International Journal of Control and Automation
Vol. 11, No. 8 (2018), pp.85-94
http//dx.doi.org/10.14257/ijca.2018.11.8.08

Finding Frequent Itemsets over Data Streams with Sliding Window

Jeong Hee Hwang[1], Hyeok Kim[2] and Jeong Hee Chi[3*]

[1]*Department of Computer Science, Namseoul University, Korea*
[2,3]*Department of Software, Konkuk University, Korea*
[1]*jhhwang@nsu.ac.kr,* [2,3]*{gkdl1599, jhchi}@konkuk.ac.kr*

Abstract

Frequent pattern mining is the area studied the most in the data stream mining area, and it is important to explore frequent itemsets rapidly. This paper proposes a method to rapidly explore frequent items by expressing transaction items as bit sequence, based on sliding window which considers a certain number of transactions as window to explore frequent itemsets in data stream. Frequent itemsets are explored focusing on items changed by sliding window by expressing transaction items deleted and inserted by sliding window as bit sequence and applying bitwise exclusive OR operation. As the proposed method discovers frequently generated items focusing on the changed items using sliding window, it improves performance of mining, compared with existing algorithms. It was found by experiments that it improves performance at least 13% over the performance using existing algorithms using bit sequence.

Keywords: *Data stream, Data mining, Real-time mining, Frequent pattern*

1. Introduction

Data stream mining is the big issue in data mining. A large amount of data stream is generated real-time. Some examples are sensor data in sensor network, medical data in the medical field, web record and web click stream data of web application. Data stream is an infinite set where items are continuously generated, and a large amount data is generated rapidly real-time. To analyze data generated real-time, we need data stream mining technique [1-12]. Data stream mining is the technique of getting useful information by processing data transmitted real-time, and frequent itemset mining belongs to data stream mining technique. It is the technique to find out what kind of data is generated the most, or what kind of data is generated above a certain level of frequency [2-7].

Stream data has the following characteristics. Data stream is continuous and unlimited. Unlike static data, this kind of data is generated limitlessly, and there is no division on transaction. In the stream, data distribution changes over time. And, people are usually interested in recent data. Owing to such characteristics of data, we cannot make knowledge exploration using the whole stream data. Thus, in order to useful knowledge from data stream, we need a technique to explore it real-time.

Considering the continuous characteristics of data stream, we can define a certain of transaction as a window. Sliding window method is applying data, using a window with a static size in data stream [15]. If a user requests information, sliding window-based mining mines all the frequent pattern items satisfying the threshold from recent data stored in the window. Therefore, as data is stored in regular window, it is possible to save memory.

Received (January 7, 2018), Review Result (March 13, 2018), Accepted (July 12, 2018)
* Corresponding Author

ISSN: 2005-4297 IJCA

Frequent pattern is the pattern frequently generated pattern in a dataset, that is, item set. It is important to discover a frequent pattern in exploring connectedness of data and data stream prediction. In the case of most of frequent pattern algorithms, mining is performed using tree structures like FP-growth and prefix tree. In [16], Closed Enumeration Tree (CET) which maintains only dynamically closed frequent itemsets which is different from prefix tree maintaining all itemsets is suggested. And, [17] suggests a method of not constructing a new tree, but restructuring tree using only transactions which will be deleted and transactions which will be inserted. The proposed method uses bit and makes it possible to explore real-time frequent patterns by reconstructing deleted and inserted parts, instead of restructuring the whole tree by using bit. But, it is costly because it requires that the order of inserting all the transactions be expressed with bits, and that, whenever a new transaction is inserted, it traverse the whole tree.

[18] proposed MFI-TimeSW algorithm which detects frequent itemsets real-time within sliding window using bit sequence expression of item. By expressing whether items contained in transactions within a window as 0, 1, that is, as bit sequence, and using bitwise and operator, it discovers items of frequent pattern using bitwise and operator.

This paper, to extract real-time useful information from stream data, expresses items contained in transaction as bit sequence, which is similar to the method of [18]. But, to detect items changed by sliding window, we propose a method to detect frequent items using bitwise Exclusive OR(XOR) operation. That is, unlike [18] which does operation on all the transactions in a related window, our method discovers frequent items by doing bit operation focusing on items in transaction deleted and inserted by sliding window. As the method proposed in this paper performs bit operation on items changed in transactions, it reduces cost of memory use and improves performance speed of frequent item mining.

