Incrementally Mining Recently Repeating Patterns over Data Streams*

Jia-Ling Koh and Pei-Min Chou

Department of Information Science and Computer Engineering
National Taiwan Normal University
Taipei, Taiwan 106, R.O.C
Email: jlkoh@csie.ntnu.edu.tw

Abstract. Repeating patterns represent temporal relations among data items, which could be used for data summarization and data prediction. More and more data of various applications is generated as a data stream. Based on time sensitive concern, mining repeating patterns from the whole history data sequence of a data stream does not extract the current trend of patterns in the stream. Therefore, the traditional strategies for mining repeating patterns on static database are not applicable to data streams. For this reason, an algorithm, named appearing-bit-sequence-based incremental mining algorithm, for efficiently discovering recently repeating patterns from a data stream is proposed in this paper. The appearing bit sequences are used to monitor the occurrences of patterns within a sliding window. Two versions of algorithms are proposed by maintaining the appearing bit sequences of maximum repeating patterns and closed repeating patterns, respectively. Accordingly, the cost of re-mining repeating patterns over a sliding window is reduced to that of monitoring frequency changes of the maintained patterns. The experimental results show that the incremental mining methods perform much better than the re-miming approach.

Keyword. repeating patterns, incremental mining, data streams.

1. Introduction

Repeating patterns represent temporal relations among data items in a data sequence, which could be used for data summarization and data prediction. Therefore, repeating patterns discovery is of great interest for applications with sequential data representations. There have been many approaches proposed for mining repeating patterns [5][6][7][10]. The concept of repeating patterns was used in [5] to represent the significant content of a music object. It proposed a data structure called *correlative matrix* to aid the process for extracting repeating patterns. Moreover, for providing more efficient processing, the *String-Join* algorithm was developed to extract the *non-trivial* repeating patterns extracted instead of the repeating patterns in a music object. In [6], we designed *bit index sequences* to characterize note sequences of music objects. In the mining process, the frequency of a candidate pattern was obtained by performing **shift** and **and** operations on bit sequences and counting the number of 1s in the resultant bit sequence. Therefore, frequency checking of a pattern could be performed quickly. In addition, this approach avoided scanning the data sequences repeatedly. Fault-tolerant data mining would discover more general and

This work was partially supported by the R.O.C. N.S.C. under Contract No. 96-2221-E-003-018 and 96-2524-S-003-001.

useful information for real-world dirty data. By extending the idea of appearing bit sequences, fault-tolerant appearing bit sequences were defined in [7] to represent the locations where candidate patterns appear in a data sequence with insertion/deletion errors being allowed. Accordingly, the fault-tolerant frequency of each candidate pattern could be obtained quickly.

Recently, the data stream, which is an unbounded sequence of data elements generated at a rapid rate, provides a dynamic environment for collecting data. Knowledge discovery on streaming data is a research topic of growing interest especially for learning concept drifts [4][12]. Lossy-counting is the representative approach for mining frequent itemsets over data streams [11]. For reducing the required memory usage, the Lossy-counting algorithm only maintains patterns with support being no less than an error tolerance parameter ε. Consequently, the frequency of a pattern is estimated by compensating the maximum number of times that the pattern could have occurred before being monitored. Time sensitivity is another important issue when mining frequent itemsets on data streams. It is likely that the embedded knowledge in a data stream will change quickly as time goes by. In order to catch the recent trend of data, the *estDec* algorithm [2] decayed the old occurrences of each itemset to diminish the effect of old transactions on the mining result of frequent itemsets in the data steam. However, in particular applications, it is interested only the frequent patterns mined from the recently arriving data within a fixed time period. A time-sensitive sliding window approach was proposed in [9] for mining recently frequent itemsets within the current sliding window in a data stream. Accordingly, the recently frequent itemsets were discovered from the most recent w blocks of transactions. For discovering closed frequent itemsets in a sliding window, [3] proposed a compact data structure, named a closed enumeration tree (CET), to monitor the itemsets which consist of the boundary between closed frequent itemsets and the rest of the itemsets. Accordingly, the closed frequent itemsets in a sliding window are discovered by maintaining the CET structure incrementally. For achieving the similar purpose, a data structure called summary frequent itemset forest was introduced in [8] for incremental maintaining the essential information about maximal frequent itemsets embedded in a data stream.

