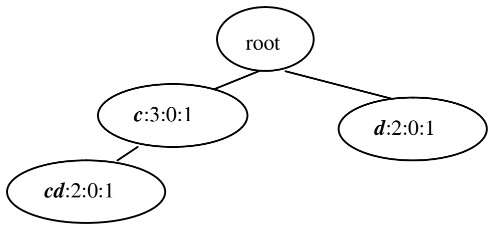
Consequently, $|F_t^+(t')|$is estimated using the following equation:

$$|F_t^+(t')| = \sum_{k=1}^{SC_{\min}-1} \textbf{\textit{addnum\_up}}[k] \tag{5}$$

On the other hand, $|F_t^-(t')|$is obtained directly from the value of ***removenum***.

Finally, the value of $FChange_t(t')$ is estimated using equation (1) defined in section 2 to decide whether a concept shift of frequent itemsets occurs. If $FChange_t(t')$ is

B_1	B_2	B_3	B_4	B_5	B_6	B_7	B_8	B_9	B_{10}	B_{11}	B_{12}	B_{13}	B_{14}	B_{15}	B_{16}
cd	***bcd***	***ac***	***cde***	***bcd***	***ab***	***cde***	***Cd***	***ab***	***ab***	***cd***	***ae***	***bc***	***ab***	***bce***	***e***



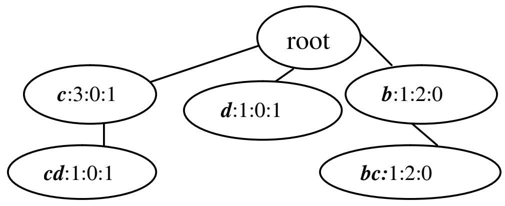
ICount

a	***b***	***c***	***d***	***e***
5	5	8	7	3

PLarray

1	2	3	4	5
2	6	2	0	2

(a)

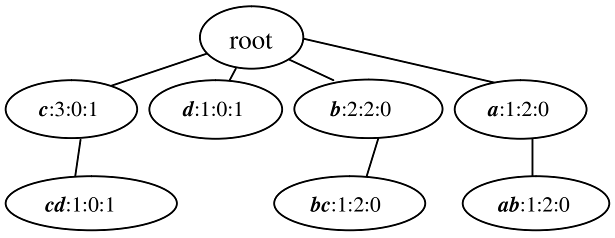


ICount

a	*b*	*c*	*d*	*e*
5	6	8	6	3

removenum=0
addnum[1] = 2

(b)

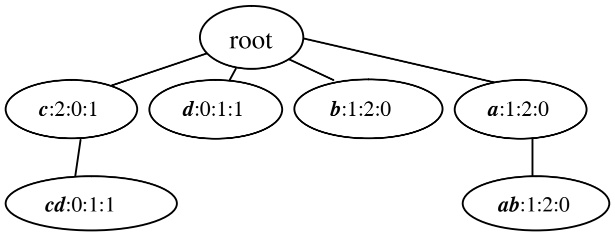


ICount

a	*b*	*c*	*d*	*e*
6	7	8	6	3

removenum=0
addnum[1] = 3
addnum[2] = 1

(c)



ICount

a	*b*	*c*	*d*	*e*
6	6	7	5	3

removenum=2
addnum[1] = 3

(d)

Fig. 2. The monitoring process of the example.

under the given threshold value δ, the monitoring process continues. Otherwise, the task of frequent pattern mining is triggered to discover the complete set of frequent itemsets in $SW_{t'}$. After resetting the monitoring environment, including resetting t to be t', reconstructing the *PM-tree*, and initializing counters ***addnum*** and ***removenum***, a new cycle of frequent pattern monitoring begins.

[Example 3] By following Example 2, the initial PM-tree is constructed as shown in Fig. 2(a). The corresponding ***ICount*** and ***PLArray*** are also shown. After that, the pattern monitoring process begins.

At time 13, after inserting transaction {***bc***} and removing transaction {***cd***}, the *PM-tree* and ***ICount*** table are updated as shown in Fig. 2(b). Because of the insertion of transaction {***bc***} in B_{13}, itemsets ***b*** and ***bc***, whose support counts are estimated to be (*min_SC*–1)+1, possibly become new frequent itemsets with respect to F_{12}. Accordingly, the value in ***addnum***[1] is set to be 2. On the other hand, no previously frequent itemset becomes infrequent due to the removing of transaction {***cd***} in B_1.