2. Related Works

As stream data has the characteristics that, as it is generated continuously and rapidly, and it has no limit in its length, it is difficult to process it referring to existing data. Such characteristics necessitates minimal access to data and quick processing. There is limits in storing the whole data in memory and treat it. To solve such problems, various stream mining techniques have been suggested such as landmark window model [13], damped model [14], and sliding window model [15].

Among data mining techniques, frequent pattern mining is the method of detecting pattern-typed meaningful information from a huge database. Patterns are itemsets, and an item is a good in market database, or a disease or symptom in medical database. Traditional frequent pattern mining mines information on patterns which occur over minimum frequency the user FP-Growth is to overcome the scan problem of Apriori, the best-known technique for frequent pattern mining mines frequent pattern items without generating candidate through two times scanning of data base using tree structure-based depth first search method.

And, prefix-tree structure is the structure of lexicographic order, and allows one to insert transaction data without restructuring tree. But, in the method of exploring frequent item set using prefix-tree, as tree should be managed in the memory while exploration is performed, the tree size should be smaller than limited memory space. As the tree structure to detect items of frequent pattern depends on the number of itemsets above minimum support showing in data stream, if the tree size showing itemsets which should be managed is bigger than limited memory size, it becomes problematic.

[16] constituted compact prefix-tree-based CET and includes four types of nodes: infrequent gateway nodes, unpromising gateway nodes, intermediate nodes, and closed nodes. When a new transaction arrives, moment algorithm traverse nodes related with items of the new transaction of CET, and renews support, tid-sum about related nodes. And, when the oldest transaction in the current window is deleted, moment algorithm

traverses related nodes and judges the type of related nodes. Tree exploration and node type checking are expensive. [19] proposed CFP-tree (compressed FP-tree) to process a large number of transactions. The method makes it possible to do incremental mining without re-scanning of original database. But, it has problem of restructuring Item-Frequent list (IF-list) and CFP-tree. [18] expressed item-containing transactions as bit sequence and detects itemsets of frequent pattern using bitwise AND operator on the bit sequence. It was found that the method of bit sequence in [18] needs smaller memory and increases performance speed compared with [17].

3. Frequent Pattern Mining using Sliding Window-Based Bit Operation

We consider a certain number of transactions as a window, and describes a method to quickly find frequent items in the transactions contained in the related window. If sliding window is defined as the window size containing 3 transactions for the exampled data sets in Figure 1, transactions contained in the series of windows are $W_1=\{T_1, T_2, T_3\}$, $W_2=\{T_2, T_3, T_4\}$, $W_3=\{T_3, T_4, T_5\}$, as shown in Figure 2.

TID	Item
T1	a, c, d
T2	b, c
T3	a, b, c, e
T4	a, c, d
T5	a, b, c, d
T6	b, c

Figure 1. Transaction Data

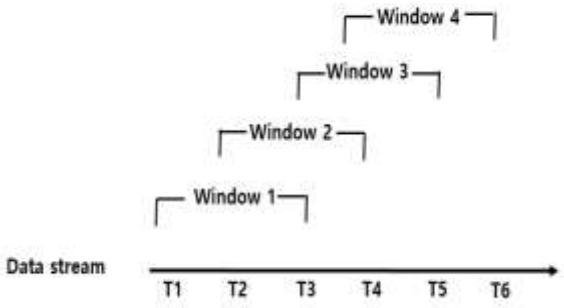


Figure 2. Example of Window Sliding

3.1. Detecting the Frequent Itemsets in a Window

The mining process using sliding window-based bit operation to detect frequent items in the N data stream transactions is as follows. The basic definition to detect frequent itemsets from transactions contained in a window are as follows. Item set is expressed as $I=\{i_1, i_2, i_3, .., i_i\}$, and transaction is expressed as $T=\{ T_1, T_2, T_3, .., T_n\}$.

Definition 1. Window Size

A window contains a certain number of transactions. The number of transactions contained in the window is defined as window size, and expressed as $|W|$. And, a window has a serial number.