Although the problem of mining frequent itemsets over data streams has been investigated in the above literatures [2][3][8][9][11], the temporal relations among data items were not considered in these studies. Accordingly, it is essential to provide a data structure for maintaining sequential information of items within a sliding window to discover recently repeating patterns in the window. Moreover, an incremental pattern mining strategy is necessary to avoid performing re-mining when the window slides. In this paper, an algorithm, named appearing-bit-sequence-based incremental mining algorithm, for efficiently mining recently repeating patterns over a data stream is proposed. The appearing bit sequences are used to monitor the occurrences of patterns within a sliding window. Two versions of algorithms are proposed by maintaining the appearing bit sequences of maximum repeating patterns and closed repeating patterns, respectively. Accordingly, the cost of mining repeating patterns in a sliding window is reduced to that of mining frequency changes of the maintained patterns. The experimental results show that the incremental mining methods perform much better than the re-miming approach.

This paper is organized as follows. The related terms used in this paper are defined in Section 2 first. In Section 3, the proposed algorithm for discovering recently repeating patterns incrementally from the maintained structure is introduced. The performance evaluation on the proposed algorithms is reported in Section 4. Finally, in Section 5, we conclude this paper.

2. Preliminaries

2.1 Problem Definition

Let $I = \{I_1, I_2, I_3 \ldots\ldots I_m\}$ denote the set of data items in the specific application domain. Suppose an item in I is inputted at each time point. Accordingly, a data stream $DS = [S_1, S_2, S_3 \ldots)$ is formed, where each S_i denoting a data item in I is associated with an time identifier i. Under a predefined sliding window size W, the ***sequence window*** at time t, denoted as SW_t, represents the sequence of items $[S_{t-W+1}, S_{t-W+2}, \ldots, S_t]$. The ith item in SW_t is denoted as $SW_t[i]$. A pattern $P = p_1 p_2 \ldots p_k$ ($k \geq 1$) is a data sequence consisting of one or more items in I. The number of items in pattern P is called the length of P, denoted as $|P|$.

Suppose a sequence window SW_t and a pattern $P = p_1 p_2 \ldots p_m$ are given. If there exists an positive integer i such that $SW_t[i\text{-}m+1]SW_t[i\text{-}m+2]\ldots SW_t[i] = p_1 p_2 \ldots p_m$, it is called that p ***appears in*** SW_t and SW_t ***contains*** p on position i. The number of positions in SW_t, where SW_t contains p, is named the ***recent frequency*** of p in DS at time t, denoted as $RF_t^{DS}(p)$. Given a user specified minimum frequency, denoted as F_{min}, a pattern p is called a ***recently repeating patterns*** in DS at time t if $RF_t^{DS}(p) \geq F_{min}$. A recently repeating pattern with length k is named a *k-RRP*.

2.2 Appearing Bit Sequences

In our previous work [6], the ***appearing bit sequences*** were proposed to speed up the mining of repeating patterns in a data sequence with fixed length. For each kind of data item N in a data sequence, N has a corresponding *appearing bit sequence* (denoted as $Appear_N$). The length of an appearing bit sequence equals the length of the data sequence. The leftmost bit is numbered as bit 1. If some data item appears on the ith position of the data sequence, bit i in the appearance bit sequence of this data item is set to be 1; otherwise, it is set to be 0. A bit index table is used to store the appearing bit sequences for all the data items in the data sequence. Therefore, the frequency of a data item is obtained according to the number of bits with value 1 in its appearing bit sequence, without needing to scan the data sequence repeatedly.

[Example 1]

The bit index table of "BCBCABCABC" is given as shown in Figure 1.

1) Suppose we would like to get $Appear_{AB}$. A position i where "AB" appears implies "A" must appear on position i and "B" appears on the next position (i+1).
   Step1. Get $Appear_A$=0000100100 and $Appear_B$=1010010010 from Table 1.
   Step2. Perform one **right shift** operation on $Appear_A$ (shift bit i to bit(i+1), where $1 \leq i \leq 19$, and set bit 0 to be 0), $R\_shift(Appear_A, 1)$ = 0000010010.
   Step3. $Appear_{AB} = R\_shift(Appear_A, 1) \wedge Appear_B$ = 0000010010.

	B	C	B	C	A	B	C	A	B	C
***Appear**$_A$*	0	0	0	0	1	0	0	1	0	0
***Appear**$_B$*	1	0	1	0	0	1	0	0	1	0
***Appear**$_C$*	0	1	0	1	0	0	1	0	0	1

Figure 1: the bit index table for data sequence "BCBCABCABC."

