Available online at www.sciencedirect.com
ScienceDirect
Expert Systems with Applications 36 (2009) 885–893
Expert Systems with Applications
www.elsevier.com/locate/eswa

Efficient mining of temporal emerging itemsets from data streams

Chun-Jung Chu [a], Vincent S. Tseng [b,*], Tyne Liang [a]

[a] Department of Computer Science, National Chiao Tung University, Taiwan, ROC
[b] Department of Computer Science and Information Engineering, National Cheng Kung University, Taiwan, ROC

Abstract

In this paper, we propose a new method, namely *EFI-Mine*, for mining temporal emerging frequent itemsets from data streams efficiently and effectively. The temporal emerging frequent itemsets are those that are infrequent in the current time window of data stream but have high potential to become frequent in the subsequent time windows. Discovery of emerging frequent itemsets is an important process for mining interesting patterns like association rules from data streams. The novel contribution of *EFI-Mine* is that it can effectively identify the potential emerging itemsets such that the execution time can be reduced substantially in mining all frequent itemsets in data streams. This meets the critical requirements of time and space efficiency for mining data streams. The experimental results show that *EFI-Mine* can find the emerging frequent itemsets with high precision under different experimental conditions and it performs scalable in terms of execution time.

Keywords: Temporal emerging frequent itemsets; Data streams; Association rules

1. Introduction

The mining of association rules for finding the relationship between data items in large databases is a well studied technique in data mining field with representative methods like *A priori* (Agrawal, Imielinski, & Swami, 1993; Agrawal, Mannila, Srikant, Toivonen, & Verkamo, 1996; Brin, Motwani, Ullman, & Tsur, 1997). The problem of mining association rules can be decomposed into two steps. The first step involves finding all frequent itemsets (or say large itemsets) in databases. Once the frequent itemsets are found, generating association rules is straightforward and can be accomplished in linear time.

An important research issue extended from the association rules mining is the discovery of temporal association patterns in data streams due to the wide applications on various domains. Temporal data mining can be defined as the activity of looking for interesting correlations or patterns in large sets of temporal data accumulated for other purposes (Bettini, Wang, & Jajodia, 1996). For a database with a specified transaction window size, we may use the algorithm like A priori to obtain frequent itemsets from the database. For time-variant data streams, there is a strong demand to develop an efficient and effective method to mine various temporal patterns (Das, Lin, Mannila, Renganathan, & Smyth, 1998). However, most methods designed for the traditional databases cannot be directly applied for mining temporal patterns in data streams because of the high complexity.

Without loss of generality, consider a typical market-basket application as illustrated in Teng, Chen, and Yu (2003) has been considered. The transaction flow in such an application is shown in Fig. 1 where items *a* to *g* stand for items purchased by customers.

In Fig. 1, for example, the third customer bought item *c* during time $t = [0, 1)$, items *c*, *e* and *g* during $t = [2, 3)$, and item *g* during $t = [4, 5)$. It can be seen that in such a data stream environment it is intrinsically difficult to conduct the frequent pattern identification due to the limited time

* Corresponding author. Tel.: +886 6 2757575x62536; fax: +886 6 2747076.
E-mail addresses: cjchu@cis.nctu.edu.tw (C.-J. Chu), tsengsm@mail.ncku.edu.tw (V.S. Tseng), tliang@cis.nctu.edu.tw (T. Liang).

0957-4174/$ - see front matter
doi:10.1016/j.eswa.2007.10.040

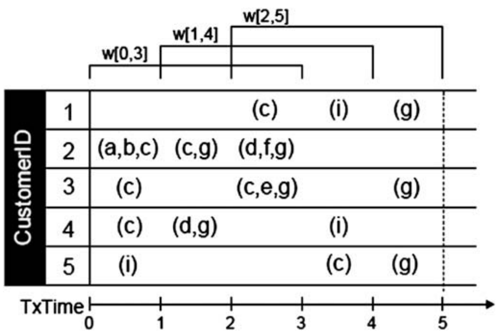

Fig. 1. An example of online transaction flows.

and space constraints. Furthermore, it wastes too much times finding frequent itemsets in different window times. Therefore, we develop a new scheme to find potential emerging frequent itemsets before next window times.

Dong and Li (1999) define an emerging pattern as an itemset the support of which increases significantly between two databases. We view emerging frequent itemsets as a special case of the emerging patterns described by Dong and Li. An *Emerging Frequent Itemset (EFI)* can be considered as an itemset that is infrequent (i.e., small) in the current database and gets increased for its support so that it will eventually become frequent (i.e., large) in the next database temporally added with new data transactions. For example, in the market basket domain, we may assume an interval as the time between wholesale purchases. Recognizing the set of items that will emerge or become frequent in the next time period with the size of the window may allow the storekeeper to order these emerging items much earlier than usual. Thus, the storekeeper will know what kinds of items will be popular in the next time period and avoid losing the income that their sales could have generated. Although some related issues like mining emerging frequent itemsets (Imberman, Tansel, & Pacuit, 2004) and incremental frequent itemsets (Cheung, Lee, & Kao, 1997b; Cheung, Han, Ng, & Wong, 1996b; Cheng, Yan, & Han, 2004; Parthasarathy, Zaki, Ogihara, & Dwarkadas, 1999) have been studied, they have been focused on traditional databases and are not suited for data streams.

In this paper, we explore the issue of efficiently mining emerging frequent itemsets in temporal databases like data streams (Lin, Chiu, Wu, & Chen, 2005; Li, Lee, & Shan, 2004, 2005; Jin, Qian, Sha, Yu, & Zhou, 2003). We propose an algorithm named *EFI-Mine* that can discover emerging frequent itemsets from data streams efficiently and effectively. The *EFI-Mine* algorithm is based on the concept of A priori algorithm (Agrawal et al., 1996) for mining frequent itemsets. The novel contribution of *EFI-Mine* is that it can effectively identify the potential emerging frequent itemsets in data streams so that the execution time for mining frequent itemsets can be substantially reduced. That is, *EFI-Mine* can discover the itemsets that are infrequent in current time window but will become frequent ones with high probability in subsegment time windows. In this way, the process of discovering all frequent itemsets under all time windows of data streams can be achieved efficiently with limited memory space. This meets the critical requirements of time and space efficiency for mining data streams. Through experimental evaluation, *EFI-Mine* is shown to deliver high precision in finding the emerging frequent itemsets and it also achieves high scalability in terms of execution time.