Definition 2. Frequent Pattern Itemset

The ratio of the number of transactions containing item I_i in a window is called support of the item, and expressed as Supp(I_i). If Supp(I_i) satisfies pre-defined minimum support min_sup, it is called frequent item. The equation expression of them is as follows equation 1.

$$Supp(I_i) = \frac{|T_{\ni I_i}|}{|W|} \geq \min\_\sup \ (0 < \min\_sup \leq 1) \quad (1)$$

To extract frequent pattern items from transactions in a window, those items are expressed as bit sequence. In Figure 1, $W_1=\{T_1, T_2, T_3\}$, $T_1=\{a, c, d\}$, $T_2=\{b, c\}$, $T_3=\{a, b, c, e\}$. Items can be expressed as bits, as follows: Bit(a) = 101, Bit(b) = 011, Bit(c) = 111, Bit(d) = 100, Bit(e) = 001. The method to express an item as bit sequence is as follows: *tid* number matches the order of bit sequence. If an item is contained in the transaction, it

is expressed as 1, and, if it is not included in the transaction, it is expressed as 0. For example, Bit(*a*) = 101 means that a exists in *T1* and *T3*, but not in *T2*.

If we assume that minimum support for the transaction contained in W_1 is 0.5, frequent items satisfying min_sup among items expressed as bit sequence Bit(*a*) = 101, Bit(*b*) = 011, Bit(*c*) = 111, Bit(*d*) = 100, Bit(*e*) = 001 are *a*, *b*, *c*. Using bitwise AND operation, we discover length_2 items from these items. Bit(*a*) = 101 and Bit(*b*) = 011 => 001, Bit(*a*) = 101 and Bit(*c*) = 111 => 101, Bit(b) = 011 and Bit(c) = 111 => 011. Thus, length_2 items satisfying min_sup are *ac*, *bc*. Figure 3 summarizes above discussions.

Freq. 1_item		Freq. 2_item	
item	Supp	item	Supp
Bit(*a*) = 101 Bit(*b*) = 011 Bit(*c*) = 111	2 2 3	Bit(*a*) = 101 and Bit(*c*) = 111 => 101 Bit(*b*) = 011 and Bit(*c*) = 111 => 011	2 2

Figure 3. Frequent Items and Frequency in W1

As Figure 3 shows, the method to explore frequent items among transaction items included in a window is to express each item as bit sequence indicating its order in the transaction, and extract length_1 item satisfying min_sup, and find length_2 item with the extracted item. From length_2 item, we need to check whether each item satisfies critical value using bitwise AND operator to bit sequence expressing the location where it is generated, and determines frequent item. Algorithm of it is as follows algorithm 1.

Algorithm 1. Detect Frequent Itemsets by Bit Sequence (FIBS)

Input : $|w|$, T_k in w_i, I_j in each transaction T_k , *min_sup*
Output : frequent itemsets FIS

Step 1: detect the length_1 frequent items
1. initiate window w_i
2. for each item $I_j \in$ T in w_i
   translate items into Bit sequence, Bit(I_j)
3. if Supp(I_j) $\geq$ *min_sup*
   put item into FIS

Step 2: detect the frequent itemsets more than length_2 items
1. for each item in FIS
   compute Supp(I_iI_j) by bitwise Bit(I_i) and Bit(I_j)
   if Supp(I_iI_j) $\geq$ *min_sup*
   put item I_iI_j into FIS
2. repeat step 1 by increasing the length of item until no more frequent itemsets

3.2. Insert and Delete a Transaction by Sliding Window

Data stream is infinite set composed of transactions which are generated continuously. Consequently, it is impossible to store all the transaction in a data stream, it is necessary to generate mining results by scanning all the information contained in a transaction.

As the number of transactions contained in a window is fixed, if a new transaction is inserted by sliding window, the oldest transaction is deleted. We illustrate the method of detecting frequent items in the inserted transaction and the deleted transaction with transaction example of Figure 4. If window size is 4, the formula is *W1={T1, T2, T3, T4}*. If items contained in *T1* to *T4* are expressed as bit sequence, they can be expressed as Bit(*a*)= 1110, Bit(*b*)= 1010, Bit(*c*)= 1111, Bit(*d*)= 0101, Bit(*e*)= 0101. And, from items

expressed as bit sequence, frequent items are detected using FIBS algorithm as shown in Figure 5, and the detected items are stored in hash table. Keys of hash table is ordered as arranged length_1 items, and frequent items including related items are stored in the order of their lengths. To rapidly reflect the changes of frequent items by sliding window, it is necessary to store frequent items in efficient hash table, which allows detection, insertion, and deletion of frequent items to be done.