2) Suppose we would like to get $Appear_{ABC}$ after getting $Appear_{AB}$. A position i where "ABC" appears implies "AB" must appears on position i and "C" appears on position i+1.

   Step1. Obtain $Appear_C$=0101001001 from Table 1.

   Step2. Perform one **right shift** operation on $Appear_{AB}$, $R\_shift(Appear_{AB},1)$ = 0000001001.

   Step3. $Appear_{ABC} = R\_shift(Appear_{AB}, 1) \wedge Appear_C$ = 0000001001.

Accordingly, the frequency of "ABC" in the sample data sequence equals the number of bits with value 1 in $Appear_{ABC}$ (that is 2 in this case).

Suppose pattern $P=P_1P_2\ldots P_m$, where P_i (i=1, …,m) is a data item and m>1. Let $P'=P_1P_2\ldots P_{m-1}$ and $X=P_m$. In general, $Appear_P$ is deduced from $Appear_{P'}$ and $Appear_X$ according to the following recursive formula:

$Appear_P = R\_shift(Appear_{P'},1) \wedge Appear_X$.

3. Incremental Repeating Patterns Mining

The processing of the incremental repeating patterns mining approach is characterized into two phases: window initialization phase and window sliding phase. The window initialization phase is activated at the time when the first sequence window becomes full. In this phase, the appearing information of repeating patterns in the *initial sequence window* is discovered and maintained. After that, the window sliding phase is activated. A newly coming data item is inserted and the oldest data item has to be removed from the sequence window. Without needing to re-perform repeating patterns mining on the new sequence window, the recently repeating patterns are discovered incrementally from the maintained information efficiently.

3.1 Window Initialization Phase

[Def. 1] Prefix-subpattern and suffix-subpattern
Given a pattern $P=x_1x_2\ldots x_k$. For any pattern $P_1= x_1x_2\ldots x_i$, where $i<k$, P_1 is named an *i-prefix-subpattern* of P. Pattern $x_1x_2\ldots x_{k-1}$ is the *maximum prefix-subpattern* of P. For any pattern $P_2= x_jx_{j+1}\ldots x_k$, where $1<j$, P_2 is named a (k-j+1)-*suffix-subpattern* of P. Pattern $x_2x_3\ldots x_k$ is the *maximum suffix-subpattern* of P. All the prefix and suffix-subpatterns of P are named *sub-patterns* of P, and P is a *super-pattern* of its sub-patterns.

[Def. 2] maximum repeating pattern
A *maximum repeating pattern* is defined as a recent repeating pattern for which none of its super-patterns are recently repeating patterns.

In the window initialization phase, a bit index table, named a *recent bit index table*, is constructed to represent the initial sequence window. An Apriori-like algorithm is developed to extract recently repeating patterns from the initial sequence window, in which appearing bit sequences of patterns are used to count the recent frequencies of candidate patterns efficiently. For providing incremental mining in the window sliding phase, the discovered repeating patterns and their corresponding appearing bit sequences are maintained. In order to reduce the storage space of the maintained patterns, only the maximum repeating patterns are remained.

First, the initial sequence window is scanned once to create the recent bit index table. The 1-*RRP*s are those data items with recent frequencies being no less than F_{min}. A pair of (k-1)-*RRP*s are merged to produce a candidate k-pattern. To avoid generating duplicate candidates, a pattern P_1 is merged with another pattern P_2 only if the (k-2)-suffix-subpattern of P_1 is identical to the (k-2)-prefix-subpattern of P_2. The resulting candidate pattern P_3 is a pattern with length (k+1), which is obtained by concatenating pattern P_1 with the last data item of P_2. The appearing bit sequence of P_3 is derived by performing a right shift operation on $\boldsymbol{Appear}_{P_1}$ followed by an **and** operation with $\boldsymbol{Appear}_{P_2}$. Then the recent frequencies of the candidate (k+1)-patterns are counted from their appearing bit sequences efficiently.

Whenever a (k+1)-*RRP* P is discovered, the pattern P and its appearing bit sequence $\boldsymbol{Appear}_P$ is inserted into a table for maintaining the recent maximum repeating patterns. Moreover, all the sub-patterns of P are removed from the table. The above process repeats until no more (k+1)-*RRP* is discovered.

3.2 Window Sliding Phase

In the window sliding phase, a new data item is appended into and the first data item is removed from the sequence window. However, the middle part of SW_t and SW_t is unchanged. Accordingly, the recently repeating patterns in the new current sequence window are discovered incrementally from the maintained information to prevent from generating candidate patterns iteratively.