The rest of this paper is organized as follows: Section 2 gives the problem definition for mining temporal patterns and the emerging frequent itemsets. Section 3 describes the proposed approach, *EFI-Mine*, for finding the emerging frequent itemsets. In Section 4, we describe the experimental results for evaluating the proposed method. The conclusion of the paper is provided in Section 5.

2. Problem definitions

In this section, we first describe a support framework for mining of frequent temporal patterns, and given in Section 2.1. Then, the detail definition of emerging frequent itemset and interesting emerging itemsets are given in Section 2.2.

2.1. Support framework for mining temporal patterns

In this paper, the mining of temporal patterns are explored for illustrative purposes since not only the patterns should be efficiently and effectively extracted but also variations of corresponding occurrence frequencies should be tracked. In market-basket analysis, patterns along with their frequencies are extracted from a sliding window in transactions. So the data expires after a user-specified time window. As time advances, new data is included while obsolete data is discarded. With the mining task for discovering frequent temporal patterns, only patterns with occurrence frequencies no less than a specified threshold are being tracked. We focus in this paper on handling the different sliding windows to find emerging frequent itemsets.

An example showing the basic process in transforming transactions into numerical time series, for discovering frequent temporal patterns, is provided as follows.

Example 1. Consider the transaction flows shown in Fig. 1. Given the window size $w = 3$ and the minimum support value as 40%, occurrence frequencies of the inter-transaction itemset $\{c, g\}$ from time $t = 1$ to $t = 5$ can be obtained as shown in Table 1.

Table 1
The support values of the inter-transaction itemset $\{c, g\}$

T × Time		Occurrence(s) of $\{c, g\}$	Support
$t = 1$	$w[0, 1]$	None	0
$t = 2$	$w[0, 2]$	CustomerID = {2, 4}	$2/5 = 0.4$
$t = 3$	$w[0, 3]$	CustomerID = {2, 3, 4}	$3/5 = 0.6$
$t = 4$	$w[1, 4]$	CustomerID = {2, 3}	$2/5 = 0.4$
$t = 5$	$w[2, 5]$	CustomerID = {1, 3, 5}	$3/5 = 0.6$

With the sliding window model, the frequent temporal patterns can be discovered for different time windows. The main goal of our research is to discover interesting emerging itemsets under progressive time windows.

2.2. Emerging frequent itemsets and interesting emerging itemsets

In a database, the frequent itemsets will be changed when new datum is added. As time progresses, we can see many interesting patterns with regards to the change in status of individual itemsets. An itemset that was infrequent may become frequent (large), while frequent itemsets may become infrequent (small) and an itemset may remain frequent or infrequent. We define infrequent itemsets that are moving toward being frequent as *emerging*. Conversely, frequent itemsets moving toward infrequent are *submerging*. An infrequent (frequent) itemset that becomes large, i.e. with support above (below) minimum support value, is said to have emerged (submerged). The problems we address in this paper are: (1) How can we identify itemsets that are emerging (submerging)? (2) Which of these itemsets have the potential to emerge (submerge) within the next time window? That is, we focus on finding emerging frequent itemsets in this paper.

According to the emerging itemsets of incremental scheme, we develop this concept on the temporal data mining. Temporal data mining has the limitation on window size for finding emerging itemsets. Therefore, we must change the formula for finding emerging itemsets. For the remainder of this paper, we give definitions to the formula.

Definition 2.1. db_k is the transactions in $t = k$, i.e., db_1 is the transactions in $t = 1$.

Definition 2.2. $DB_{i,i+1,\ldots,j}$ is the transactions in $t = i$ to j, i.e., DB_{12345} is the transactions in $t = 1$–5. We also view DB_{12345} as the accumulation of $db_1 + db_2 + db_3 + db_4 + db_5$.

Suppose the original database is $DB_{i,i+1,\ldots,j}$ with window size = N and $N = j - i + 1$. Due to the limitation of window size, we should discard the old database db_i when adding a new database db_{j+1}. The new database should be $DB_{i+1,i+2,\ldots,j+1}$. In our scheme, we should find emerging itemsets before a new database is added. So we should focus on the database $DB_{i+1,i+2,\ldots,j}$. The old database db_i is useless for finding emerging itemsets. For example, suppose an original database is DB_{1234} and we set the limitation of window size as 5. If a database db_5 is added, the new database will be DB_{12345}. Due to the limitation of window size, when adding a database db_6, we should discard the old database db_1. Thus, the new database becomes DB_{23456}. In our scheme, we would find potential emerging frequent itemsets before a database is added. So we should focus on the database DB_{2345} finding potential emerging frequent itemsets. And the potential emerging frequent itemsets of the database DB_{2345} can be represented more accurate in the new database DB_{23456}. In practice, with the feature of data streams, we first remove db_1 from DB_{1234} and then add db_5 to form the database DB_{2345}. So we should find potential emerging frequent itemsets from the database DB_{2345} before adding a new database db_6 to form DB_{23456} and this conforms the limitation of window size. Fig. 2 shows that we would find potential emerging frequent itemsets from the database DB_{2345}. So the window size should be $N - 1$ for finding potential emerging itemsets.

