Finding Recent Frequent Itemsets Adaptively over Online Data Streams

Joong Hyuk Chang Won Suk Lee
Department of Computer Science, Yonsei University
134 Shinchon-dong Seodaemun-gu Seoul, 120-749, Korea
+82-2-2123-2716

{ jhchang, leewo }@amadeus.yonsei.ac.kr

ABSTRACT

A data stream is a massive unbounded sequence of data elements continuously generated at a rapid rate. Consequently, the knowledge embedded in a data stream is more likely to be changed as time goes by. Identifying the recent change of a data stream, specially for an online data stream, can provide valuable information for the analysis of the data stream. In addition, monitoring the continuous variation of a data stream enables to find the gradual change of embedded knowledge. However, most of mining algorithms over a data stream do not differentiate the information of recently generated transactions from the obsolete information of old transactions which may be no longer useful or possibly invalid at present. This paper proposes a data mining method for finding recent frequent itemsets adaptively over an online data stream. The effect of old transactions on the mining result of the data steam is diminished by decaying the old occurrences of each itemset as time goes by. Furthermore, several optimization techniques are devised to minimize processing time as well as main memory usage. Finally, the proposed method is analyzed by a series of experiments.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Application - Data mining

General Terms

Algorithm

Keywords

Recent frequent itemsets, Data stream, Decay mechanism, Delayed-insertion, Pruning of itemsets

1. INTRODUCTION

A data stream is a massive unbounded sequence of data elements continuously generated at a rapid rate. Due to this reason, it is impossible to maintain all elements of a data stream. As a result, data stream processing should satisfy the following requirements [1]. First, each data element should be examined at most once to analyze a data stream. Second, memory usage for data stream analysis should be restricted finitely although new data elements are continuously generated in a data stream. Third, newly generated data elements should be processed as fast as possible. Finally, the up-to-date analysis result of a data stream should be instantly available when requested. In order to satisfy these requirements, data stream processing sacrifices the correctness of its analysis result by allowing some error.

The target application domains of a data stream are either a bulk addition of new transactions as in a data warehouse system or an individual addition of a continuously generated transaction as in a network monitoring system. The former is called as an offline data stream while the latter is called as an online data stream [2]. For an offline data stream, it is possible to enhance the performance of data mining through a batch operation by processing a considerable number of newly generated transactions together [2]. Due to this reason, the up-to-date mining result of an offline data stream is available only after a batch operation is finished. Therefore, the granularity of generating the most up-to-date result depends on the number of new transactions batch-processed together. However, data mining over an online data stream should support flexible trade-off between processing time and mining accuracy without any fixed granule of data mining in order to catch the sensitive change of its mining result as quickly as possible.

Among the frequency counting algorithms [2,3] of data elements over a data stream, the *Lossy Counting* algorithm [2] is the most representative method. In the Lossy Counting algorithm, the set of frequent itemsets in a data stream is found when a maximum allowable error ε as well as a minimum support is given. A set of newly generated transactions in a data stream is loaded together into a fixed-sized buffer in main memory and they are batch-processed. The information about the previous mining result up to the latest batch operation is maintained in a data structure called D containing a set of entries of a form (e, f, Δ) where e is an itemset, f is the count of the itemset e, and Δ is the maximum possible error count of the itemset e. In order to update the information of the data structure D, all of its entries are looked up in sequence. For the entry (e, f, Δ) of an itemset e in D, if the itemset e is one of the itemsets identified by the new transactions in the buffer, its previous count f is incremented by its count in the new transactions. Subsequently, if its estimated count i.e., $f+\Delta$ is less than $\varepsilon \times N$, it is pruned from D. On the other hand, when there is no entry in D for a new itemset e identified by the new transactions in the buffer, a new entry (e, f, Δ) is inserted to D. Its maximum possible error Δ is set to $\lfloor \varepsilon \times N' \rfloor$ where N' denotes the number

SIGKDD '03, August 24-27, 2003, Washington, DC, USA.

of transactions that were processed up to the latest batch operation.

Generally, knowledge embedded in a data stream is more likely to be changed as time goes by. Identifying the recent change of a data stream quickly, specially for an online data stream, can provide valuable information for the analysis of the data stream. In addition, monitoring the continuous variation of a data stream enables to find the gradual change of embedded knowledge, so that it can be timely utilized. In order to achieve this, the effect of obsolete information in old transactions on the current mining result of a data stream should be eliminated effectively. As a simple solution, it is possible to consider a sliding window approach. It restricts the target transactions of data mining to those transactions that are generated within the most recent period of a fixed-sized window. However, its current mining result totally depends on recently generated transactions in the range of the window. Due to this reason, this approach is a primitive way of disregarding obsolete information. In addition, all the transactions in the window need to be maintained in order to remove their effects on the current mining result when they are out of the range of a sliding window.