First, the recent bit index table is updated. Each appearing bit sequence in the table is performed by a left shift operation to remove the first bit which corresponds to the out-of-date time point. Besides, for the newly coming data item x, the last bit of $Appear_x$ is set to be 1.

Let y denote the removed data item. There are two cases that the recent frequencies of patterns will be changed. The first case is for those patterns that are beginning with y and have an occurrence beginning from the removed data item. Then the recent frequencies of these repeating patterns are reduced by one. Similarly, it is possible that the recent frequency of a pattern ending with x is increased such that the pattern becomes into a recent repeating pattern. In the following, the process of incremental mining recently repeating patterns will be introduced according to these two cases.

Step 1: remove the out-of-date data item y

For each pattern maintained in the maximum repeating pattern table, its appearing bit index sequence is updated by performing a left shift operation. According to the new appearing bit index sequence of a pattern P, let bit i denote the first non-zero bit in $Appear_P$. It is implied that P has an occurrence located out of the new sequence

window if $i < |P|$. Therefore, bit i of $Appear_P$ is set to be 0 and the recent frequency of P is reduced by one.

If the recent frequency of P after updating is less than F_{min}, P is not a recent repeating pattern any more. However, all the suffix-subpatterns of P remain to be recent repeating patterns. Let P be denoted by $yp_1p_2...p_k$, where each p_i (i=1,..k) denotes a data item, individually. For keeping the information of maximum repeating patterns, the maximum suffix-subpattern $P'=p_1p_2...p_k$, which is composed of the pattern beginning from the second data item to the last one in P, is maintained. $Appear_{P'}$ is computed according to the information in the recent bit index table. Then pattern P' is inserted into the maximum repeating pattern table to replace pattern P.

Moreover, it is possible that part of the prefix-subpatterns of P remain to be recently repeating patterns. Initially, P'' is set to be y and $RF_t^{DS}(P'')$ is obtained according to $Appear_y$. If P'' is a recently repeating pattern, it is appended with p_i (i=1,..k) one by one until a maximum repeating pattern is discovered.

Step 2: append the newly coming data item x

Let z denote the last data item in the previous current sequence window. If $RF_t^{DS}(x)$ is less than F_{min}, there is not any recently repeating pattern newly generated. Otherwise, it is possible to form longer repeating patterns for the repeating patterns ending with z.

According to the maintained maximum repeating patterns, there are two situations to get the repeating patterns ending with z. The first one is to retrieve the suffix-subpatterns of a maximum pattern ending with z. Let P denote a maximum repeating pattern ending with z. The pattern P is appended with the newly coming data item x to generate a longer pattern Px, where $Appear_{Px}$ is computed according to the maintained information of $Appear_P$ and $Appear_x$. If Px is certified to be a recently repeating pattern, it is a new maximum repeating pattern. Accordingly, Px is inserted into the maximum repeating pattern table to replace pattern P. In the case that Px is not a recently repeating pattern, it is still possible to find new recently repeating patterns by appending x to the suffix-subpatterns of P. Let P be denoted by $p_1p_2...p_mz$, where each individual p_i (i=1,..m) is a data item. Each suffix-subpattern of P, denoted by P', is composed of $p_kp_{k+1}...p_mz$ for $k \geq 2$. Initially, P' is set to be the 1-suffix-subpattern of P, that is z in this case. Then, $Appear_{P'x}$ is computed according to $Appear_z$ and $Appear_x$. If $P'x$ is a recently repeating pattern, P' is extended to be p_mz, which is the 2-suffix-subpattern of P. The appearing bit sequence of p_mzx is obtained by performed two left shift operations on $Appear_{pm}$ followed by an AND operation with $Appear_{zx}$. The above process repeats until finding a k such that $P'=p_{k-1}p_k...p_mz$ and $RF_t^{DS}(P'x)$ is less than F_{min}. Finally, pattern $p_k...p_mzx$ is a maximum repeating pattern, which is inserted into the maximum repeating pattern table.