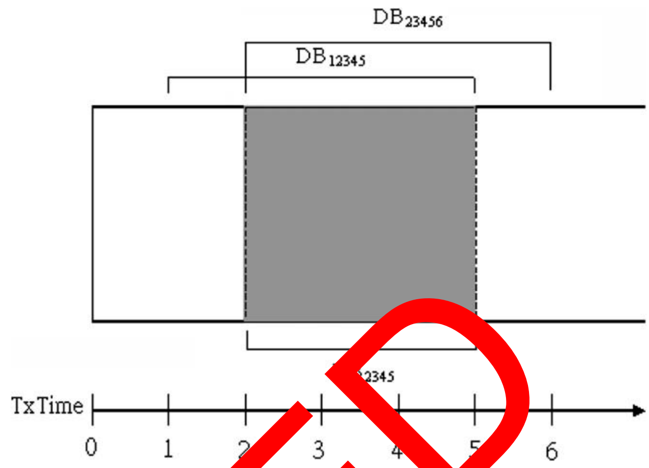

Fig. 2. Potential emerging frequent itemsets in DB_{2345}.

3. Mining temporal emerging itemsets

In Section 3.1, we give an example for mining temporal emerging itemsets from data stream. The proposed algorithm, *EFI-Mine*, is described in details in Section 3.2.

3.1. An example for mining emerging itemsets

Fig. 3 shows an example of emerging itemsets modified on that proposed by Dong and Li (1999) for the special case of EFI. It shows partitions of the space of itemsets, indicating all possible transitions for an itemset X from original database DB to the new database DB + db.

Fig. 3 plots the support count in DB (denoted as SC_{DB}) against the support count in db (denoted as SC_{db}). Each point in the graph depicts an ordered pair (SC_{db}, SC_{DB}) where the sum of SC_{db} and SC_{DB} is an itemset's support count in DB + db at some increment interval. If the increment adds no transactions to an itemset's support count, then its support count in DB has to be equal to $minSC_{DB} + minSC_{db}$ in order to achieve $minSC_{DB+db}$. This corresponds to point H in Fig. 3. Alternatively, if an itemset's SC is equal to |db| in db, then its support in DB has to be some $SC = n$, where $n > 0$, and $n = minSC_{DB} + minSC_{db} - |db|$ for the itemset to be frequent. This is point C in Fig. 3. Line HC partitions the space of all itemsets in DB + db into frequent and infrequent. The shaded area in Fig. 3 represents all the frequent itemsets and it includes

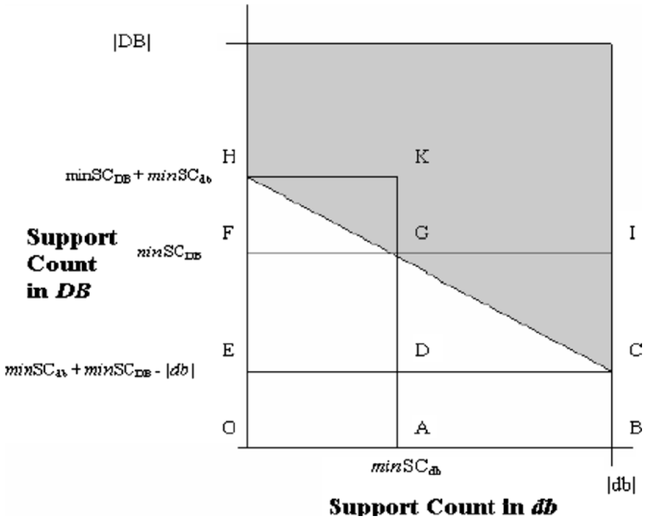

Fig. 3. Emerging frequent itemsets.

Line HC. Specific partitions under HC contain itemsets that are emerging in the current increment. For example, the area defined by ΔHFG represents those itemsets that were frequent itemsets in DB, infrequent itemsets in db, and now are infrequent in DB + db. These itemsets have therefore submerged. ΔGIC represents itemsets that were infrequent in DB and frequent in db. These itemsets have emerged. Therefore, we can find all itemsets in area ABCG are emerging in the current interval and all itemsets in area OAGH are submerging.

However, there are too many emerging itemsets in area ABCG. In fact, we should focus more potential emerging itemsets. To have the potential to emerge in the next increment, the support count of the itemset in DB + db needs to be greater than or equal to $2\text{minSC}_{db} + \text{minSC}_{DB} - |db|$ in the current increment. All points with this value are represented by line RS in Fig. 4.

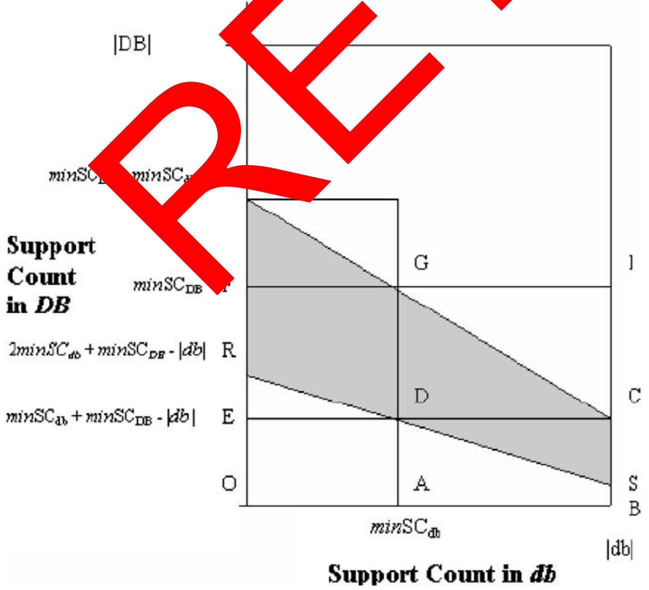

Fig. 4. Potentially emerging frequent itemsets.

For example, if we have a database with $|DB| = 10{,}000$, $|db| = 1000$ and minsup = 0.2, then the minimum support count for the current increment is 2200 (2000 from DB plus 200 from db). If an itemset can add the maximum support from incremental support count, a total of 1000 from db, in the next increment, it would need a support count of at least 1400 in the current increment to be able to attain the minimum support count of 2400 ((11,000 + 1000) * 0.2 = 2400) needed to become frequent.

The band of itemsets between line RS and line HC are all itemsets that have the potential to become frequent in the next increment, by this formula. Intersecting area ABCG and HCSR, we get itemsets in GDSC are most likely to emerge in the next increment.