In terms of information differentiation, the *SWF* algorithm [4] uses a sliding window to find frequent itemsets in the fixed number of recent transactions. The sliding window is composed of a sequence of partitions. Each partition maintains a number of transactions. The candidate 2-itemsets of all transactions in the window are maintained separately. When the window is advanced, the oldest partition is disregarded and a new partition containing newly generated transactions is appended to the window. At the same time, the candidate *2*-itemsets of the advanced window are adjusted. Subsequently, all possible candidate itemsets are generated by these candidate *2*-itemsets. The new set of frequent itemsets is identified by scanning the transactions of the slid window. A more flexible way of information differentiation is presented in [5] where correlations among co-evolving time sequences are analyzed. The missing values of the sequences are estimated and their future values are predicted. In order to identify the recent change of correlations adaptively, a forgetting factor is used to diminish the effect of old correlations among sequences. A forgetting factor determines how fast the effect of old information is faded away. This type of an information decay model is also introduced in NIDES [6] for anomaly intrusion detection. NIDES models the historical behavior of a user's activities in terms of various measures and generates a long-term profile containing a statistical summary for each measure. In order to concentrate on the recent behavior of the user, the statistics of old activities in the long-term profile are decayed as new activities are performed by the user.

This paper proposes a method of finding recent frequent itemsets adaptively over an online data stream. It examines each transaction in a data stream one-by-one without any candidate generation. The occurrence count of a significant itemset that appears in each transaction is maintained by a prefix-tree lattice structure in main memory. The effect of old transactions on the current mining result is diminished by decaying the old occurrence count of each itemset as time goes by. In addition, the rate of decay old information is flexibly defined as needed. The total number of significant itemsets in main memory is minimized by delayed-insertion and pruning operations of an itemset. As a result, its processing time is flexibly controlled while sacrificing its accuracy.

2. PRELIMINARIES

For finding frequent itemsets, a data stream can be defined as follows:

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

ii) An itemset e is a set of items such that $e \in (2^I - \{\emptyset\})$ where 2^I is the power set of I. The length $|e|$ of an itemset e is the number of items that form the itemset and it is denoted by an $|e|$-itemset. An itemset $\{a,b,c\}$ is denoted by *abc*.

iii) A transaction is a subset of I and each transaction has a unique transaction identifier *TID*. A transaction generated at the k^{th} turn is denoted by T_k.

iv) When a new transaction T_k is generated, the current data stream D_k is composed of all transactions that have ever been generated so far i.e., $D_k = <T_1, T_2, \dots, T_k>$ and the total number of transactions in D_k is denoted by $|D|_k$.

When a transactions T_k is generated currently, the current count $C_k(e)$ of an itemset e is the number of transactions that contain the itemset among the k transactions. Likewise, the current support $S_k(e)$ of an itemset e is the ratio of its current count $C_k(e)$ over $|D|_k$.

A decay rate means the reducing rate of a weight for a fixed decay-unit. A decay-unit determines the chunk of information to be decayed together. A decay rate is defined by two parameters: a *decay-base b* and a *decay-base-life h*. A ***decay-base b*** determines the amount of weight reduction per a decay-unit and it is greater than 1. When the weight of the current information is set to 1, a ***decay-base-life h*** is defined by the number of decay-units that makes the current weight be b^{-1}. Based on these two parameters, a ***decay rate d*** is defined as follows:

$$d = b^{-(1/h)} \qquad (b>1,\ h\geq 1,\ b^{-1} \leq d < 1) \qquad (1)$$

Theorem 1. Given a decay rate $d = b^{-(1/h)}$ $(b>1,\ h\geq 1,\ b^{-1} \leq d < 1)$, the total number of transactions $|D|_k$ in the current data stream D_k is found as follows:

$$|D|_k = \begin{cases} 1 & \text{if } k=1 \\ |D|_{k-1} \times d + 1 & \text{if } k \geq 2 \end{cases}$$

The value of $|D|_k$ converges to $1/(1-d)$ as the value k increases infinitely.