In addition to the suffix-subpatterns of a maximum repeating pattern ending with z, it is possible that a repeating pattern ending with z is contained in the middle of a maximum repeating pattern. Let Q denote a maximum repeating pattern consisting of $q_1q_2...q_mzq_{m+1...}q_n$, where each individual q_i (i=1,..n) is a data item. Let Q' denote the maximum prefix-subpattern of Q ending with z, which is composed of $q_1q_2...q_mz$. Then all the suffix-subpatterns of Q' are repeating patterns ending with z. Accordingly, Q'' is set to be the 1-suffix-subpattern of Q', that is z in this case. If $Q''x$ is a recently repeating pattern, Q'' is extended to be the 2-suffix of Q'. The above

SW_{20} A B C B C A B B C A B C B A B C B A B C

Data Item	Appearing Bit Sequence	Frequency
A	1 0 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0	5
B	0 1 0 1 0 0 1 1 0 0 1 0 1 0 1 0 1 0 1 0	9
C	0 0 1 0 1 0 0 0 1 0 0 1 0 0 0 1 0 0 0 1	6

(a)

Pattern	Appearing Bit Sequence	Frequency
ABCB	0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0	3

(b)

SW_{21} B C B C A B B C A B C B A B C B A B C A

Data Item	Appearing Bit Sequence	Frequency
A	0 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 1 0 0 1	5
B	1 0 1 0 0 1 1 0 0 1 0 1 0 1 0 1 0 1 0 0	9
C	0 1 0 1 0 0 0 1 0 0 1 0 0 0 1 0 0 0 1 0	6

(c)

Pattern	Appearing Bit Sequence	Frequency
ABCB	0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0	3

(d)

Pattern	Appearing Bit Sequence	Frequency
ABC	0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 0	3
BCB	0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0	3

(e)

Pattern	Appearing Bit Sequence	Frequency
ABC	0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 0	3
BCB	0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0	3
BCA	0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1	3

(f)

Figure 2: The recent bit index table and maintained patterns of the example

process repeats until finding a k such that $Q''x=q_kq_{k+1}...q_mzx$ is a recently repeating pattern but $q_{k-1}q_k...q_mzx$ is not or k=1. Finally, pattern $Q''x$ is a maximum repeating pattern, which is inserted into the maximum repeating pattern table.

According to the updated information in the maximum repeating pattern table, all the recently repeating patterns could be enumerated. Moreover, it provides the hints for mining recently repeating patterns incrementally in the next sequence window.

[Example 2]

Suppose a data stream DS=[ABCBCABBCABCBABCBABCA…) is given, the size of the sliding window is 20, and F_{min} is set to be 3. Accordingly, the initial sequence window SW_{20}=[ABCBCABBCABCBABCBABC]. The corresponding recent bit index table is constructed as shown in Figure 2(a). After window initialization phase is performed, the maximum repeating pattern ABCB is discovered and maintained in the maximum repeating pattern table as shown in Figure 2(b).

When the sequence window slides to be SW_{21}=[BCBCABBCABCBABC BABCA], a data item "A" is inputted and the first "A" is removed. All the appearing bit sequences are performed by a left shift operation, individually. Besides, the last bit of $Appear_A$ is set to be 1. The recent bit index table is updated to be the one shown in Figure 2(c). To eliminate the occurrence of the removed item "A", the appearing bit

sequence of the maximum repeating pattern "ABCB" is performed by a left shift operation to get the one shown in Figure 2(d). It is implied that the first occurrence of "ABCB" is out-of-date because the first non-zero bit in $Appear_{ABCB}$, located on bit 3, is less than the length of pattern "ABCB". Accordingly, the bit is reset to be 0 and the recent frequency of "ABCB" becomes to be 2. Although pattern "ABCB" is not a recently repeating pattern in SW_{21}, the maximum suffix-subpattern of "ABCB", "BCB" in this case, remains. The appearing bit sequence of "BCB" is obtained from a series of computing on $Appear_B$ and $Appear_C$. Furthermore, the prefix-subpatterns of "ABCB" are enumerated to find a longest prefix-subpattern of "ABCB" which is a recently repeating pattern. Consequently, "ABC" is discovered to be a maximum repeating pattern in SW_{21}. The maintained maximum repeating table is updated to be the one shown in Figure 2(e).

The last data item in the previous sequence window, SW_{20}, is "C". It is possible to generate a longer repeating pattern by appending the inputted item "A" to a repeating pattern ending with "C". First, the new data item "A" is appended to the maximum repeating pattern ending with "C", "ABC" in this case, to generate a longer pattern "ABCA". Since "ABCA" is not a repeating pattern, the other repeating patterns ending with "C" are generated by enumerating the suffix-subpatterns of "ABC": "C" and "BC" in sequence. Finally, "BCA" is discovered to be a new recently repeating pattern. After the process of window sliding phase is performed, the maintained patterns in the maximum repeating table are shown as Figure 2(f).

