

## Short Paper

---

# A Sliding Window Method for Finding Recently Frequent Itemsets over Online Data Streams

JOONG HYUK CHANG AND WON SUK LEE

*Department of Computer Science*

*Yonsei University*

*Seoul, 120-749, Korea*

*E-mail: {jhchang, leewo}@amadeus.yonsei.ac.kr*

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 likely to be changed as time goes by. However, most of mining algorithms or frequency approximation algorithms for a data stream do not able to extract the recent change of information in a data stream adaptively. This paper proposes a *sliding window* method of finding recently frequent itemsets over an online data stream. The size of a window defines a desired life-time of the information of a transaction in a data stream.

**Keywords:** recently frequent itemsets, sliding window, data stream, mining data stream, change of data stream

## 1. INTRODUCTION

A data stream is a massive unbounded sequence of data elements continuously generated at a rapid rate. 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 [7].

Recently, various algorithms [4, 5, 7] are actively proposed to extract different types of knowledge embedded in a data stream. Among these, the *Lossy Counting* algorithm [7] is the most representative method for finding frequent itemsets over a data stream. In the Lossy Counting algorithm, the set of frequent itemsets in a data stream is found when an error parameter  $\epsilon$  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 exact current counts of all single items in the data stream are maintained in main memory separately. The local count of every item in the buffer is obtained first and added to its current count. An item whose updated count is less than  $\epsilon \times N$  is not considered to generate a set of local itemsets that appear in the new transactions, where  $N$  denotes the total number of transactions so far including the newly

---

Received June 13, 2003; revised November 13, 2003 & January 29, 2004; accepted February 16, 2004.  
Communicated by Ming-Syan Chen.

generated transactions in the buffer. This is because such an item cannot be a frequent item for the given error  $\varepsilon$ . The local count of every local itemset is identified by scanning the new transactions in the buffer.

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  $(e, f, \Delta)$  where  $e$  is an itemset,  $f$  is the count of the itemset  $e$ , and  $\Delta$  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  $(e, f, \Delta)$  of an itemset  $e$  in  $D$ , if the itemset  $e$  is one of the local itemsets identified by the new transactions in the buffer, its previous count  $f$  is incremented by its local count. 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 local itemset  $e$ , a new entry  $(e, f, \Delta)$  is inserted to  $D$ . Its maximum possible error  $\Delta$  is set to  $\lfloor \varepsilon \times N' \rfloor$  where  $N'$  denotes the number of transactions that were processed up to the latest batch operation. This is because  $\lfloor \varepsilon \times N' \rfloor$  is the maximum possible count that could be missed for the itemset under the given error  $\varepsilon$ . However, it does not differentiate the recent occurrences of each itemset from the old ones. Therefore, an itemset can be regarded as a frequent itemset although it rarely occurs in recent transactions.

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 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. Several sliding window methods [5, 6] are recently proposed to find the knowledge embedded in recently generated data elements. A window is defined by a fixed number of recently generated data elements which is the target of data mining. The *SWF* algorithm [6] uses a sliding window to find frequent itemsets in a 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 a set of 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. In the *SWF* algorithm, to get an up-to-date mining result for entire transactions including newly generated transactions, all the transactions in the current window should be re-scanned entirely. As a result, each transaction in the current window should be scanned as many times as the number of partitions in the window. For a data stream of bits, the concept of a sliding window is utilized in [5] to estimate the count of 1's in recently generated  $N$  bits. Unlike the *SWF* algorithm, it does not physically store the data elements of a data stream. Furthermore, the algorithm is extended to trace the estimated sum of the last  $N$  positive integers in a data stream of integers. In [8], customer's up-to-moment preferences are found for e-commerce recommendation is proposed. In this method, the past preferences are maintained permanently and the up-to-moment preferences are found based on a set of newly generated data elements. Customer's preferences for the entire data elements including the up-to-moment data elements are identified by combining the past preferences with the up-to-moment preferences.