(Proof)

When the first transaction is looked up, the number of transactions $|D|_1$ is obviously 1 since there is no previous transaction whose weight should be decayed. When the second transaction is looked up, the total number of transactions $|D|_2$ is computed by $|D|_1 \times d+1$ since the weight of the first transaction is decayed. Subsequently, when a new transaction is generated at the k^{th} $(k\geq 2)$ turn, the total number of transactions $|D|_k = |D|_{k-1} \times d + 1$. Consequently, it can be expressed by $|D|_k = d^{k-1} + d^{k-2} + \dots + d + 1 = (1-d^k)/(1-d)$. Since $b^{-1} \leq d < 1$, $|D|_k$ converges to $1/(1-d)$ as k increases infinitely. □

Similarly, the count $C_k(e)$ of an itemset e in the current data stream D_k is obtained as follows:

$$C_k(e) = C_{k-1}(e) \times d + W_k(e), \quad W_k(e) = \begin{cases} 1 & \text{if } e \in T_k \\ 0 & \text{otherwise} \end{cases}$$

3. FINDING RECENT FREQUENT ITEMSETS

In this section, a method of finding recent frequent itemsets adaptively over an online data stream is proposed based on the decay mechanism described in Section 2. The different combinations of items that appear in each transaction are maintained in a prefix-tree lattice structure [7,8] called as a ***monitoring lattice***. A node in a monitoring lattice contains an item and it denotes an itemset composed of items that are in the nodes of its path from the root.

3.1 Count Estimation of an Itemset

In the *Carma* algorithm [9], the maximum possible count of an itemset is estimated by the minimum value among the maximum possible counts of all of its subsets. A new itemset is inserted to a lattice of itemsets if it is potentially frequent and all of its subsets are maintained in the lattice. Similarly, the count of an itemset that are not maintained can be estimated by its subsets that are maintained in a monitoring lattice. For this purpose, the terms defined in Definition 1 and Definition 2 are used.

Definition 1. For an n-itemset e ($n \geq 2$), a set of its subsets $P(e)$, a set of its m-subsets $P_m(e)$ and a set of counts for its m-subsets $P^C_m(e)$ are formally defined as follows.

i) A set of its subsets $P(e)$ is composed of all possible itemsets that can be generated by one or more items of the itemset e i.e., $P(e)=\{\alpha | \forall\alpha \;\; \text{s.t.} \; \alpha \in 2^e - \{e\} \text{ and } \alpha \neq \emptyset\}$.

ii) A set of its m-subsets $P_m(e)$ is composed of those itemsets in $P(e)$ that have m items ($m<n$) i.e., $P_m(e)=\{\alpha | \forall\alpha \;\; \text{s.t.} \; \alpha \in P(e) \text{ and } |\alpha|=m\}$.

iii) A set of counts for its *m-subsets* $P^C_m(e)$ is composed of the distinct counts of all itemsets in $P_m(e)$ i.e., $P^C_m(e)=\{C(\alpha) | \forall\alpha \;\; \text{s.t.} \; \alpha \in P_m(e)\}$, where $C(e)$ denotes the count of an itemset e over a data stream. □

Definition 2. For two itemsets e_1 and e_2, a ***union-itemset*** $e_1 \cup e_2$ and an ***intersection-itemset*** $e_1 \cap e_2$ are defined as follows.

i) A union-itemset $e_1 \cup e_2$ is composed of all items that are members of either e_1 or e_2.

ii) An intersection-itemset $e_1 \cap e_2$ is composed of all items that are members of both e_1 and e_2. □

For an itemset, each of its subsets appears in at least as many transactions as the itemset appears in. Furthermore, when all items of an itemset always appear together in each transaction, the count of the itemset should be identical to those of its subsets. Therefore, the count of an itemset depends on how often its items appear together in each transaction. Based on this observation, the possible range of the count of an itemset can be identified by two extreme distributions: ***least exclusively distributed*** (***LED***) and ***most exclusively distributed*** (***MED***). When the items of an itemset are LED, they appear together in as many transactions as possible. On the other hand, the items of an itemset appear exclusively as many transactions as possible when they are MED.

In a data set D, the count of an n-itemset e can be estimated by the individual counts of its subsets. Its maximum count $C^{max}(e)$ is found when all of its subsets are LED. It is the smallest value among the counts of all the subsets. However, since the set of its (n-1)-subsets can provide the most accurate information about the count of the n-itemset, $C^{max}(e)$ can be estimated by only its (n-1)-subsets. Therefore, when $min(V)$ denotes the smallest value in a set of values V, the maximum count $C^{max}(e)$ of an itemset e is found as follows:

$$C^{max}(e) = min(P^C_{n-1}(e)) \tag{2}$$

For two itemsets e_1 and e_2, the minimum count $C^{min}(e_1 \cup e_2)$ of their union-itemset $e_1 \cup e_2$ can be estimated as follows.