At time 14, after inserting the transaction {***ab***} in B_{14}, the itemsets ***a*** and ***ab*** are estimated to possibly become new frequent itemsets with respect to F_{12}, whose corresponding nodes are inserted into the *PM-tree*. Accordingly, addnum[1] is increased by 2. Besides, for the itemset ***b***, its support count becomes (*min_SC*–1)+2 from (*min_SC*–1)+1. Thus, addnum[1] is decreased by 1 and addnum[2] is increased by 1. The resultant *MP-tree*, ***ICount*** table, and ***addnum*** are updated as Fig. 2(c). Finally, the expired transaction {***bcd***} in B_2 is removed. Therefore, the itemsets {***c***} and {***cd***} become infrequent from frequent with respect to F_{12}, which result in the value in ***removenum*** is updated to be 2. For the itemset ***b***, whose support count is updated to be (*min_SC*–1)+1 from (*min_SC*–1)+2. Moreover, the itemset ***bc*** becomes an infrequent itemset set and is removed from the MP-tree because its support count is reduced to be (*min_SC*–1). The values in ***addnum***[1] and ***addnum***[2] are consequently updated. Fig. 2(d) show the result after removing the transaction in B_2. According to equation (4), ***addnum_up***[1]=**min**(***addnum***[1], ***PLArray***[5])= **min**(3,2) =2. Since ***addnum***[*i*]=0 for *i* = 2 to 5, $|F_{12}^{+}(14)|$ is estimated to be 2. It shows that the over-estimated value of ***addnum***[1] is adjusted according to the value in ***PLArray***[5]. Consequently, $FChange_{12}(14)$ is estimated to be (2+2)/(3+2) =0.8.

4.3 Extension for processing the general cases

In a data stream being updated per batch, because more than one transaction is inputted in a bucket, it is possible that a pattern appears both in the newly inputted block $B_{t'}$ and the expired block $B_{t'-w}$. For such a case, the support count of the pattern is unchanged. To reduce the costs of modification on the *PM-tree*, another algorithm named the FCDB is proposed to maintain a base pattern tree and a pattern changing tree. The complete set of frequent itemsets discovered at time t is stored in the base pattern tree. All the inserted or expired patterns after time t are collected in the pattern changing tree. By combining this with the information in the base pattern tree, the amount of frequent pattern changes is estimated. Moreover, in a data stream which is updated per batch, the number of transactions in a sliding window is not fixed. Therefore, an array named ***TranArray*** is constructed to maintain the number of transactions in each basic block within the current sliding window, which is used to compute the supports of itemsets. The estimation method of the changing ratio adopted in FCDB is similar to the one in FCDT. Due to page limit, the processing detail of FCDB refers to [7].

5 Performance Evaluation

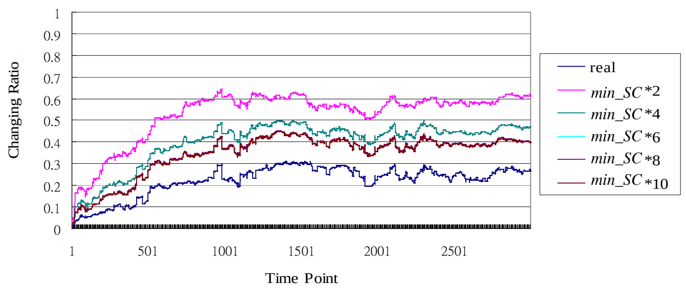
The proposed FCDT and FCDB algorithms were implemented using the Visual C++ programming language. The experiments were performed on a 3.4 GHz Intel Pentium IV machine with 2 gigabytes of memory and running Windows XP Professional. Moreover, the datasets were generated using the IBM data generator [1] where each dataset simulates a data stream. Each dataset is denoted in the form TxxIxxDxx, where T, I, and D specify the average transaction length, the average length of the maximum pattern, and the total number of transactions, respectively.

In the following two subsections, the accuracy of the estimated changing ratios and execution time were measured to show the effectiveness and efficiency of the proposed FCDT and FCDB algorithms.