3.2. Algorithm of EFI-Mine

With window size we mention in Section 2.2 and the concepts of emerging itemsets in Section 3.1, we set support value as S and assume the original database as $DB_{i,i+1,\ldots,j-1}$. According to the scheme we mentioned previously, if we want to find frequent itemsets from $DB_{i+1,i+2,\ldots,j+1}$, we should focus on $DB_{i+1,i+2,\ldots,j}$ for finding potential emerging frequent itemsets after adding database db_j and then find potential emerging frequent itemsets of the database $DB_{i+1,i+2,\ldots,j+1}$ before adding next incremental new database db_{j+1}. It means db_i would be an old database that needs not be considered. After adding new database db_{j+1}, the new database would be $DB_{i+1,i+2,\ldots,j+1}$. So the window size is N when database is changed from db_{i+1} to db_{j+1}. It also indicates $N = (j+1) - (i+1) + 1$. By the feature of temporal data mining, we set $|db| = |db_i| = |db_{i+1}| = \ldots = |db_j|$. In Fig. 4, various lines bear the following meaning:

$$LineHC = \text{minSC}_{DB_{i+1,i+2,\ldots,j-1}} + \text{minSC}_{db_j}$$

$$LineFI = \text{minSC}_{DB_{i+1,i+2,\ldots,j-1}}$$

$$LineRS = 2\text{minSC}_{db_j} + \text{minSC}_{DB_{i+1,i+2,\ldots,j-1}} - |db_j|$$

$$LineEC = \text{minSC}_{db_j} + \text{minSC}_{DB_{i+1,i+2,\ldots,j-1}} - |db_j|$$

$$LineAK = \text{minSC}_{db_j}$$

According to the feature of window size in temporal mining, incremental database means adding length of original transactions and also promoting the probability of infrequent itemsets to become frequent. Because we focus on $N-1$ window size for finding potential emerging frequent itemsets, these formulas should be divided by $N-1$ base on the number of database as follows:

$$LineHC = (\text{minSC}_{DB_{i+1,i+2,\ldots,j-1}} + \text{minSC}_{db_j})/N - 1$$

$$LineRS = (2\text{minSC}_{db_j} + \text{minSC}_{DB_{i+1,i+2,\ldots,j-1}} - |db_j|)/N - 1$$

Because Line FI does not add new database, it should be divided by $(N-1) - 1$. It means Line FI should be divided by $N-2$ as follows:

$$LineFI = \text{minSC}_{DB_{i+1,i+2,\ldots,j-1}}/N - 2$$

Line EC means that adding new database db_j and an itemset's SC is equal to $|db_j|$ in db_j, so it should be divided by $(N-1)$ as follows:

$$LineEC = (\mathrm{minSC}_{db_j} + \mathrm{minSC}_{DB_{i+1,i+2,\dots,j-1}} - |db_j|)/N - 1$$

Because db_j belongs to one of N window size, the formula should be divided by N as follows:

$$LineAK = \mathrm{minSC}_{db_j}/N$$

Fig. 5 illustrates the potentially emerging frequent itemsets in area GDSC with window size limitation. The formula for each line is as mentioned above.

According to these formulas, we can simplify these lines as follows:

$$\begin{aligned}
&\mathrm{HC} = [S^*(j-1-(i+1)+1)^*|db| + S^*|db|]/\\
&N-1 = [S^*(N-2)^*|db| + S^*|db|]/N-1 = S^*|db|\\
&\mathrm{FI} = [S^*(j-1-(i+1)+1)|db|]/N-2 = S^*|db|\\
&\mathrm{RS} = [2^*S^*|db| + S^*[(j-1)-(i+1)+1]^*|db| - |db|]/\\
&N-1 = [2^*S^*|db| + S^*(N-2)^*|db| - |db|]/\\
&N-1 = [(S^*N)-1]^*|db|/N-1\\
&\mathrm{EC} = [S^*|db| + S^*[(j-1)-(i+1)+1]^*|db| - |db|]/\\
&N-1 = [S^*|db| + S^*(N-2)^*|db| - |db|]/\\
&N-1 = [S^*(N-1)-1]^*|db|/N-1\\
&\mathrm{AK} = S^*db/N
\end{aligned}$$

We can also find potentially emerging frequent itemsets in area HRSC without concerning support count in [illegible]. However, it will reduce the accuracy with potentially emerging frequent itemsets. Taking into consideration of db_j would get the trend of itemsets and get better accuracy with potentially emerging frequent itemsets. Therefore, itemsets in GDSC are most likely to emerge in the next increment.

Fig. 6 shows the algorithm of *EFI-Mine* and the processing procedure is outlined below. The basic processing procedure is like A priori except the definition of for minimum support value for finding temporal emerging itemsets from data stream. With window size N, we would not only remove db_i but also add new database db_j for finding 1-emerging itemsets on the database $DB_{i+1,i+2,\dots,j}$ and finding large 1-itemsets on the database db_j from Step 1 to Step 3. So the purpose is to find potential emerging frequent itemsets of the database $DB_{i+1,i+2,\dots,j+1}$ before adding next new database db_{j+1}. We generate k-candidates and find k-emerging itemsets by calculating support count as mentioned previously from Step 4 to Step 13. Then, we generate k-candidates and find k-large itemsets by support count we mention from Step 14 to Step [illegible]. Finally, those itemsets meeting the constraints $S^*|db| >$ c.count $\geqq [(S^*N)-1]^*|db|/N-1$ on $DB_{i+1,i+2,\dots,j}$ and c.count $\geqq S^*db/N$ db_j are obtained as the potentially emerging frequent itemsets.