$$C^{min}(e_1 \cup e_2) = \begin{cases} max(0, C(e_1) + C(e_2) - C(e_1 \cap e_2)) & \text{if } e_1 \cap e_2 \neq \emptyset \\ max(0, C(e_1) + C(e_2) - |D|) & \text{if } e_1 \cap e_2 = \emptyset \end{cases} \tag{3}$$

where $|D|$ denotes the total number of transactions in D and $max(V)$ denotes the largest value in a set of values V. Based on Equation (3), the minimum count $C^{min}(e)$ of an itemset e can also be estimated by the counts of its (n-1)-subsets. In other words, for each distinct pair (α_i, α_j) of its (n-1)-subsets i.e., α_i and $\alpha_j \in P_{n-1}(e)$, the count of their union-itemset $\alpha_i \cup \alpha_j$ can be estimated. Among the estimated counts for the itemset e, the largest count is the guaranteed appearance count i.e., the minimum count $C^{min}(e)$ of the itemset e as follows:

$$C^{min}(e) = max(\{C^{min}(\alpha_i \cup \alpha_j) \mid \forall\alpha_i, \alpha_j \in P_{n-1}(e) \text{ and } i \neq j\})$$

The maximum count $C^{max}(e)$ of an itemset e is used as the estimated count of the itemset. Consequently, there may exist an estimation error count since $C^{max}(e)$ is the largest possible count that the itemset can appear in the transactions of a data set. Let the difference between $C^{max}(e)$ and $C^{min}(e)$ be the *estimation error* $E(e)$ of the itemset.

3.2 *estDec* Method

Not all of itemsets that appear in a data stream are significant for finding frequent itemsets. An itemset which has much less support than a predefined minimum support is not necessarily monitored since it cannot be a frequent itemset in the near future. Therefore, the insertion of a new itemset can be delayed until it can possibly be a frequent itemset in the near future. When the estimated support of a new itemset is large enough, it is regarded as a *significant itemset* and it is inserted to a monitoring lattice. On the other hand, an efficient pruning technique is obviously another way of reducing the usage of memory space. Although an itemset in a monitoring lattice was significant enough to be monitored in the past, if its current support becomes much less than a predefined minimum support, it can be eliminated from the monitoring lattice.

This section proposes an *estDec* method for finding recent frequent itemsets adaptively over an online data stream. Every node in a monitoring lattice maintains a triple (*cnt*, *err*, *MRtid*) for its corresponding itemset *e*. The count of the itemset *e* is denoted by *cnt*. The maximum error count of the itemset *e* is denoted by *err*. Finally, the transaction identifier of the most recent transaction that contains the itemset *e* is denoted by *MRtid*. The *estDec* method is composed of four phases: parameter updating phase (Phase Ⅰ), count updating phase (Phase Ⅱ), delayed-insertion phase (Phase Ⅲ) and frequent itemset selection phase (Phase Ⅳ). The detailed steps of these phases are illustrated in Figure 1.

When a new transaction T_k is generated in a data stream, the total number of transactions in the current data stream $|D|_k$ is updated in the parameter updating phase (line 4 in Figure 1) as follows:

$$|D|_k = |D|_{k-1} \times d + 1$$

In the count updating phase (line 5-9), the counts of those itemsets in a monitoring lattice that appear in the new transaction are updated. All the paths of a monitoring lattice that are induced by the items of the transaction are traversed and the previous triple (cnt_{pre}, err_{pre}, $MRtid_{pre}$) of each node in the paths is updated to the current triple (cnt_k, err_k, $MRtid_k$) as follows:

$$cnt_k = cnt_{pre} \times d^{(k - MRtid_{pre})} + 1,$$

$$err_k = err_{pre} \times d^{(k-MRtid_{pre})}, \quad MRtid_k = k$$

When the updated support i.e., $cnt_k / |D|_k$ of an itemset in a monitoring lattice becomes less than a predefined threshold, the itemset is regarded as an insignificant itemset, so that it is pruned from the monitoring lattice as in conventional lattice-based data mining methods [7,8]. However, if a *1*-itemset is pruned from a monitoring lattice, it is impossible to estimate its count later. Therefore, it should not be pruned. This mechanism is called as a ***pruning*** operation of an itemset. The threshold of this operation is defined as a ***threshold for pruning S_{prn}*** which should be less than a minimum support S_{min}.

After all of these itemsets are updated, the delayed-insertion phase (line 10-22) is started in order to find any new itemset that has a high possibility to become a frequent itemset in the near future. A new itemset is inserted to a monitoring lattice only in the following two cases. The first case is when a new *1*-itemset appears in a newly generated transaction. In this case, the itemset is instantly inserted to a monitoring lattice without any estimation process. Consequently, the count *cnt* of every *1*-itemset in a monitoring lattice is not an estimated value but an actual value. The second case is when the estimated support of an *n*-itemset ($n \geq 2$) that is not in the monitoring lattice is large enough to be monitored. In this phase, among the items of the new transaction, the items whose supports are less than S_{ins} are not considered. While navigating the lattice according to the remaining items of the new transaction, the count of an insignificant itemset that is composed of a significant itemset and one of the remaining items is estimated by its maximum count $C^{max}(e)$ as described in Section 3.1. Due to the characteristics of a prefix lattice structure, there is no candidate itemset generation process. This is because such an itemset is identified systematically while navigating the lattice according to the remaining items in the new transaction. If any of its ($|e|$-*1*)-subsets in $P_{n-1}(e)$ is not currently maintained in the monitoring lattice, the count of the itemset *e* is not estimated. This is because its $C^{max}(e)$ is always 0 in this case. Subsequently, the estimated support of the itemset can be found by the ratio of its count *cnt* over the current total number of transactions $|D|_k$. If it is greater than or equal to a predefined threshold, the itemset is inserted to the monitoring lattice. This mechanism is called as a ***delayed-insertion*** operation and the pre-defined threshold for this insertion is defined as a ***threshold for delayed-insertion S_{ins}*** which should be also less than a minimum support S_{min}.