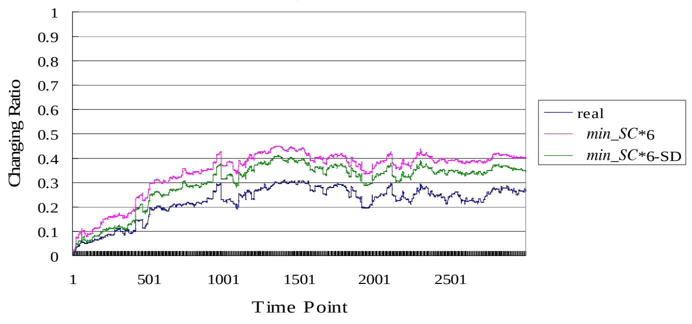
The experiments presented here used the dataset T5I4D60K to simulate a data stream with a transaction occurring at every time point. Furthermore, the following experiments were performed with *min_support* = 0.03 and $w = 1000$. The FP-growth algorithm[6] is used to discover frequent itemsets from a sliding window at every time point. Consequently, with respect to the time when the sliding window is full initially, the true frequent pattern changing rates at the following time points are obtained to evaluate the accuracy of the ones estimated by the FCDT algorithm. The execution time of the FCDT was compared against the performance of the FP-growth algorithm over each sliding window.

As stated in section 4.2, to estimate the amounts of infrequent itemsets with various supports at t, the parameters of the power-law relationship are extracted by solving the linear regression problem. In order to reduce the cost of computation, only part of the (f_i, s_i) pairs of frequent itemsets at t are selected to solve parameters θ and Ω in equation (3). In the first experiment, we observe how the amount of selected pairs influences the accuracy of the estimated changing ratio. Let the real changing ratio be known as “real”. A curve denoted by “$min\_SC \times n$” represents the estimated result of the FCDT obtained by selecting samples of (f_i, s_i) with $min\_SC \times n \geq f_i \geq min\_SC$. When n is varied from 2, 4, 6, 8, to 10, the results of the estimated changing ratios are shown in Fig. 3(a). As indicated in the figure, when n increases, the estimated result approaches the real value. However, the results of n = 6, 8, or 10, are almost identical. Accordingly, in the following experiments, $n = 6$ is used as a base for selecting (f_i, s_i) pairs to solve the parameters in the power-law distribution of supports of itemsets.

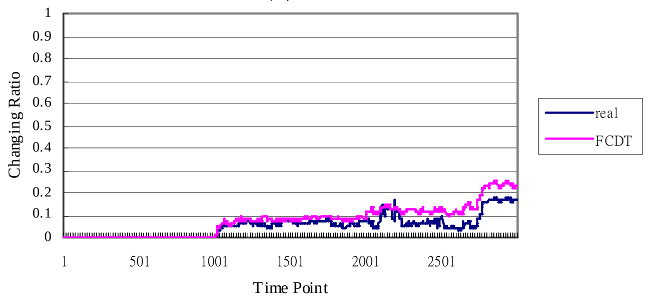
According to the results of the previous experiment, it is observed that the estimated distribution of supports of itemsets is over-estimated according to the solved parameters. Therefore, given the selected values of f_i, the standard derivation of the estimated values and real values of s_i is computed to compensate for the estimation error from the estimated values of s_i. The obtained experimental result is shown in Fig. 3(b), where the curve is labeled with “$min\_SC \times 6$-SD”.



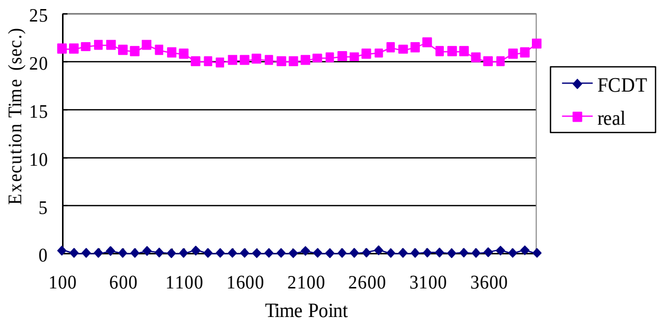
(a)



(b)