By sliding window, transaction *T5* is inserted and *T1*, the oldest one, in *W2* is deleted. *T1* is efficiently deleted by bitwise left shift operation. ***W2 ={****T2,T3,T4,T5}*, and new inserted transaction *T5* includes item set*{a, b, f}*. To quickly detect frequent items in the deleted transaction and the inserted transaction, we use bitwise XOR operation.

The processes of detecting frequent items on changes of transactions are as follows. First, items in *T1*, the deleted transaction, and *T5*, the inserted transaction, are expressed as bit sequence in columns: *T1={111000}, T5={100101}*. *T1* contains a, b, c items, and T5 contains a, d, f. Second, bitwise XOR operation is applied to *T1* and *T5* whose items are expressed as bit sequence to detect changed items. That is, *T1={111000}* XOR *T5={100101} = {011101}*. Thus, we can notice that there are changes in *b, c, d, f* items. Third, calculate supports for the changed items *b, c, d,* and *f*, and, delete frequent items including related items which do not satisfies min_sup from hash table. As Supp(*b*)=1 and Supp(*f*)=1, items *b* and *f* do not satisfy the threshold that min_sup=0.5, they are deleted. Forth, among changed items (*b, c ,d, f*), *b, f* were deleted because they did not satisfy the threshold, and among the remaining values *c* and *d*, *d* which comes later in the item order, becomes the standard in detecting the scope of frequent items. That is, applying FIBS algorithm, we explore frequent items among items *a*, *c*, *d* (*b* is deleted because it does not satisfy min_sup.). Consequently, we leave e, frequent item in *W1*, as a frequent item, and we do not explore the frequent item any more. Because the items coming after *d*, which is standard in the item order, do not change, it is not necessary to explore them again.

Tid	items
T1	a b c
T2	a c d e
T3	a b c
T4	c d e

Figure 4. Transaction in W1

Key	Frequent Itemsets
a	a, ab, ac, abc
b	b, bc
c	c, cd, ce, cde
d	d, de
e	e

Figure 5. Frequent itemsets in W1

In such a way, through the changes of window sliding, we detect change of items contained in deleted transaction and newly inserted transaction. As we detect item changes and whether an item occurs frequently by using bitwise XOR operation on bit sequence of items, and we only check frequency only for the items satisfying threshold values, we need not re-explore all the items of transactions belonging to new window. So, by reducing exploration scope, we improve performance of frequent item exploration.

4. Experimental Results

The experiment compares FIBS suggested in this paper and TimeSW algorithm suggested in [15]. The experiment was performed on the Window 10, 16GB RAM, 3.40GHz CPU system. As experimental datasets, we used T10I4100K and T40I10D100K created by the data generator of the IBM Almaden Quest Research Group. Parameters for the experiment are listed in Table 1.

Table 1. Simulation Parameters

Parameter	Description	Value
D	Number of transactions in data streams	100K
N	Number of distinct items	1K
I	Average length of maximal frequent itemsets	4, 10
T	Average length of transactions	10, 40
s	Minimum support thresholds	0.01~0.1
w	Window Size	5K~10K

The number of transactions in each dataset is 100K, and the number of items is 1K. The length of transactions ranges from 10 items to 40 items. The numbers of the maximum frequent items consist of 4 and 10. We tested them while changing the critical value of minimum support s from 0.01 to 0.1, and window sizes from 5k to 10k.

First, we checked the trend of changes of frequent items. While existing TimeSW seeks frequent items in every window, FIBS algorithm performs frequency checks only for items which are inserted or deleted. If there is a large number of changed items, FIBS technique needs many operations like TimeSW. So, as shown in Figure 6, we observed the trend of changes of the single frequent item in each data.