3.3 Maintaining Closed Repeating Patterns

[Def. 3] closed repeating pattern
Given patterns P_1 and P_2, it is named that P_1 is *close contained* in P_2 if P_1 is a sub-pattern of P_2 and $RF_t^{DS}(P_1) = RF_t^{DS}(P_2)$. A *closed repeating pattern* is defined as a recent repeating pattern which is not close contained in any of its super-patterns.

[Theorem1] If pattern P is close contained in pattern Px, the sub-patterns of P, which are close contained in P, are also close contained in Px.

For providing a lossless compression of the whole collection of repeating patterns, the closed repeating patterns in a sequence window are maintained instead of keeping the maximum repeating patterns. The processing steps described in the previous section are performed similarly on the maintained closed repeating patterns. Besides, the following properties are used to improve the processing efficiency.

[Property 1] Let $P=yp_1p_2...p_k$ denote a closed repeating pattern in SW_t. Besides, the maximum suffix-subpattern of P, $P'=p_1p_2...p_k$, is close contained in P. In the case that P becomes a non-repeating pattern in SW_{t+1} due to the removing of the out-of-date data item y, P' becomes a new closed repeating pattern in SW_{t+1} to replace P and $Appear_P=Appear_P$.

[Property 2] Let $P=yp_1p_2...p_k$ denote a closed repeating pattern in SW_t. Besides, a prefix-subpattern of P, $P''=yp_1p_2...p_j$ ($j<k$), is close contained in P. In the case that P becomes a non-repeating pattern in WS_{t+1} due to the removing of the out-of-date data item y, P'' is also a non-repeating pattern in SW_{t+1}. Accordingly, P is removed from the list of maintained patterns without needing to check the prefix-subpatterns of P.

[Property 3] Let z denote the last data item in SW_t and P denote a closed repeating pattern ending with z. In the case that Px becomes a repeating pattern in RS_{t+1} due to the inserting of the new data item x, according to [Theorem 1], all the sub-patterns of P which are close contained in P are close contained in Px if P is close contained in Px. Therefore, Px becomes a new closed repeating pattern in SW_{t+1} to replace P.

[Property 4] Let z denote the last data item in SW_t and P denote a closed repeating pattern ending with z. In the case that Px is not a repeating pattern in SW_{t+1} due to the inserting of the new data item x, for any suffix-subpattern of P, P', being close contained in P, $P'x$ is a non-repeating pattern. Therefore, it is not necessary to enumerate the suffix-subpatterns of P for generating longer patterns.

[Property 5] Let z denote the last data item in SW_t and Q denote a closed repeating pattern consisting of $q_1q_2...q_mzq_{m+1...}q_n$. It is not possible for a repeating pattern ending with z, which is close contained in the middle of Q, to become a repeating pattern in SW_{t+1} by inserting the new data item x. Therefore, it is not necessary to enumerate the sub-patterns of P ending with z for generating longer patterns.

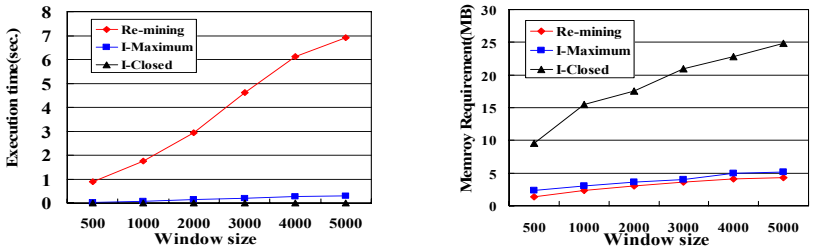
4. Performance Study

The proposed appearing-bit-sequence-based incremental mining algorithms were implemented using Visual C++ 6.0. In the following experiments, according to the properties of the maintained patterns, the two versions of algorithms are notated as ***I-Maximum*** and ***I-Closed***, respectively. Moreover, the appearing bit sequence approach [6] is applied to discover recently repeating patterns within each sequence window for comparison, which is notated as ***Re-mining*** algorithm. The experiments have been performed on a 3.4GHz Intel Pentium IV machine with 1 GB main memory.

The data sets are generated by the IBM data generator [1], where the generated sequential transaction data are concatenated together to simulate a sequence data stream with hiding patterns. Two input parameters are varied when running the data generator, where I is used to specify the number of various data items and P denotes the number of potential patterns in the generated data sequences.