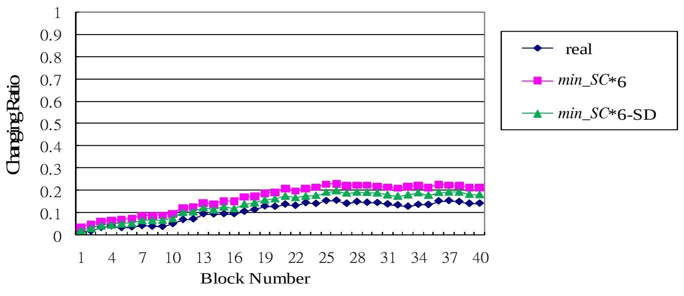


(c)

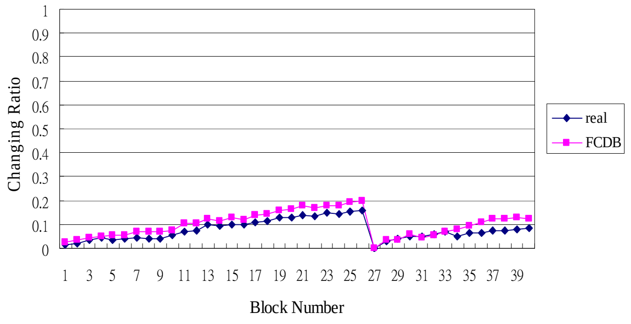


(d)

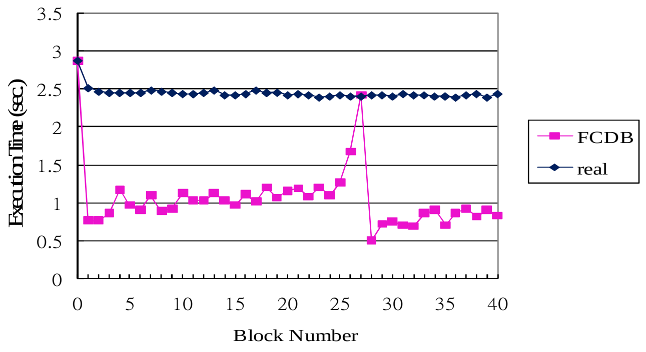
Fig. 3. The experimental results of the FCDT.



(a)



(b)



(c)

Fig. 4. The experimental results of the FCDB.

For simulating the occurrence of changes in frequent itemsets, two data sets, TD1 and TD2, consisting of 1000 transactions, are generated individually. Let the initial sliding window contain the data of TD1. After frequent itemsets in the initial window are discovered, the pattern monitoring process begins. At the beginning, from time point 1 to 1000, the data in TD1 is inputted. From time point 1001 to 2000, 90% of the data comes from TD1 and 10% of the data comes from TD2. From time point 2001 to 3000, half of the data comes from TD1 and the other half comes from TD2. It is indicated in Fig. 3(c) that the trend of estimated changing ratios is consistent with the trend of their real values. The average error of the estimated values for changing ratios is 0.025.

To evaluate the performing efficiency of the FCDT, the cumulated execution times of the FCDT and FP-Growth at every 100 time points is measured as shown in Fig. 3(d). The time required by the FCDT is almost 1/100 of the cost required by FP-Growth to discover complete frequent itemsets.1(d). The time required by the FCDT is almost 1/100 of the cost required by FP-Growth to discover complete frequent itemsets.

5.2 Experimental Results of the FCDB Algorithm

In set of experiments, the dataset T5I4D60K is used to simulate a data stream with each basic block consisting of 1000 transactions. Furthermore, when running the FCDB algorithm, the parameters *min_support* and *w* are set to be 0.02 and 20, respectively. By comparing this with the real changing ratio obtained from the mining results of FP-growth at each time point, the accuracy of the value estimated by the FCDB algorithm is observed. The execution time of the FCDB and FP-growth are also compared.

Fig. 4(a) shows the estimated changing ratios of the FCDB, where *min_SC* $\times$6 is adopted to select samples for solving the parameters θ and Ω in equation (3). The curve labeled "*min_SC* $\times$6-SD" indicates that the estimated changing ratio approaches its real value after using the standard derivation of errors to compensate for the estimation error from the estimated values of s_i.
Let the threshold of the changing ratio δ equal 0.2. The FCDB will trigger FP-growth to discover a new set of frequent itemsets when the changing ratio is larger than 0.2. In other words, it is indicated that a concept shift of frequent patterns occurs. As indicated in Fig. 4(b), the 27th block shows when a concept shift occurs. Accordingly, after the set of frequent itemsets is updated, a new cycle of detection begins. It is shown in Fig. 4(b) that the trend of estimated changing ratios is consistent with the trend of their real values. The average error of the estimated ratios is 0.024. When no concept drift occurs, only the costs of maintaining monitoring structure and the computation of the estimated changing ratio are required. Therefore, the execution of the FCDB is between 0.5 to 1.5 seconds in most cases, which is about one half of that required to perform FP-Growth at each time point.