Based on the estimation mechanism of the Lossy Counting algorithm [7], this paper proposes a sliding window method for finding recently frequent itemsets in a data stream when a minimum support  $S_{min} \in (0, 1)$ , an error parameter  $\epsilon \in (0, S_{min})$  and the size of a sliding window  $w$  are given. A recently frequent itemset is an itemset whose support in the transactions within the current sliding window is greater than or equal to  $S_{min}$ . Contrary to the Lossy Counting algorithm that analyses the frequency of an itemset in the transactions generated so far in a data stream, the proposed method analyses the frequency of an itemset within the range of the current window. By restricting the target of a mining process as a fixed number of recently generated transactions in a data stream, the recent change of the data stream can be efficiently analyzed. Consequently, the proposed method can catch the recent change of a data stream as well as analyze the past variation of knowledge embedded in a data stream.

To find recently frequent itemsets accurately over a data stream, the occurrence of every itemset should be carefully maintained. However, it is almost impossible to monitor every itemset that appears in the transactions of a data stream. Such monitoring not only requires a large amount of main memory but also prolongs the processing time for finding frequent itemsets. However, not all of itemsets that appear in a data stream are significant for finding recently frequent itemsets. An itemset whose support is much less than a predefined minimum support is not necessarily monitored since it cannot be a frequent itemset in the near future. In this paper, an itemset whose current support is greater than or equal to an error parameter  $\epsilon$  is defined as a *significant itemset*. Monitoring only significant itemsets can solve the above problems and provide an accurate mining result for a given  $\epsilon$ .

## 2. FINDING RECENTLY FREQUENT ITEMSETS: *Sliding Window* METHOD

In the *sliding window* method, the set of significant itemsets in the transactions of a window are maintained in a prefix-tree lattice structure [2]. In a prefix-tree lattice structure, an itemset is represented by a path and its appearance count is maintained in the last node of the path. In this paper, a prefix-tree lattice in main memory is called as a *monitoring lattice*. Each node of a monitoring lattice contains an entry  $(e, f, t)$  where  $e$  denotes its corresponding itemset,  $f$  denotes the count of the itemset, and  $t$  denotes the *TID* of the transaction which makes the itemset  $e$  be newly inserted into the monitoring lattice. In the Lossy Counting algorithm [7], the data structure  $D$  is maintained in a secondary storage device. However, accessing such a device delays the processing of an online data stream enormously even if it is accessed sequentially. All transactions in the range of the window should be separately maintained by a structure called as a *current transaction list CTL* in order to eliminate the effect of the oldest transaction that becomes out of the window range.

Since the target of data mining in the Lossy Counting algorithm is all the transactions of a data stream, when a new itemset that appears in a new transaction is inserted into a data structure  $D$ , its maximum possible error count is initialized by the number of previous transactions that could be missed for counting the appearance of the itemset. Furthermore, the maximum possible error count of an itemset maintained in the data

structure  $D$  remains the same although new transactions are additionally generated. On the contrary, in the *sliding window* method, the maximum possible error of a new itemset should be initialized by  $\lfloor w \times \varepsilon \rfloor$  since the target of data mining is fixed to  $w$  recent transactions. In addition, the old transactions that are out of the window range should be excluded from the target of data mining. Consequently, the maximum possible error should be decreased as the window slid. As in the Lossy Counting algorithm [7], if the current support of an itemset in a monitoring lattice is less than  $\varepsilon$  in the *sliding window* method, it is not necessary to monitor its count. As a result, its corresponding node is pruned from the monitoring lattice.

**Theorem 1** Given an error parameter  $\varepsilon$ , when  $w_{first}$  denotes the *TID* of the first transaction of the current window, the maximum possible count  $C_k^{max}(e)$  of an itemset with its entry  $(e, f, t)$  is found as follows:

$$C_k^{max}(e) = \begin{cases} f & \text{if } t \leq w_{first} \\ f + \lfloor (t - w_{first}) \times \varepsilon \rfloor & \text{otherwise} \end{cases}$$

**Proof:** If  $t$  is less than or equal to  $w_{first}$ , monitoring the count of the itemset  $e$  is started before the first transaction of the current window. Therefore, there is no possible error count and its count and its maximum possible count  $C_k^{max}(e)$  is  $f$ . On the contrary, if  $t$  is greater than  $w_{first}$ , the total number of transactions generated before the itemset is inserted into the monitoring lattice is  $(t - w_{first})$ . Therefore, its maximum possible error count in these transactions is  $\lfloor (t - w_{first}) \times \varepsilon \rfloor$ , so that its maximum possible count  $C_k^{max}(e)$  is  $f + \lfloor (t - w_{first}) \times \varepsilon \rfloor$ .  $\square$