When an itemset *e* is inserted, all of its ($|e|$-*1*)-subsets should be significant. Due to this reason, it is possible to find the upper bound $C^{upper}(e)$ of its actual count when it is inserted at the k^{th} transaction. In other words, among the *k* transactions generated so far, at least $|e|$-*1* transactions that contain the itemset *e* are required to insert all of its subsets to the monitoring lattice in advance. Therefore, its actual count is maximized when these $|e|$-*1* transactions are most recently generated. The similar approach is used in [9]. The decayed count of the itemset *e* for the insertion of its subsets by these recent $|e|$-*1* transactions is represented by a term *cnt_for_subsets* as follows:

$$cntt\_for\_subsets = \{1 - d^{(|e|-1)}\}/(1-d) \tag{4}$$

In addition, the maximum possible decayed count of the itemset *e* before the recent $|e|$-*1* transactions is denoted by *max_cnt_before_subsets* and it is represented as follows:

$$max\_cnt\_before\_subsets = S_{ins} \times \{|D|_{k-(|e|-1)}\} \times d^{(|e|-1)} \tag{5}$$

Consequently, $C^{upper}(e)$ can be found as follows:

$$C^{upper}(e) = max\_cnt\_before\_subsets + cnt\_for\_subsets \tag{6}$$

If $C^{max}(e)$ in Equation (2) is greater than the upper bound $C^{upper}(e)$, $C^{upper}(e)$ is used as its count *cnt*. Accordingly, the current triple (cnt_k, err_k, $MRtid_k$) of the itemset *e* in the corresponding node of the monitoring lattice is updated as follows:

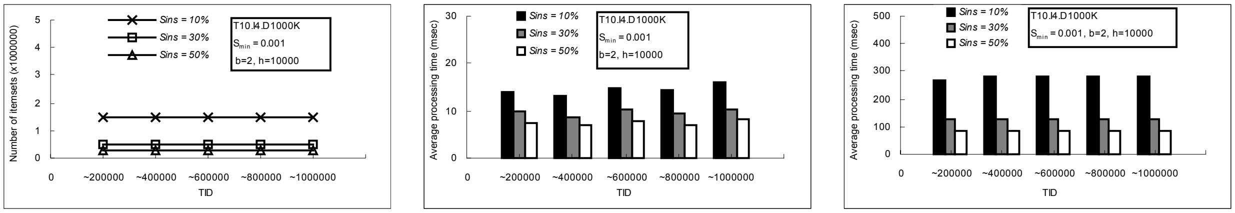
$$cnt_k = min\{C^{max}(e), C^{upper}(e)\}$$
$$err_k = E(e) = cnt_k - C^{min}(e), \quad MRtid_k = k$$

```
Input:   A data stream $D$
Output:  A complete set of recent frequent itemsets $L_k$

         $d$ : A given decay rate
         $ML$ : A monitoring lattice
1:       $ML = \varnothing$;
2:       for each new transaction in $D$ {
3:           read current transaction $T_k$;
         // Parameter updating phase
4:           $|D|_k = |D|_{k-1} \times d + 1$;

         // Count updating phase
5:           for all   itemset  $e$  s.t. $e \in (2^{T_k} - \{\varnothing\})$ and $e \in ML$  {
6:               $cnt = cnt \times d^{(k-MRtid)} + 1$; $err = err \times d^{(k-MRtid)}$; $MRtid = k$;
7:               if  $(cnt/|D|_k) < S_{prn}$  and  $|e| > 1$      // Pruning
8:                   Eliminate $e$ and it's child node from $ML$;
9:           } // of for in line 5

         // Delayed-insertion phase
10:          $T_k^{/}$ = ItemFiltering($T_k$);

11:          for all   itemset  $e$  s.t. $e \in (2^{T_k^{/}} - \{\varnothing\})$ and $e \notin ML$  {
12:              if  $|e| = 1$ {
13:                  Insert $e$ into $ML$; $cnt$=1; $err$ = 0; $MRtid = k$;
14:              } else {
15:                  Estimate  $C^{max}(e)$ and $C^{min}(e)$;
16:                  if  $C^{max}(e) > C^{upper}(e)$
17:                      $C^{max}(e) = C^{upper}(e)$;

18:                  if  $(C^{max}(e) / |D|_k) \geq S_{ins}$  {
19:                      Insert $e$ into $ML$; $cnt = C^{max}(e)$; $err = C^{max}(e) - C^{min}(e)$; $MRtid = k$;
20:                  } // of if n line 18
21:              } // of else in line 14
22:          } // of for in line 11

         // Frequent itemset selection phase
23:          $L_k = \varnothing$;
24:          for all   itemset  $e \in ML$ {
25:              $cnt = cnt \times d^{(k-MRtid)}$; $err = err \times d^{(k-MRtid)}$; $MRtid = k$;
26:              if  $(cnt/|D|_k) \geq S_{min}$
27:                  $L_k = L_k \cup \{e\}$;
28:      } // of for in line 23
29:  } // of for in line 2
```