We may utilize the formulas mentioned before to discuss the following situations. Notice that an itemset is emerging or not depends on support count of the itemset. Given an itemset whose support counts in $DB_{i+1,i+2,\dots,j-1}$ and $DB_{i+1,i+2,\dots,j-1+db_j}$ are $SC_{DB_{i+1,i+2,\dots,j-1}}$ and $SC_{DB_{i+1,i+2,\dots,j-1}+db_j}$, respectively, the growth rate of that itemset is $SC_{DB_{i+1,i+2,\dots,j-1}+db_j} - SC_{DB_{i+1,i+2,\dots,j-1}}$. The growth rate of an itemset that maintains minimal support is $\mathrm{minSC}_{DB_{i+1,i+2,\dots,j-1}+db_j} - \mathrm{minSC}_{DB_{i+1,i+2,\dots,j-1}}$. An itemset meeting the $\frac{SC_{DB_{i+1,i+2,\dots,j-1}+db_j} - SC_{DB_{i+1,i+2,\dots,j-1}}}{\mathrm{minSC}_{DB_{i+1,i+2,\dots,j-1}+db_j} - \mathrm{minSC}_{DB_{i+1,i+2,\dots,j-1}}} > 1$ is an emerging itemset. An itemset needs a support count of at least $\mathrm{minSC}_{DB_{i+1,i+2,\dots,j-1}+db_j+db_{j+1}} = \mathrm{minSC}_{DB_{i+1,i+2,\dots,j-1}+2db}$ to emerge [illegible]dding a new database db_{j+1} with expanding one window size. A potential emerging frequent itemset is the one that is emerging and meets the following constraint: $SC_{DB_{i+1,i+2,\dots,j-1}+db_j} + (SC_{DB_{i+1,i+2,\dots,j-1}+db_j} - SC_{DB_{i+1,i+2,\dots,j-1}}) > \mathrm{minSC}_{DB_{i+1,i+2,\dots,j-1}+2db}$. Hence, we can infer that an itemset that will potentially emerge with expanding n window sizes is an itemset that is currently emerging and $SC_{DB_{i+1,i+2,\dots,j-1}+db_j} + n(SC_{DB_{i+1,i+2,\dots,j-1}+db_j} - SC_{DB_{i+1,i+2,\dots,j-1}}) > \mathrm{minSC}_{DB_{i+1,i+2,\dots,j-1}+ndb}$. Of course, the larger n is, the less accurate with finding potential emerging frequent itemsets might be.

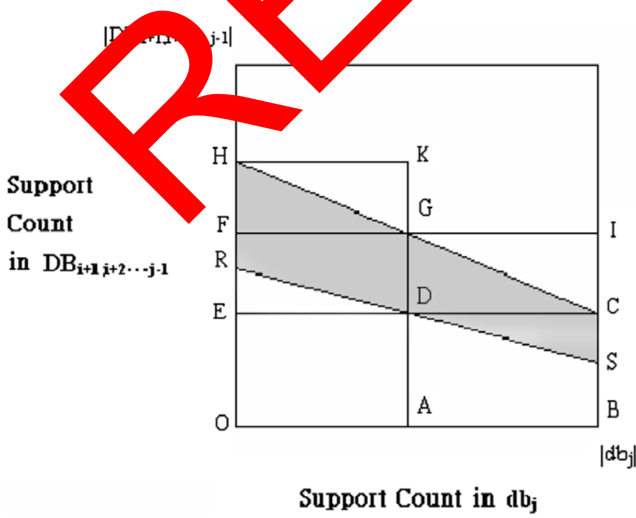

Fig. 5. Potentially emerging frequent itemsets for temporal patterns.

4. Experimental evaluation

To evaluate the performance of *EFI-Mine*, we conducted experiments of using synthetic dataset generated via a randomized transaction generation algorithm in Agrawal and Srikant (1995). The synthetic data generation program takes the parameters as shown in Table 2, and the values of parameters used to generate the datasets are shown in Table 3. The simulation is implemented in C++ and conducted in a machine with 1.4 GHz CPU and 512 MB memory. The main performance metrices used are execution time and accuracy. We recorded the execution time that *EFI-Mine* spends in finding potential emerging frequent itemsets. The accuracy is to measure the number of actual emerging frequent itemset in ratio of the total potential emerging frequent itemsets that we found. Hence, the accuracy is defined as follows:

```
Input: DB_{i,i+1,...,j-1}
Remove db_i from DB_{i,i+1,...,j-1} and add new database db_j then the database becomes DB_{i+1,i+2,...,j}
E_1 = {emerging 1-itemsets on database DB_{i+1,i+2,...,j}} and L_1={large 1-itemsets on database db_j};
for ( k = 2; E_{k-1} ≠ 0 k++ ) do begin
    C_k = candidate-gen(C_{k-1}); // New candidates
    forall transactions t ∈ DB_{i+1,...,j} do begin
        C_t = subset(C_t , t); // Candidates contained in transactions
        forall candidates c ∈ C_t do
            c.count++;
        end
        E_k = {c ∈ C_k | S*|db|> c.count ≧ [(S*N)-1]*|db|/N-1 }
        // SC between LineHC and LineRS
end
for ( k = 2; L_{k-1} ≠ 0 k++ ) do begin
    C_k = candidate-gen(C_{k-1}); // New candidates
    forall transactions t ∈ db_j do begin
        C_t = subset(C_t , t); // Candidates contained in transactions
        forall candidates c ∈ C_t do
            c.count++;
        end
        L_k = {c ∈ C_k | c.count ≧ S*db/N }
        // SC larger than LineAK
end
Output: ( ∪_k E_k ) ∩ L_k;
```

Fig. 6. Algorithm of *EFI-Mine*.

Table 2
Parameters of the synthetic datasets

N	Number of items
T	Average numbers of items per transaction
C	Number of customers
D	Number of transactions
W	Windows size
S	Support value

Table 3
Parameter settings of synthetic datasets

Dataset parameters	*N*	*T*	*C*	*D*	*W*
N100T5C1000	100	5	1000	100,000	10

$$\text{Accuracy} = (\text{number of actual emerging frequent itemset}) / (\text{total potential emerging frequent itemsets})$$

4.1. Effects of varying support threshold

In this experiment, we vary the values of support threshold from 30% to 70% for interesting the effects on the accuracy. The other parameters were kept fixed as default values. Fig. 7 shows the accuracy of *EFI-Mine* under different support threshold values. It is observed that the average accuracy of potential emerging frequent itemsets raises as the support value is increased. Especially, the accuracy reaches to 100% when the support value is beyond 60%. Hence, *EFI-Mine* is verified to be very effective in finding the emerging itemsets.