There are two different phases in the proposed *sliding window* method. One is a *window initialization phase*. This phase is activated while the number of transactions generated so far in a data stream is less than or equal to a predefined window size. Therefore, a new transaction is appended to a current transaction list *CTL* and there is no extracted transaction. The other is a *window sliding phase*. This phase is activated after the *CTL* becomes full. A new transaction is appended to the *CTL* and the oldest transaction is extracted from the *CTL*. The proposed method is composed of five steps: appending a transaction (step 1), Count updating and insertion of new itemsets (step 2), Extracting a transaction (step 3), Pruning of itemsets (step 4), and frequent itemset selection (step 5). These steps except step 4 and step 5 are performed in sequence for a new transaction. Step 3 is performed only in the window sliding phase. Step 4 is usually performed periodically or when it is needed. Step 5 is performed only when the up-to-date set of recently frequent itemsets is requested.

When a new transaction  $T_k$  is generated, it is appended to the current transaction list *CTL*. Only in the window initialization phase, the total number of transactions in the *CTL* is increased by one. In the updating the current transaction list *CTL*, a newly generated transaction should be appended ahead. Otherwise, an itemset pruned by extracting the oldest transaction may need to be inserted again when the new transaction is appended.

**Step 1: Appending a transaction** The transaction  $T_k$  is appended to the current transaction list  $CTL$ .  $\square$

Subsequently, in the step for count updating and insertion of new itemset, those paths in the monitoring lattice that are induced by the items of the transaction  $T_k$  are traversed respectively. If an itemset is maintained in the monitoring lattice, the count  $f$  of its corresponding node is incremented by 1. On the other hand, for each itemset  $e$  not maintained in the monitoring lattice, its corresponding node is inserted to the monitoring lattice.

**Step 2: Count updating and insertion of new itemsets** For an itemset  $e$  that appears in the new transaction  $T_k$  with an entry  $(e, f, t)$ , if its corresponding node is in the monitoring lattice, the count  $f$  of the corresponding node is increased by one i.e.,  $e.f = e.f + 1$ . Moreover, for a new itemset  $e$  induced by the items of the new transaction  $T_k$  is inserted into the monitoring lattice with an entry  $(e, f = 1, t = k)$ .  $\square$

Due to the characteristic of the estimation mechanism of the Lossy Counting algorithm, when a new transaction is generated, every new itemset in the transaction should be inserted into a monitoring lattice. Therefore, the number of monitored itemsets may be increased greatly when a new transaction contains many new items. Therefore, not all the nodes of a monitoring lattice may be maintained in main memory. In this case, some of the nodes should be maintained in a secondary storage device as in the Lossy Counting algorithm.

When the current window in the window sliding phase, the oldest transaction in the current transaction list  $CTL$  is extracted. Subsequently, each itemset in the monitoring lattice that is induced by the items of the extracted transaction is traversed. While traversing, the effect on the count of its corresponding node by the extracted transaction is diminished.

**Step 3: Extracting a transaction: only in the window sliding phase** The oldest transaction in the current transaction list  $CTL$  is extracted. At the same time, for an itemset  $e$  that appears in the oldest transaction in the  $CTL$ , if its corresponding node with an entry  $(e, f, t)$  is in the monitoring lattice, the count  $f$  of the corresponding node is updated as follows:

$$e.f = e.f - 1 \quad (\text{if } t \leq w_{\text{first}}), \quad e.f = e.f \quad (\text{otherwise})$$

If its maximum possible count  $C_k^{\text{max}}(e)$  becomes less than  $\lceil w \times \varepsilon \rceil$ , it is an insignificant itemset, so that it is pruned from the monitoring lattice. This because its maximum possible support  $C_k^{\text{max}}(e)/|D^w|_k$  is less than an error parameter  $\varepsilon$  if its maximum possible count  $C_k^{\text{max}}(e)$  is less than  $\lceil w \times \varepsilon \rceil$ , where  $|D^w|_k$  denotes the total number of transactions in the current window.  $\square$

On the other hand, in order to reduce the number of itemsets that are should be maintained in the monitored, insignificant itemsets can be pruned from the monitoring lattice periodically or when it is needed. That is, among the itemsets maintained in the

monitoring lattice, all insignificant itemsets are pruned by traversing all the paths of the monitoring lattice.