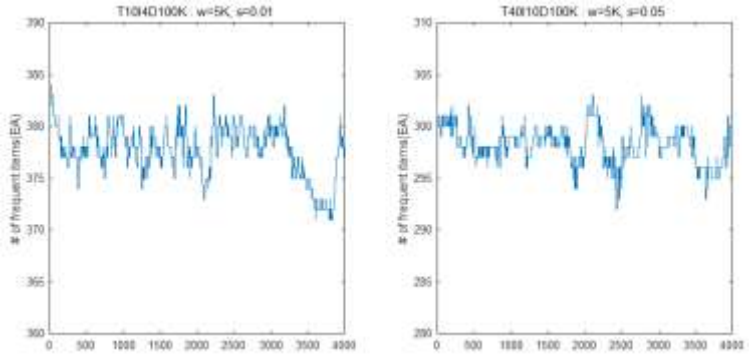


Figure 6. Changes in the Number of Frequent Items

Figure 6 shows part of trend changes of frequent items in the following cases: the case of T10I4D100K's w=5K, s=0.01; the case of T40I10D100K's w=5k, s=0.05. In T10I4D100K, the average number of items constituting a transaction is 10, and the numbers of frequent items ranges from 370 to 385. In T40I10D100K, the average number of items constituting a transaction is 40. As it is a dataset where many items are occurred at the same time, the experiment showed that, when s is low, there is little change in frequent items. The experiment changed in a range from 290 to 303. The experiment showed that many frequent items in each window do not change.

To test performance of FIBS, we experimented generation time of frequent items depending on the size of window as illustrated in Figure 7 and Figure 8. Figure 7 is the findings of T10I4D100K dataset. In this case, as the window size becomes larger, the number of transactions satisfying minimum support also gets larger. Thus, processing time tended also to increase. This finding shows that FIBS is considerably better than TimeSW. Such findings are caused by the fact that in the data where the number of items in a transaction is small, the changes of frequent items rarely occur. And, with the increase of minimum support, the difference of performance between FIBS and TimeSW rapidly increases from 42% to 92%. The reason is from the fact that as support gets larger, the number of items satisfying minimum support decreases, and frequent items do not change. These findings tell us that FIBS perform excellently in the case where frequent items do not change.

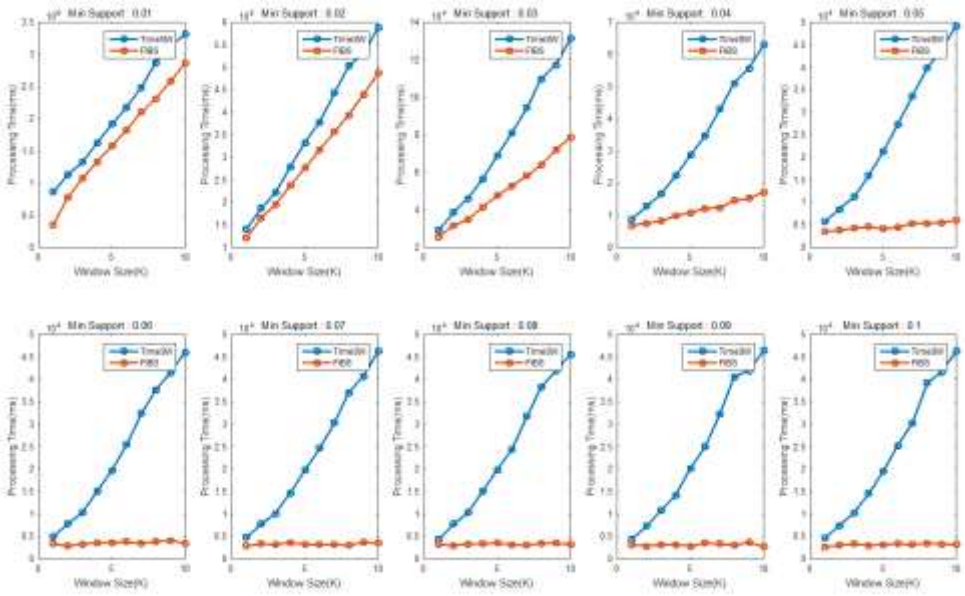


Figure 7. Processing Time According to Minimum Support with T10I4D100K Dataset

Figure 8 shows that FIBS perform more excellently than TimeSW, from T40I4D100K data. This experiment is to evaluate FIBS performance in the data where there are many items per transaction, and there are many changes in items in each transaction. It was found that, in the case where many changes of items occur within a transaction as well, FIBS perform more excellently than TimeSW.