Figure 1. The *estDec* method

An itemset pruned at present can be inserted into the monitoring lattice in the future by the delayed-insertion operation if it appears frequently in new transactions. Consequently, S_{prn} should be less than S_{ins}. As the gap between the two thresholds S_{prn} and S_{ins} is enlarged, the possibility of repeating the insertion and pruning of the same itemset frequently is reduced. Furthermore, as the gap between these two thresholds is enlarged, the accuracy of frequent itemsets is improved while the size of a monitoring lattice is increased.

The frequent itemset selection phase (line 23-28) is performed only when the mining result of the current data set is required. It produces all current frequent itemsets in a monitoring lattice by the same way as in conventional mining methods [7,8] based on a prefix-tree lattice structure. When this phase is performed in the current data stream D_k, an itemset *e* is frequent if its current support $\{cnt \times d^{(k-MRtid)}\} / |D|_k$ is greater than a predefined minimum support S_{min}. Furthermore, its current support error $\{err \times d^{(k-MRtid)}\} / |D|_k$ can be found as well.



(a) **(b)** **(c)**

Figure 2. Performance of the *estDec* method for the data set *T10.I4.D1000K*

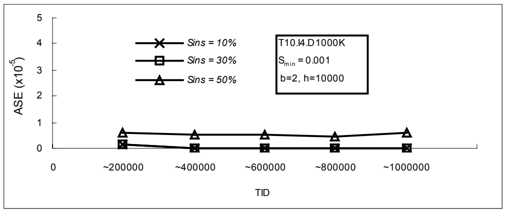


Figure 3. Accuracy of mining results

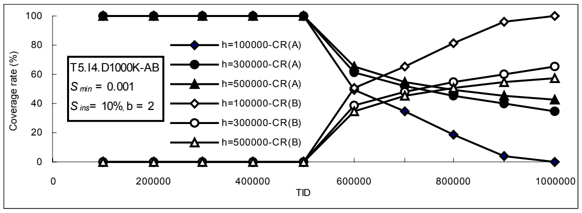


Figure 4. Coverage rate for the data set *T5.I4.D1000K-AB*

All insignificant itemsets in a monitoring lattice can be pruned together by examining the current support of every itemset in the monitoring lattice. This mechanism is called as a ***force-pruning*** operation and can be performed periodically or when the current size of a monitoring lattice reaches a pre-defined threshold value.

4. STABILIZED ANALYSIS

When the information of a transaction is rapidly decayed, the resulting set of recently frequent itemsets may be too sensitively varied since it is dominated by the itemsets of newly generated transactions. This type of fluctuation in the set of frequent itemsets is meaningless. In order to avoid this, a ***safety factor*** γ is introduced. It is defined by the maximum number of most recently generated consecutive transactions that contains a new itemset while the itemset remains as an infrequent itemset. The resulting set of recently frequent itemsets in a data stream D_k becomes stable when the decay-base-life h of a decay rate $d = b^{-(1/h)}$ is set to be greater than or equal to its lower bound h^{LB} defined in Theorem 2. If a decay-base-life h is less than its lower bound h^{LB}, an itemset whose current count is less than the safety factor γ may become a frequent itemset.