**Step 4: Pruning of itemsets** For an itemset  $e$  with an entry  $(e, f, t)$  in the monitoring lattice, if its maximum possible support  $C_k^{max}(e)/|D^w|_k$  is less than an error parameter  $\varepsilon$ , it can be regarded as an insignificant itemset and it is pruned from the monitoring lattice. That is, if its maximum possible count  $C_k^{max}(e)$  is less than a pruning threshold, it is pruned from the monitoring lattice. The pruning threshold in the window initialization phase is  $\lceil |D^w|_k \times \varepsilon \rceil$  and that in the window sliding phase is  $\lceil w \times \varepsilon \rceil$  since  $|D^w|_k = w$ .  $\square$

The frequent itemset selection step is performed only when the mining result of the current window is requested. All the currently frequent itemsets in the monitoring lattice are found by traversing all the paths of the monitoring lattice as in conventional mining methods [2] based on a prefix-tree lattice structure.

**Step 5: Frequent itemset selection** For an itemset  $e$  with an entry  $(e, f, t)$  in the monitoring lattice, if its maximum possible support  $C_k^{max}(e)/|D^w|_k$  is greater than or equal to  $S_{min}$ , it is a frequent itemset.  $\square$

If the size of a window is less than or equal to  $1/\varepsilon$ , the support of an itemset that appears at least once in the transactions of the current window is greater than or equal to the maximum possible error  $\varepsilon$ . Therefore, no itemset is pruned. Consequently, the number of itemsets in a monitoring lattice is monotonically increased in this case. Therefore, the size of a window should be greater than  $1/\varepsilon$  in order to distinguish a set of significant itemsets from the total set of itemsets formed by the transactions in the window.

### 3. EXPERIMENTAL RESULTS

In this section, two data sets in Fig. 1 are used to evaluate the performance of the proposed method. Following the conventions set forth in [1], the names of the data sets are *T5.I4.D1000K-I* and *T5.I4.D1000K-II* where the three numbers of each data set denote the average transaction size ( $T$ ), the average maximal potentially frequent itemset size ( $I$ ) and the total number of transactions ( $D$ ) respectively. In all experiments, the transactions of each data set are looked up in sequence to simulate the environment of an online data stream. All experiments are performed on a 2.0 GHz Pentium PC machine with 512MB main memory running on Linux 7.3 and all programs are implemented in C.

Figs. 2 and 3 show the memory usage of the *sliding window* method for the data set *T5.I4.D1000K-I*. A minimum support  $S_{min}$  and the size of a window  $w$  are set to 0.001 and 20000 respectively. A pruning operation is performed in every 1000 transactions. The sequence of generated transactions is divided into 5 intervals each of which consists of 200000 transactions. The memory usage is represented by the maximum number of itemsets in a monitoring lattice for each interval. Fig. 2 shows the memory usage of the window initialization phase. In this phase, the memory usage of the proposed method is monotonically increased until every  $\lceil 1/\varepsilon \rceil^{th}$  transaction is processed. However, it is rapidly dropped when a subsequent pruning operation in step 4 is performed. This is be-

TID	1 – 500000	500001 – 1000000
T5.14.D1000K- I	TA	
T5.14.D1000K- II	TA	TB

♣ TA denotes a set of transactions generated by a set of items A and TB denotes a set of transactions generated by a set of items B. There is no common item between the two sets of items A and B. Each set of items is generated by the same method as described in [1] and the total number of items is 1000.

Fig. 1. Allocation of transactions in data sets.

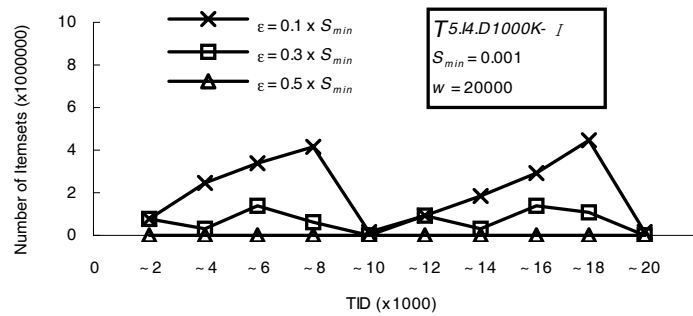


Fig. 2. Memory usage in the window initialization phase.