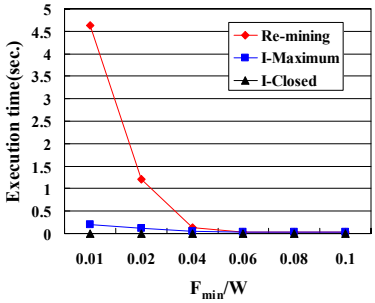
In the following four experiments, the data sets are generated by concatenating 50,000 sequential data with I100P100 setting. The scalabilities of I-Maximum and I-Closed algorithms on execution time and memory usage are compared with the ones of Re-mining algorithm under various parameters setting. According to theses experimental results, the effectiveness of incremental mining of recent repeating patterns is observed.

[Experiment 1] Changing the sliding window size **(W)**
In this experiment, F_{min} is set to be W *0.01. Figure 3(a) shows the average running time of I-Maximum, I-Closed, and Re-mining over a sliding window under different W setting. It is reported that the execution efficiency of I-Maximum and I-Closed outperforms the one of Re-Mining algorithm significantly. When W increases, the execution efficiency of Re-mining is much slower than the other two algorithms. Both I-Maximum and I-Closed are linear scalable, but I-Closed has better scalability. The total space

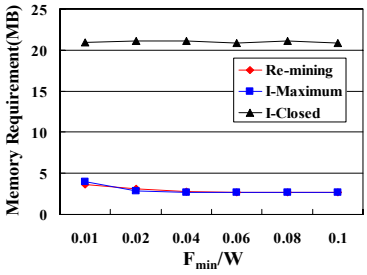


(a)

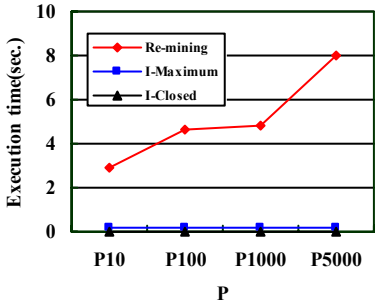
(b)



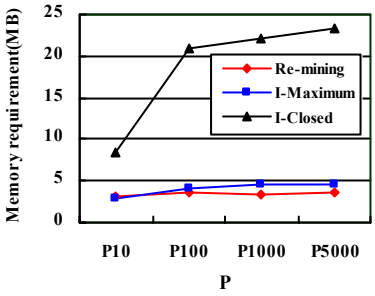
(c)



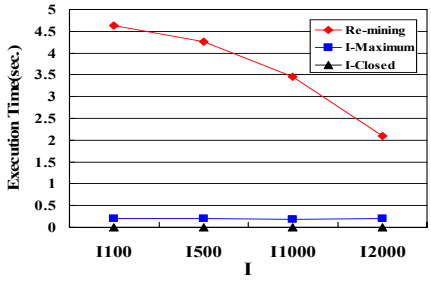
(d)



(e)



(f)



(g)

Figure 3: Results of Experiments

needed for three algorithms are analyzed as shown in Figure 3(b). As W increases, due to the length of an appearing bit sequence increases, the memory requirement of all the three algorithms grows linearly with the window size W. For I-Closed and I-Maximum, the major memory space needed is to hold the maintained patterns for incremental mining. Since the number of closed repeating patterns is much larger than the one of maximum repeating patterns, the memory requirement of I-Maximum is larger than the other two. Although the memory requirement of I-Closed is 5 times of the one of Re-mining, the execution efficiency of I-Closed is near 1000 times faster than Re-mining. Moreover, with slightly larger memory requirement than Re-mining, 20 times of execution efficiency is achieved by I-Maximum.

[Experiment 2] Changing the minimum frequency threshold value **(F_{min})**
In this experiment, W is set to be 3000. Figure 3(c) shows the execution time of I-Maximum, I-Closed, and Re-mining under different F_{min} setting. As seen in the figure, the execution time of Re-mining grows exponentially as F_{min} decreases because the number of satisfying patterns increases dramatically. However, the execution of I-Maximum is faster than that of Re-mining nearly an order of magnitude. The execution of I-closed is even faster than that of I-Maximum. In the situations when F_{min} is larger than W*0.04, much fewer repeating patterns are discovered. Therefore, the execution time of all the three algorithms is approximately the same. Because of the reason described above, it is shown in Figure 3(d) that the memory requirements of both I-Maximum and Re-mining increase as F_{min} decreases. Although I-Closed has the largest memory requirement among the three algorithms, the amount of its required memory keeps nearly stable.