Theorem 2. Given a minimum support S_{min}, a safety factor γ and a decay-base b, the decay-base-life h of a decay rate $d = b^{-(1/h)}$ should be greater than or equal to its lower bound h^{LB} defined as follows:

$$h^{LB} = \lceil -\{\gamma / (\log_b (1 - S_{\min})\} \rceil$$

(Proof)

When a new itemset e appears in most recently generated γ consecutive transactions in a data stream D_k, its $C_k(e)$= $(1-d^{\gamma})/(1-d)$. Consequently, its support $S_k(e)$ is always greater than $(\frac{1-d^{\gamma}}{1-d} / \frac{1}{1-d})$ = $1-d^{\gamma}$ since the maximum value of $|D|_k$ is $1/(1-d)$. The itemset should not be a frequent itemset for the given safety factor γ, so that its support should be always less than the minimum support S_{min}. Therefore, $1-d^{\gamma}$ <S_{min} should be satisfied. Since $d = b^{-(1/h)}$, the lower bound of a decay-base-life h^{LB} is found as follows:

$$h^{LB} = \lceil -\{\gamma / (\log_b (1 - S_{\min})\} \rceil . \qquad \square$$

5. EXPERIMENTAL RESULTS

In this section, two data sets *T10.I4.D1000K* and *T5.I4.D1000K-AB* are used to evaluate the performance of the *estDec* method. Each data set is generated by the same method as described in [10] and the total number of items is 1,000. In all experiments, the transactions of each data set are looked up one by one in sequence to simulate the environment of an online data stream. In addition, every value of a decay-base-life h used in these experiments is above its lower bound. Each of two thresholds S_{ins} and S_{prn} is assigned relatively to the value of a predefined minimum support S_{min}. These two thresholds are not necessarily to be the same in practice. However, in all experiments, they are set to be the same value. When the value of the threshold S_{ins} is denoted by p%, the actual value of the threshold S_{ins} is equal to S_{min}×*(p/100)*. Likewise, S_{prn} is also equal to S_{min}×*(p/100)*. All experiments are performed on a 1.8GHz Pentium PC machine with 512MB main memory running on Linux 7.3 and all programs are implemented in C.

Figure 2 shows the performance of the *estDec* method for the data set *T10.I4.D1000K*. Figure 2-(a) shows the memory usage and it is represented by the maximum number of itemsets in a monitoring lattice. A minimum support S_{min}, a decay-base b and a decay-base-life h are set to 0.001, 2 and 10,000 respectively. The sequence of generated transactions is divided into 5 intervals each of which consists of 200,000 transactions and a force-pruning operation is performed in every 1,000 transactions. Since only significant itemsets are maintained in a monitoring lattice by the delayed-insertion and pruning of an itemset, the memory usage of the *estDec* method remains the same although new transactions are continuously generated. In addition, it decreases as the value of S_{ins} is increased.

Figure 2-(b) shows the average processing time in each interval of the experiment in Figure 2-(a). The processing time is measured by a period from the generation of a new transaction to the delayed-insertion phase (Phases Ⅰ-Ⅲ) of the *estDec* method. As shown in this figure, the average processing time is less than about 15 msec. It is influenced by not only the number of new itemsets whose current support should be estimated for delayed-insertion but also the current size of a monitoring lattice. The number of itemsets maintained in a monitoring lattice is inversely proportional to the value of a threshold S_{ins} as shown in Figure 2-(a). Therefore, as the value of S_{ins} is increased, the average processing time is decreased. Figure 2-(c) shows the average processing time of the frequent itemset selection phase (Phase Ⅳ) in each interval of the experiment in Figure 2-(a). Compared with Figure 2-(b), the average processing time of this phase is considerably larger. This is because it requires to search the entire space of a monitoring lattice. However, it is also decreased as the value of S_{ins} is increased.

A term *average support error* is introduced to model the relative accuracy of the proposed method. When two sets of mining results $R_1=\{(e_i, S_1(e_i)) \mid S_1(e_i)\geq S_{min}\}$ and $R_2=\{(e_j, S_2(e_j)) \mid S_2(e_j)\geq S_{min}\}$ are given for the same data set, the *average support error* $\boldsymbol{ASE(R_2|R_1)}$ of R_2 with respect to R_1 is defined as follows:

$$ASE\ (R_2|R_1)$$

$$= \{ \sum_{e_l\in R_1-R_1\cap R_2} S_1(e_l) + \sum_{e_l\in R_1\cap R_2} (|\, S_2(e_l) - S_1(e_l)\,|) + \sum_{e_l\in R_2-R_1\cap R_2} S_2(e_l)\} / \mid R_1 \mid$$

where $|R_1|$ denotes the number of itemsets in R_1. As the average support error of R_2 gets smaller, the mining result of R_2 is more similar to R_1. To show the accuracy of the *estDec* method, Figure 3 shows $ASE(R_{estDec}|R_{dApriori})$ for the experiment in Figure 2. The mining results of the *estDec* method and the *dApriori* algorithm are denoted by R_{estDec} and $R_{dApriori}$ respectively. The *dApriori* algorithm is the *Apriori* algorithm [10] with the decay mechanism proposed in Section 2. The average support error $ASE(R_{estDec}|R_{dApriori})$ of the *estDec* method is influenced by the value of S_{ins}. As it becomes smaller, more itemsets are maintained in a monitoring lattice, which makes the mining result of the *estDec* method be more accurate. When the value of S_{prn} is set to be less than the value of S_{ins}, the accuracy is slightly improved although this experiment is not presented in this paper.