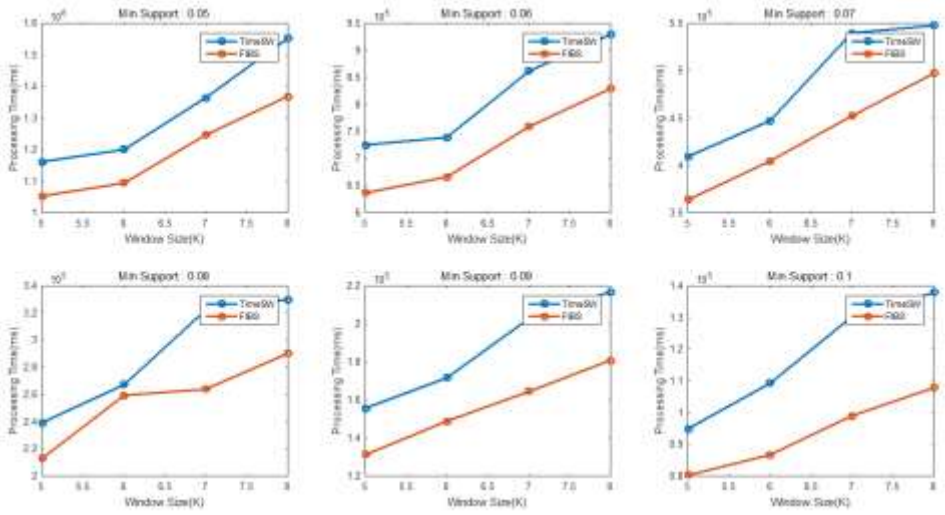
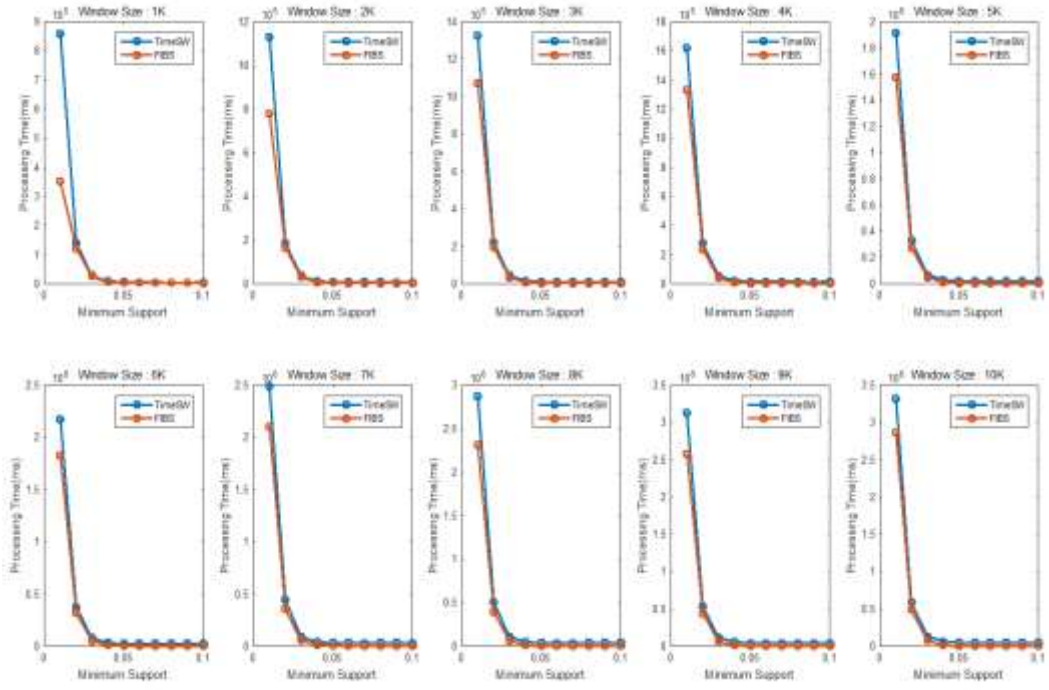