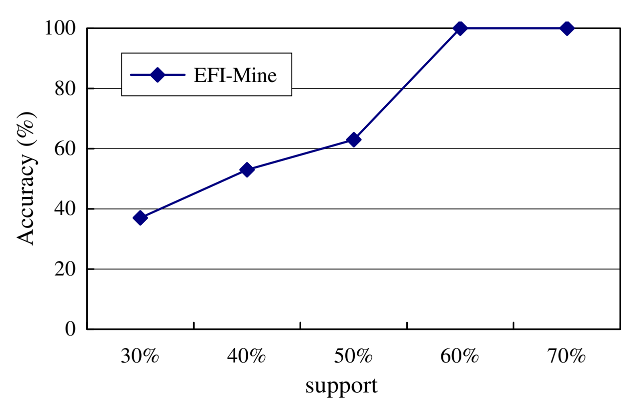

Fig. 7. Accuracy under different support values (N100T5C1000, $w = 10$).

4.2. Comparisons with A priori in execution time

In this experiment, we compare the average execution time in different support values between A priori and *EFI-Mine*. Both of these two algorithms could find frequent itemsets. However, A priori can only find frequent itemsets, while *EFI-Mine* can find frequent itemsets that were infrequent in the past. A priori algorithm processes $DB_{i+1,i+2,\ldots,j+1}$ to find frequent itemsets, while our *EFI-Mine* algorithm needs to process fewer database $DB_{i+1,i+2,\ldots,j}$ to find potentially emerging frequent itemsets. From Fig. 8, *EFI-Mine* spends few seconds with high stability for finding potentially emerging frequent itemsets. Compared to A priori, the improvement is about 90.6% for support values varied from 30% to 60%. Although *EFI-Mine* does not always obtain frequent itemsets with 100% accuracy, it reduces substantially the time in finding frequent itemsets. Moreover, the frequent itemsets obtained by A priori are not suitable for applications in data streams since we need frequent itemsets which are infrequent in the past and frequent in the current by the time change. Hence, *EFI-Mine* meets the requirements of high efficiency and high scalability in terms of execution time for data stream mining.

4.3. Effects of varying window size

In this experiment, we investigate the effects of varying window size on the accuracy of mining results. As shown in Fig. 9, we could observe that the larger window size, the higher with accuracy. In fact, the accuracy is almost 100% when window size is large than 15 in the experiments. This is because the itemsets are tended to be stable according to the past databases. This indicates that *EFI-Mine* fits for mining data streams with large window size.

4.4. Effects of varying transaction size

In this experiment, we investigate the effects of varying transaction size on the accuracy of mining results i.e., the average number of items per transaction. As shown in Fig. 10, if T is larger, the accuracy is higher than under T. This is because T carrying more information and trend from past transactions. This indicates that *EFI-Mine* fits for mining data streams with large transaction size.

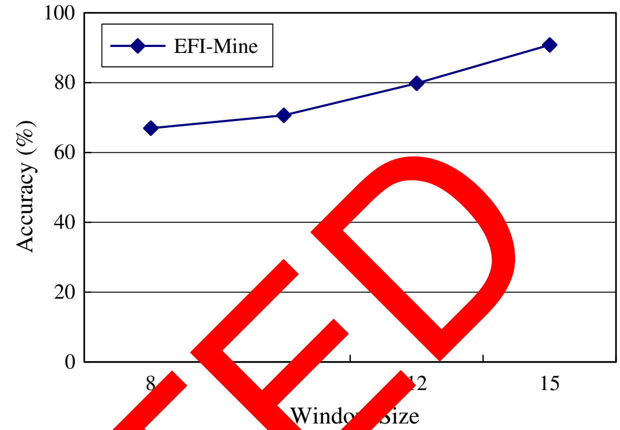

Fig. 9. Accuracy under different window sizes.

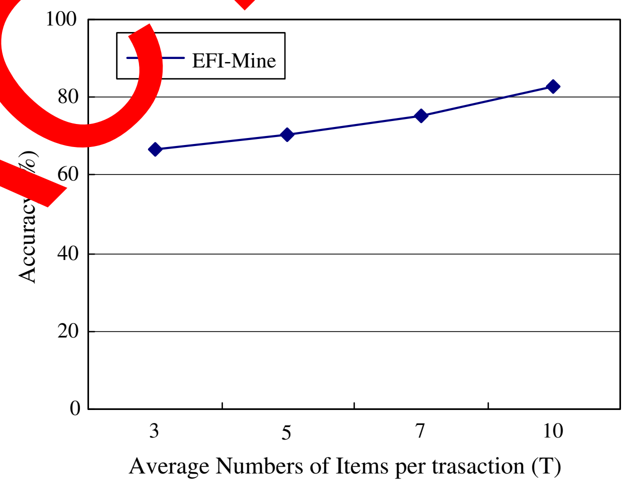

Fig. 10. Accuracy under different numbers of items per transaction with $w = 10$.

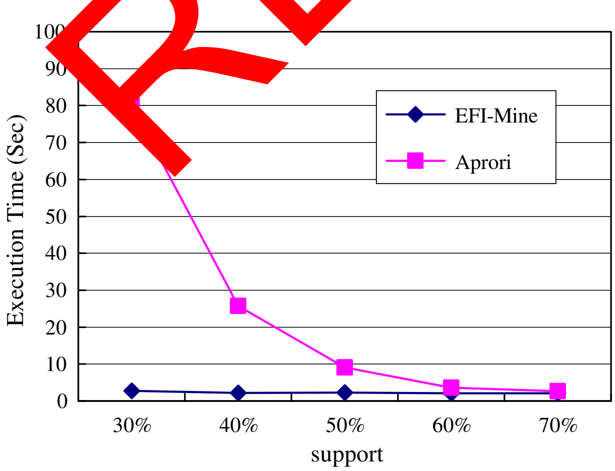

Fig. 8. Execution time with $w = 10$.