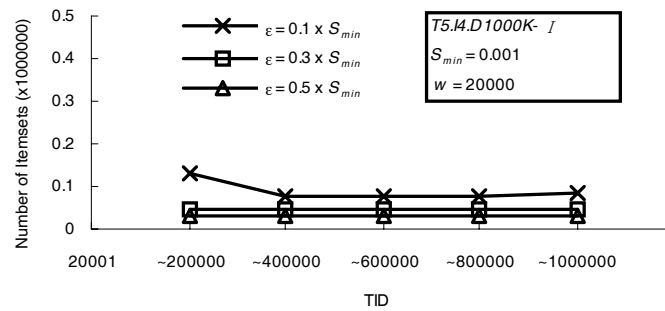


Fig. 3. Memory usage in the window sliding phase.

cause a newly inserted itemset is pruned when it does not appear in any of the subsequent  $\lceil 1/\epsilon \rceil$  transactions. Fig. 3 shows the memory usage of the window sliding phase. The memory usage of this phase remains almost the same although new transactions are continuously processed. This is because only those itemsets whose current supports are greater than  $\epsilon$  are maintained in the monitoring lattice.

To measure the relative accuracy of the proposed method, an *average support error ASE* presented in [3] is used. Fig. 4 shows the average support error of the mining result of the proposed method with respect to that of the *Apriori* algorithm performed on the transactions within the current window in each interval of the experiment in Fig. 3 by

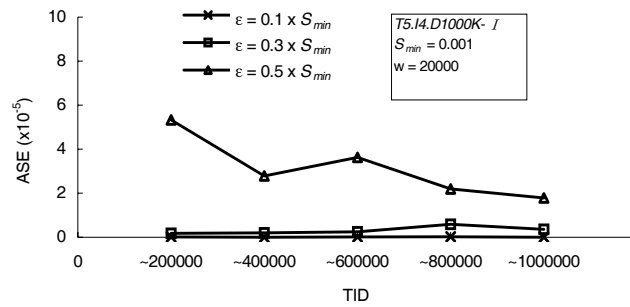


Fig. 4. Accuracy of mining results.

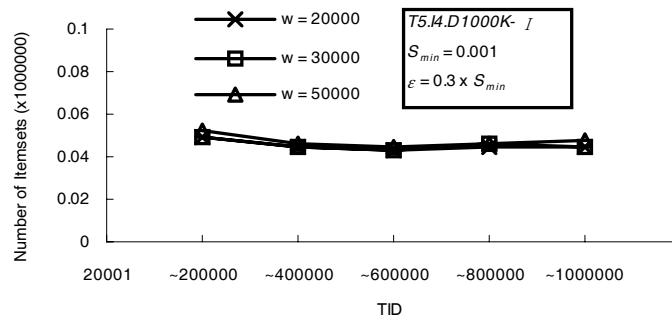


Fig. 5. Memory usage in the window sliding phase by varying the size of a window.

varying an error parameter  $\varepsilon$ . Generally, the more itemsets are maintained in a monitoring lattice, the more accurate the mining result is. Since the number of itemsets maintained in a monitoring lattice is inversely proportional to the value of  $\varepsilon$  as shown in Fig. 3, the average support error is increased as the value of  $\varepsilon$  is increased

Fig. 5 shows the memory usage of the window sliding phase for the data set *T5.I4.D1000K-I* by varying the size of a window. The values of  $S_{min}$  and  $\varepsilon$  are set to 0.001 and  $0.3 \times S_{min}$  respectively. The data set is looked up by the same sequence as in Fig. 3. As shown in this figure, the memory usage of the *sliding window* method remains almost the same. This is because the number of significant itemsets in the current window is almost the same although the size of a window  $w$  is enlarged.