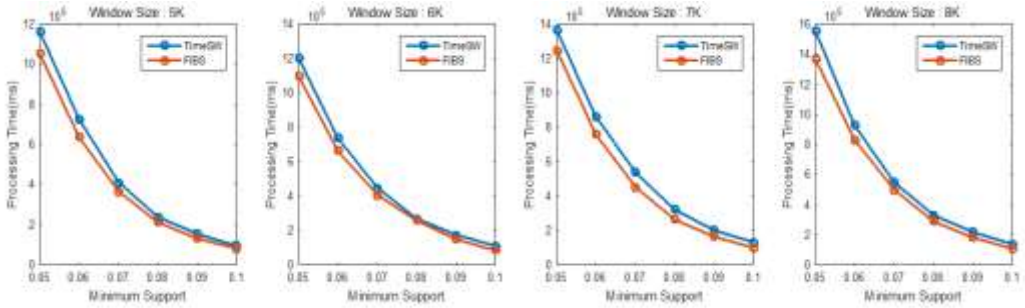
Figure 8. Processing Time According to Minimum Support with T40I10D100K Dataset

We also compared performance time depending on minimum support. As the minimum support gets higher, the number of frequent items decreases. Thus, these tests evaluate performance times. It is shown that FIBS perform better than TimeSW, based on T10I4D100K. In particular, when s=0.1, as the number of items satisfying minimum support increases, performance of FIBS improves from minimum 13% to maximum 59%. The reason is from the fact that, as minimum support gets lower, more items get frequent. So, the changes of frequent items get smaller. And also, we conduct an experiment for the

data, T40I10D100K where there are smaller changes in frequent items, and it was found that, in the case where many changes of items occur within a transaction as well, FIBS perform more excellently than TimeSW.



(a) T10I4D100K Dataset



(b) T40I10D100K Dataset

Figure 9. Processing Time According to Window Size

We compared performance time depending on minimum support. As the minimum support gets higher, the number of frequent items decreases. Thus, this test evaluates performance times. Figure 9(a) is based on T10I4D100K. It is shown that in all the cases, FIBS perform better than TimeSW. In particular, when s=0.1, as the number of items satisfying minimum support increases, performance of FIBS improves from minimum 13% to maximum 59%. The reason is from the fact that, as minimum support gets lower, more items get frequent. So, the changes of frequent items get smaller. Figure 9(b) is based on T40I10D100K. This is an experiment for the data where there is smaller changes in frequent items, and it was found that, in this case as well, FIBS performs better than TimeSW. The above experiments proved that, compared with TimeSW, FIBS which only checks frequency on item changes within transaction is better, whether the changes in frequent items are big or small.

5. Conclusions

This paper proposed a method to find frequent itemset on data stream based on sliding window. The proposed method expresses transaction items as bit sequence, and detects frequent itemset inserted or deleted by window sliding using bitwise exclusive OR operation. To evaluate the performance of the algorithm proposed in this paper, we did experiment comparing it with existing algorithm. The experiment showed at least 13% improvement of performance over the existing algorithm. This finding is from the fact that, unlike the existing method where all the transactions are explored again and again whenever window sliding is applied, the method proposed in this paper detects frequent items by selecting items which have changed.

Acknowledgments

This research was supported by the MSIT(Ministry of Science and ICT), Korea, under the Seoul Accord Vitalization Program(IITP-2018-2012-1-00593) supervised by the IITP(Institute for Information & communications Technology Promotion).