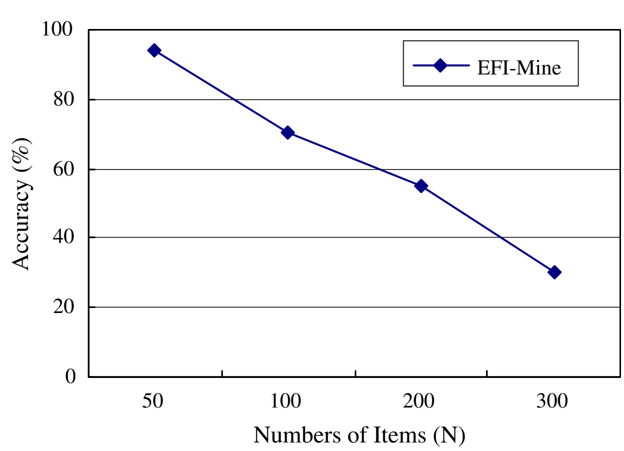

Fig. 11. Accuracy under different numbers of items.

4.5. Effects of varying number of items

In this last experiment, we investigate the effects of varying the numbers of items on the accuracy of mining results. The results are as shown in Fig. 11. We observe that the accuracy decreases when the numbers of items are increased. This is because too many items will affect the stability of the patterns. On the contrary, the accuracy under smaller numbers of items could reach almost 100%. This indicates that *EFI-Mine* fits for mining data streams with small numbers of items.

5. Related work

In association rules mining, A priori (Agrawal et al., 1993), DHP (Park, Chen, & Yu, 1997), and partition-based ones (Lin & Dunham, 1998; Savasere, Omiecinski, & Navathe, 1995) are proposed to find frequent itemsets. Many important applications have called for the need of incremental mining. This is due to the increasing use of the record-based databases whose data are being continuously added. Many algorithms like FUP (Cheung, Han, Ng, & Wong, 1996a), FUP_2 (Cheung, Lee, & Kao, 1997a) and UWEP (Ayn, Tansel, & Arun, 1999a, 1999b) are proposed to solve incremental database for finding frequent itemsets. The FUP algorithm updates the association rules in a database when new transactions are added to the database. Algorithm FUP is based on the framework of A priori and is designed to discover the new frequent itemsets iteratively. The idea is to store the counts of all the frequent itemsets found in a previous mining operation. Using these stored counts and examining the newly added transactions, the overall count of these candidate itemsets are then obtained by scanning the original database. An extension to the work in Cheung et al. (1996a) was reported in Cheung et al. (1997a) where the authors propose an algorithm FUP_2 for updating the existing association rules when transactions are added to and deleted from the database. UWEP (update with early pruning) is an efficient incremental algorithm, that counts the original database at most once, and the increment exactly once. In addition the number of candidates generated and counted is minimum.

In recent years, processing data from data streams is a very popular topic in data mining. Many algorithms like FTP-DS (Teng et al., 2003) and RAM-DS (Teng, Chen, & Yu, 2004) are proposed to process data in data streams. FTP-DS is a regression-based algorithm to mine frequent temporal patterns for data streams. A wavelet-based algorithm, called algorithm RAM-DS, to perform pattern mining tasks for data streams by exploring both temporal and support count granularities.

Some algorithms like SWF (Lee, Lin, & Chen, 2001) and Moment (Chi, Wang, Yu, & Muntz, 2004) are proposed to find frequent itemsets over a stream sliding window. By partitioning a transaction database into several partitions, algorithm SWF employs a filtering threshold in each partition to deal with the candidate itemset generation. Moment algorithm use the closed enumeration tree (CET), to maintain a dynamically selected set of itemsets over a sliding window.

Dong and Li define an emerging pattern as an itemset the support of which increases significantly between two databases. We view emerging frequent itemsets as a special case of the emerging patterns described by Dong and Li. Recently, a new algorithm modifies an existing incremental algorithm, UWEP, so that it can identify emergent large itemsets. It uses incremental scheme for finding emerging frequent itemsets (Imberman et al., 2004).

Although there existed numerous studies on frequent itemsets mining and data stream analysis as described above, there is no algorithm proposed for finding emerging frequent itemsets in data streams. This motivates our exploration on the issue of efficiently mining emerging frequent itemsets in temporal databases like data streams in this research.

6. Conclusions

In this paper, we addressed the problem of discovering temporal emerging itemsets in data streams, i.e., the itemsets that are infrequent in current time window but have the high potential to become frequent in the subsequent time windows. We propose a new approach, namely *EFI-Mine*, which can discover emerging frequent itemsets from data streams efficiently and effectively. The novel contribution of *EFI-Mine* is that it can effectively identify the potential emerging itemsets such that the execution time can be reduced substantially in mining all frequent itemsets in data streams.

The experimental results show that *EFI-Mine* can find the emerging frequent itemsets with high precision under different conditions like varied window size, transaction size and number of items, etc. This also indicates that *EFI-Mine* fits for mining data streams with large window size transaction size and number of items. Moreover, it is highly efficient and scalable in terms of execution time. Hence, *EFI-Mine* promising for mining temporal emerging patterns in data streams. For the future work, we would extend the concepts of this paper to discover other interesting patterns in data streams like the frequent closed sets (Bastide, Taouil, Pasquier, Stumme, & Lakhal, 2000; Pasquier, Bastide, Taouil, & Lakhal, 1999; Pei, Han, & Mao, 2000).

References

Agrawal, R., & Srikant, R. (1995). Mining sequential patterns. In *Proceedings of the 11th international conference on data engineering* (pp. 3–14), March.

Agrawal, R., Imielinski, T., & Swami, A. (1993). Mining association rules between sets of items in large databases. In *Proceedings of 1993 ACM SIGMOD international conference on management of data* (pp. 207–216). Washington, DC, May.

Agrawal, R., Mannila, H., Srikant, R., Toivonen, H., & Verkamo, A. I. (1996). Fast discovery of association rules. In U. Fayyad, G. Piatetsky-

Shapiro, P. Smyth, & R. Uthurusamy (Eds.), *Advances in knowledge discovery and data mining* (pp. 307–328). AAAI/MIT Press.