Fig. 6 shows the adaptability of the *sliding window* method for the recent change of information in a data stream. In this experiment, the data set *T5.I4.D1000K-II* is used. The values of  $S_{min}$  and  $\varepsilon$  are set to 0.001 and  $0.1 \times S_{min}$  respectively. A pruning operation is performed in every 10000 transactions. In order to illustrate how rapidly the *sliding window* method can adapt the change of information in a data stream, a *coverage rate CR* presented in [3] is used. As the size of a window becomes smaller, the *sliding window* method adapts more rapidly the transition of recent information between the two subparts of the data set. By varying the size  $w$ , its adaptability for the recent change of a data stream can be controlled.

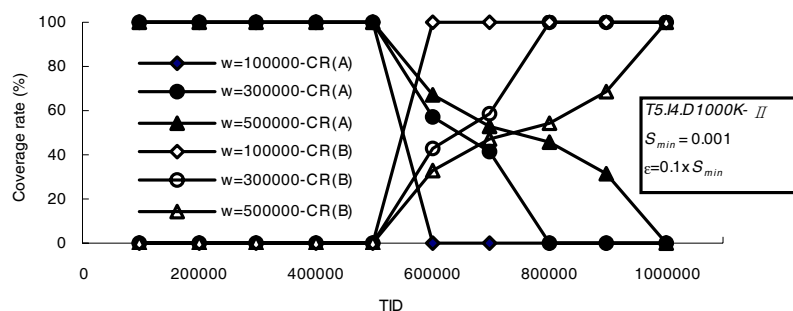


Fig. 6. Variations of the coverage rates.

#### 4. CONCLUDING REMARKS

Considering the continuity of a data stream, 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, an algorithm can be able to identify when a transaction becomes obsolete and needs to be disregarded. This paper proposes a method for finding recently frequent itemsets over a data stream based on a sliding window. The interesting recent range of a data stream is defined by the size of a window. The proposed method can be employed to monitor the recent change of embedded knowledge in a data stream.

#### REFERENCES

1. R. Agrawal and R. Srikant, "Fast algorithms for mining association rules," in *Proceedings of the 20th International Conference on Very Large Databases*, 1994, pp. 487-499.
2. S. Brin, R. Motwani, J. D. Ullman, and S. Tsur, "Dynamic itemset counting and implication rules for market basket data," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 1997, pp. 255-264.
3. J. H. Chang and W. S. Lee, "Finding recent frequent itemsets adaptively over online data streams," in *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2003, pp. 487-492.
4. M. Charikar, K. Chen, and M. Farach-Colton, "Finding frequent items in data streams," in *Proceedings of the 29th International Colloquium on Automata, Language and Programming*, 2002, pp. 693-703.
5. M. Datar, A. Gionis, P. Indyk, and R. Motwani, "Maintaining stream statistics over sliding windows," in *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms*, 2002, pp. 635-644.
6. C. H. Lee, C. R. Lin, and M. S. Chen, "Sliding-window filtering: an efficient algorithm for incremental mining," in *Proceedings of the 10th International Conference on Information and Knowledge Management*, 2001, pp. 263-270.

7. G. S. Manku and R. Motwani, "Approximate frequency counts over data streams," in *Proceedings of the 28th International Conference on Very Large Databases*, 2002, pp. 346-357.
8. Y. D. Shen, Q. Yang, Z. Zhang, and H. Lu, "Mining the customer's up-to-moment preferences for e-commerce recommendation," in *Proceedings of the 7th Pacific-Asia Conference on Knowledge Discovery and Data Mining*, 2003, pp. 166-177.

**Joong Hyuk Chang** received the B.S. and M.S. degree in Computer Science from Yonsei University, Seoul, Korea, in 1996 and 1998. He is currently a Ph.D. candidate at the Department of Computer Science, Yonsei University, Seoul, Korea. His current research interests include mining data streams, query processing in data streams, data stream management systems, and data mining in databases.

**Won Suk Lee** received the B.S. degree in Computer Engineering from Boston University, Boston, MA and the M.S. and Ph.D. degrees in Electrical and Computer Engineering from Purdue University, West Lafayette, IN. He was a senior staff engineer at Computer Division, Samsung Electronics. He is currently an associate professor of Department of Computer Science at Yonsei University, Korea. His current research interests include data streams, data mining, anomaly intrusion detection, and mediator systems.