Figure 4 shows the adaptability of the *estDec* method for the change of information in a data stream. In this experiment, a data set *T5.I4.D1000K-AB* is experimented. The data set is composed of two consecutive subparts. The front part is a set of 500,000 transactions generated by an item set *A* while the second part is a set of 500,000 transactions generated by an item set *B*. There are no common items in the item sets *A* and *B*. In this experiment, S_{min} and *b* are set to 0.001 and 2 respectively. In order to illustrate how rapidly the *estDec* method can adapt the change of information in a data stream, a ***coverage rate CR(X)*** is introduced. It denotes the ratio of frequent itemsets induced by an item set *X* in all frequent itemsets as follows:

$$CR(X) = \frac{\#\text{ of frequent itemsets induced by an item set } X}{|\,R\,|} \times 100(\%)$$

where $|R|$ denotes the total number of frequent itemsets in a monitoring lattice. As the value of a decay-base-life *h* becomes smaller, the *estDec* method adapts more rapidly the transition of information between the two subparts of the data set. By varying a decay-base-life *h*, the adaptability of the *estDec* method for the recent change of a data stream can be controlled. The similar effect can be archived by varying a decay-base *b*. As the value of a decay-base *b* becomes larger, the *estDec* method adapts more rapidly the recent change of a data stream.

6. CONCLUDING REMARKS

Considering the continuity of a data stream, the general definition of finding frequent itemsets used in conventional data mining methodology may not be valid in a data stream. This is because the old information of a data stream may be no longer useful or possibly incorrect at present. In order to support various needs of data stream analysis, the interesting recent range of a data stream needs to be defined flexibly. Based on this range, a mining method can be able to identify when a transaction becomes obsolete and needs to be disregarded. The *estDec* method proposed in this paper finds recent frequent itemsets over an online data stream by decaying the weight of old transactions as time goes by. As a result, the recent change of information in a data stream can be adaptively reflected to the current mining result of the data stream. The weight of information in a transaction of a data stream is gradually reduced as time goes by while its reduction rate can be flexibly controlled. Due to this reason, no transaction needs to be maintained physically.

7. REFERENCES

[1] M. Garofalakis, J. Gehrke and R. Rastogi. Querying and mining data streams: you only get one look. In *the tutorial notes of the 28th Int'l Conference on Very Large Databases*, Hong Kong, China, Aug. 2002.

[2] G. S. Manku and R. Motwani. Approximate frequency counts over data streams. In *Proc. of the 28th Int'l Conference on Very Large Databases*, Hong Kong, China, Aug. 2002.

[3] M. Charikar, K. Chen and M. Farach-Colton. Finding frequent items in data streams. In *Proc. of the 29th Int'l Colloq. on Automata, Language and Programming*, 2002.

[4] C.-H. Lee, C.-R. Lin and M.-S. Chen. Sliding-window filtering: An efficient algorithm for incremental mining. In *Proc. of the 10th Int'l Conference on Information and Knowledge Management*, pages 263-270, Atlanta, GE, Nov. 2001.

[5] B.-K. Yi, N. D. Sidiropoulos, T. Johnson, H. V. Jagadish, C. Faloutsos, and A. Biliris. Online data mining for co-evolving time sequences. In *Proc. of the 16th Int'l Conference on Data Engineering*, pages 13-22, San Diego, CA, Feb. 2000.

[6] H. S. Javitz and A. Valdes. The NIDES statistical component description and justification. Annual report, March 1994.

[7] S. Brin, R. Motwani, J. D. Ullman, and S. Tsur. Dynamic itemset counting and implication rules for market basket data. In *Proc. of the ACM SIGMOD Int'l Conference on Management of Data*, pages 255-264, Tucson, AZ, May 1997.

[8] R. C. Agarwal, C. C. Aggarwal and V.V.V. Prasad. Depth first generation of long patterns. In *Proc. of the 6th ACM SIGKDD Int'l Conference on Knowledge Discovery and Data Mining*, pages 108-118, Boston, MA, Sept. 2000.

[9] C. Hidber. Online association rule mining. In *Proc. of the ACM SIGMOD Int'l Conference on Management of Data*, pages 145-156, Philadelphia, PA, May 1999.

[10] R. Agrawal, and R. Srikant. Fast algorithms for mining association rules. In *Proc. of the 20th Int'l Conference on Very Large Databases*, Santiago, Chile, Sept. 1994.