6 Conclusion

This paper proposed an approach whereby a data stream is monitored continuously to detect any occurrence of a concept shift. To prevent the mining of frequent itemsets at every time point, the proposed data structures are maintained incrementally to monitor the changing of frequent itemsets. Additionally, the property of the power-law distribution of supports of itemsets is applied to estimate the approximate amount of newly generated frequent itemsets. The mining algorithm is required to discover a complete set of new frequent itemsets only when a significant change is observed.

Two algorithms, the FCDT and the FCDB, are proposed to monitor the changing ratios of frequent itemsets over a data stream based on a sliding window updated per transaction and updated per batch, respectively. The experimental results show that the FCDT and the FCDB detect concept shifts of frequent itemsets both effectively and efficiently. The execution time of the FCDT and the FCDB is significantly less than the time required to mine frequent itemsets over a sliding window at each time point. To provide a more concise data structure for monitoring the changes of frequent itemsets; this issue is a good direction for our future studies.

References

1. Agrawal, R. Srikant, R.: Fast Algorithms for Mining Association Rules in Large Databases. In VLDB, (1994).
2. Chang, J. H. Lee, W. S.: A Sliding Window Method for Finding Recently Frequent Itemsets over Online Data Streams. In Journal of Information Science and Engineering, vol. 20, no. 4, (2004)
3. Cheng, J. Ke, Y. Ng, W.: A Survey on Algorithms for Mining Frequent Itemsets over Data Streams. In Knowledge and Information Systems, vol. 16, no. 1, pp. 1--27 (2008).
4. Chuang, K.-T., Huang, J. L. Chen, M. S.: On Exploring the Power-Law Relationship in the Itemset Support Distribution. In Proceedings of the 10th International Conference on Extending Database Technology (EDBT), (2006).
5. Goethals, B. Zaki, M. FIMI'03, Frequent Itemset Mining Implementations. In Proc. of the ICDM 2003 Workshop on Frequent Itemset Mining Implementations (2003).
6. Han, J. Pei, J. Yin, Y.: Mining Frequent Patterns without Candidate Generation. In SIGMOD, (2000).
7. Kifer, D. Ben-David, S. Gehrke, J.: Detecting Change in Data Streams. In Proceedings of the 30th International Conference on Very Large Database, (2004).
8. Lin, C.-Y. and Koh, J.-L.: Frequent Patterns Change Detection over Data Streams. Technique Report, Department of Information and Computer Education, NTNU, 2007.
9. Lin, C.H. Chiu, D. Y. Wu, Y.H. Chen, A.L.P.: Mining Frequent Itemsets from Data Streams with a Time-Sensitive Sliding Window. In Proc. of SIAM Intl. Conference on Data Mining, (2005).
10. Liu, B. Hsu, W. Ma, Y.: Integrating Classification and Association Rule Mining. In KDDM, (1998).
11. Manku, G. S. and Motwani, R.: Approximate Frequent Counts over Data Streams. In Proc. of the 28th International Conference on Very Large Database, (2002).
12. Mozafari, B. Thakkar, H. Zaniolo, C.: Verifying and Mining Frequent Patterns from Large Windows over Data Streams. In Proc. of ICDM, (2008).
13. Pei, J. Han, J. Mortazavi-As, B. Wang, J. Pinto, H. Chen, Q. Dayal, U. Hsu, M.: Mining Sequential Patterns by Pattern-Growth: The PrefixSpan Approach. In IEEE TKDE, vol. 16, no. 11, pp. 1424–1440 (2004).
14. Srikant, R. Agrawal, R.: Mining Sequential Patterns: Generalizations and Performance Improvements. In EDBT, (1996).
15. Wang, H. Yang, J. Wang, Yu, W. P. S.: Clustering by Pattern Similarity in Large Datasets. In Proc. of SIGMOD, (2002).