References

[1] S. Muthukrishnan, "Data streams: algorithms and applications", Foundations and Trends® in Theoretical Computer Science, vol. 1, no. 2, **(2005)**, pp. 117-236.
[2] G. Cormode and S. Muthukrishnan, "What's hot and what's not: tracking most frequent items dynamically", Proceeding of the twenty-second ACM SIGMOD-SIGART symposium on Principles of database systems, **(2003)**, pp. 296-306.
[3] R. Karim, M. Cochez, O. Beyan, C. F. Ahmed and S. Decker, "Mining maximal frequent patterns in transactional databases and dynamic data streams: A spark-based approach", Information Sciences, vol. 432, **(2018)**, pp. 278-300.
[4] C. Jin, W. Qian, C. Sha, J. X. Yu and A. Zhou, "Dynamically maintaining frequent items over a data stream", Proceeding of the 2003 ACM CIKM International Conference on Information and Knowledge Management, **(2003)**, pp. 287-294.
[5] G. S. Manku and R. Motwani, "Approximate frequency counts over data streams", Proceeding of the 28th International Conference on Very Large Data Bases, **(2002)**, pp. 346-357.
[6] E. Demaine, A. Lopex-Ortiz and J. Munro, "Frequency estimation of internet packet streams with limited space", Proceeding of the 10th Annual European Symposium, **(2002)**, pp.348-360.
[7] A. Metwally, D. Agrawal and A. E. Abbadi, "Efficient computational of frequent and top-k elements in data streams", Proceeding of the 10th International Conference on Database Theory, **(2005)**, pp. 398-412.
[8] G. Chen, X. Wu and X. Zhu, "Mining Sequential Patterns Across Data Streams", University of Vermont Computer Science Technical Report(CS-05-04), **(2005)**.
[9] M. C. Hsieh, Y. H. Wu and A. L. Chen, "Discovering Frequent Tree Patterns over Data Stream", Proceeding of SIAM International Conference on Data Mining, **(2006)**, pp. 629-633.
[10] A. Deligiannakis, Y. Kotidis and N. Roussopoulos, "Hierarchical In-Network Data Aggregation with Quality Guarantees", LNCS(EDBT 2004), **(2004)**, pp. 658-675.
[11] C. K. Chiou and J. C. R. Tseng, "An Incremental Mining Algorithm for Association Rules Based on Minimal Perfect Hashing and Pruning", LNCS, vol. 7234, **(2002)**, pp. 106-113.
[12] Y. Li, Z. H. Zhang, W. B. Chen and M. Fan, "TDUP: an approach to incremental mining of frequent itemsets with three-way-decision pattern updating", International Journal of Machine Learning and Cybernetics, vol. 8, **(2017)**, pp. 441-453.
[13] E. T. Wang and A. P. Chen, "A novel hash-based approach for mining frequent itemsets over data streams requiring less memory space", Data Mining and Knowledge Discovery, vol. 19, no. 1, **(2009)**, pp. 132-172.
[14] J. H. Chang and W. S. Lee, "Detecting Recently Frequent Itemsets adaptively over online transactional data streams", Information Systems, vol. 31, **(2006)**, pp. 849-869.
[15] J. Chang and W. Lee, "A Sliding Window Method for detecting Recently Frequent Itemsets over Online Data Streams", Journal of Information Science and Engineering, vol. 20, no. 4, **(2004)**, pp. 753-762.
[16] Y. Chi, H. Wang, P. S. Yu and R. R. Muntz, "Moment: Maintaining Closed Frequent Itemsets over a Stream Sliding Window", Proceeding of the Fourth IEEE International Conference on Data Mining (ICDM'4), **(2004)**, pp. 59-66.
[17] C. F. Ahmed, S. K. Tanbeer and B. S. Jeong, "Efficient Mining of Weighted Frequent Patterns Over Data Streams", 11th IEEE International Conference on High Performance Computing and Communications, **(2009)**, pp. 400-406.

[18] H. F. Li and S. Y. Lee, "Mining Frequent Itemset over Data Stream using Efficient Window Sliding Techniques", Expert Systems with Applications, vol. 36, **(2009)**, pp. 1466-1477.
[19] C. M. Lin, Y. L. Hsieh and K. C. Yin, "ADMiner:An Incremental Data Mining Approach Using a Compressed FP-tree", Journal of Software, vol. 8, no. 8, **(2013)**, pp. 2095-2103.

Authors

Jeonghee Hwang, she received Ph.D degrees from Chungbuk National University in 2005 in computer science. In 2006 joined the faculty of Namseoul University, where she is now an assistant professor. Her research interests include data mining, semantic web, ubiquitous computing and big data processing.

Hyeok Kim, he is undergraduate student, Konkuk University, Korea. His research interests include data mining and big data mining.

Jeonghee Chi, she received Ph.D degrees from Chungbuk National University in 2006 in computer science. In 2007 she joined the faculty of Konkuk University, where she is now an assistant professor. Her research interests include IoT, stream data mining, machine learning and big data analysis.