Ayn, N. F., Tansel, A. U., & Arun, E. (1999a). An efficient algorithm to update large itemsets with early pruning. Technical Report BU-CEIS-9908 Dept. of CEIS Bilkent University, June.

Ayn, N. F., Tansel, A. U., & Arun, E. (1999b). An efficient algorithm to update large itemsets with early pruning. In *Proceedings of the fifth ACM SIGKDD international conference on knowledge discovery and data mining*, San Diego, August.

Bastide, Y., Taouil, R., Pasquier, N., Stumme, G., & Lakhal, L. (2000). Mining frequent patterns with counting inference. *SIGKDD Explorations, 2*(2).

Bettini, C., Wang, X. S., & Jajodia, S. (1996). Testing complex temporal relationships involving multiple granularities and its application to data mining. In *Proceedings of the 15th ACM SIGACT-SIGMOD-SIGART symposium on principles of database systems, June 3–5, 1996* (pp. 68–78). Montreal, Canada: ACM Press.

Brin, S., Motwani, R., Ullman, J., & Tsur, S. (1997). Dynamic itemset counting and implication rules for market basket data. In *Proceedings ACM SIGMOD conference management of data*, May.

Cheng, H., Yan, X., & Han, J. (2004). IncSpan: Incremental mining of sequential patterns in large database. In *Proceedings 2004 international conference on knowledge discovery and data mining (KDD'04)*, Seattle, WA, August.

Cheung, D. W.-L., Han, J., Ng, V., & Wong, C. Y. (1996a). Maintenance of discovered association rules in large databases: An incremental updating technique. In *Proceedings of 1996 international conference on data engineering* (pp. 106–114), February.

Cheung, D. W.-L., Han, J., Ng, V. T., & Wong, C. Y. (1996b). Maintenance of discovered association rules in large databases: An incremental update technique. In *Proceedings of international conference on data engineering (ICDE'96)*, New Orleans, Louisiana, February.

Cheung, D., Lee, S. D., & Kao, B. (1997a). A general incremental technique for updating discovered association rules. In *Proceedings international conference on database systems for advanced applications*, April.

Cheung, D. W.-L., Lee, S. D., & Kao, B. (1997b). A general incremental technique for maintaining discovered association rules. In *Proceedings of the 5th international conference on database systems for advanced applications (DASFAA'97)*, Melbourne, Australia, April.

Chi, Y., Wang, H., Yu, P. S., & Muntz, R. R. (2004). Moment: Maintaining closed frequent itemsets over a stream sliding window. In *Proceedings of the 2004 IEEE International Conference on Data Mining (ICDM'04)*.

Das, G., Lin, K.-I., Mannila, H., Renganathan, G., & Smyth, P. (1998). Rule discovery from time series. In *Proceedings of the 4th ACM SIGKDD* (pp. 16–22), August.

Dong, G., & Li, J. (1999). Efficient mining of emerging patterns: Discovering trends and differences. In *Proceedings of the fifth ACM SIGKDD international conference on knowledge discovery and data mining*, San Diego, August.

Imberman, S. P., Tansel, A. U., & Pacuit, E. (2004). An efficient method for finding emerging frequent itemsets. In *third International workshop on mining temporal and sequential data* (pp. 112–121), August.

Jin, C., Qian, W., Sha, C., Yu, J. X., & Zhou, A. (2003). Dynamically maintaining frequent items over a data stream. In *Proceedings of the 2003 ACM CIKM international conference on information and knowledge management*, November.

Lee, C.-H., Lin, C.-R., & Chen, M.-S. (2001). Sliding-window filtering: An efficient algorithm for incremental mining CIKM (pp. 263–270).

Li, H.-F., Lee, S.-Y., & Shan, M.-K. (2004). An efficient algorithm for mining frequent itemsets over the entire history of data streams. In *Proceedings of first international workshop on knowledge discovery in data streams*, Pisa, Italy.

Li, H.-F., Lee, S.-Y., & Shan, M.-K. (2005). Online mining (recently) maximal frequent itemsets over data streams. In *Proceedings of the 15th IEEE international workshop on research issues on data engineering*, Tokyo, Japan, April.

Lin, J.-L., & Dunham, M. H. (1998). Mining association rules: Anti-skew algorithms. In *Proceedings of 1998 international conference on data engineering* (pp. 486–493).

Lin, C. H., Chiu, D. Y., Wu, Y. H., & Chen, A. L. P. (2005). Mining frequent itemsets from data streams with a time-sensitive sliding window. In *Proceedings of 2005 SIAM international conference on data mining*, CA, April.

Park, J.-S., Chen, M.-S., & Yu, P. S. (1997). Using a hash-based method with transaction trimming for mining association rules. *IEEE Transactions on Knowledge and Data Engineering, 9*(5), 813–825.

Parthasarathy, S., Zaki, M., Ogihara, M., & Dwarkadas, S. (1999). Incremental and interactive sequence mining. In *Proceedings of the 8th international conference on information and knowledge management (CIKM'99)*, November.

Pasquier, N., Bastide, Y., Taouil, R., & Lakhal, L. (1999). Discovering frequent closed itemsets for association rules. In *Proceedings seventh international conference database theory*, January.

Pei, J., Han, J., & Mao, R. (2000). Closet: An efficient algorithm for mining frequent closed itemsets. In *Proceedings SIGMOD international workshop data mining and knowledge discovery*, May.

Savasere, A., Omiecinski, E., & Navathe, S. (1995). An efficient algorithm for mining association rules in large databases. In *Proceedings of the 21th international conference on very large data bases* (pp. 432–444), September.

Teng, W.-G., Chen, M.-S., & Yu, P. S. (2003). A regression-based temporal pattern mining scheme for data streams. In *Proceedings of the 29th international conference on very large data bases* (pp. 93–104), September.

Teng, W.-G., Chen, M.-S., & Yu, P. S. (2004). Resource-aware mining with variable granularities in data streams. In *SDM 200*.