[Experiment 3] Changing the setting of P for generating data sets
In this experiment, parameter I is set to be 100. Besides, W and F_{min} are fixed to be 3000 and W *0.01(=30), respectively. The execution times and memory requirements of I-Maximum, I-Closed, and Re-mining under different P setting are shown in Figure 3(e) and 3(f), respectively. As the number of potential patterns increases, more processing cost is required by Re-mining for generating candidate patterns. Since the main processing costs of I-Maximum, I-Closed are spent for mining frequency changes caused by window sliding, the execution time of these two algorithms keeps more stable than the one of Re-mining. The memory requirement of I-Closed is proportional to the number of repeating patterns. Therefore, as shown in Figure 3(f), the memory requirement of I-Closed grows as the number of various data items increases. However, the observed space-time trade-off between I-Closed and Re-mining is coincident with the one shown in [Experiment 1].

[Experiment 4] Changing the setting of parameter I for generating data sets
In this experiment, parameter P is set to be 100. Besides, W and F_{min} are set to be 3000 and W *0.01(=30), respectively. The execution time of I-Maximum, I-Closed, and Re-mining under different I setting is shown in Figure 3(g). As the number of various data items increases, the distribution of patterns in a sliding window becomes sparser such that the number of satisfying patterns decreases. Accordingly, the execution time of Re-mining goes down as the number of various data items increases. However, for the same reason stated in [Experiment 3], the running time of both I-Maximum and I-Closed remains stable.

5. Conclusion and Future Works

In this paper, the appearing bit sequences are used to monitor the occurrences of repeating patterns within a sliding window. Two versions of incremental algorithms are proposed to maintain the maximum and closed repeating patterns in a sliding window, respectively. Accordingly, the cost of re-mining repeating patterns over a sliding window is reduced to that of mining frequency changes of the maintained patterns. It is reported from the experimental results that the incremental mining methods perform much better than the re-miming approach.

References

[1] R. Agrawal and R. Srikant, "Fast Algorithms for Mining Association Rules," in Proc. of Int. Conf. on Very Large Data Bases, 1994.

[2] J.H. Chang and W.S. Lee, "Finding Recent Frequent Itemsets Adaptively over Online Data Streams," in Proc. the 9th ACM International Conference on Knowledge Discovery and Data Ming, 2003.

[3] Y. Chi, H. Wang, P.S. and Yu, R.R. Muntz, "Moment: Maintaining Closed Frequent Itemsets over a Stream Sliding Window," in Proc. Int. Conf. on Data Mining (ICDM'04).

[4] P. Domingos and G. Hulten, "Mining high-speed data streams," in Proc. The6th ACM International Conference on Knowledge Discovery and Data Ming, 2000.

[5] J.L. Hsu, C.C. Liu, and Arbee L.P. Chen, "Discovering Nontrivial Repeating Patterns in Music Data," in IEEE Transactions on Multimedia , 2001.

[6] J.L. Koh and W.D.C.Yu, "Efficient Feature Mining in Music Objects," Lecture Notes in Computer Science: DEXA'01: Database and Expert Systems Applications, Springer-Verlag, 2001.

[7] J.L. Koh and Y.T. Kung, "An Efficient Approach for Mining Top-K Fault-Tolerant Repeating Patterns," Lecture Notes in Computer Science: DASFAA'06: Database Systems for Advanced Applications, Springer-Verlag, 2006.

[8] H. Li, S. Lee, and M.K. Shan, "Online Mining (Recently) Maximal Frequent Itemsets over Data Streams," in Proc. of RIDE-SDMA'05.

[9] C.H. Lin, D.Y. Chiu, Y.H. Wu, and Arbee L.P. Chen, "Mining Frequent Itemsets from Data Streams with a Time-Sensitive Sliding Window" in Proc. SIAM International Conference on Data Mining, 2005.

[10] N.H. Liu, Y.H. Wu, and Arbee L.P. Chen, "An Efficient Approach to Extracting Approximate Repeating Patterns in Music Databases," Lecture Notes in Computer Science: DASFAA'05: Database Systems for Advanced Applications , Springer-Verlag , 2005.

[11] G. S. Manku and R. Chen Motwani, "Approximate Frequent Counts over Data Streams," in Proc. of the 28th International Conference on Very Large Database, 2002.

[12] H. Wand, W. Fan, P. S. Yu, and J. Han, "Mining Concept Drifting Data Streams using Ensemble Classifiers," in Proc. the 9th ACM International Conference on Knowledge Discovery and Data Ming, 2003.