Journal of Information Processing Systems, Vol.6, No.1, March 2010 DOI : 10.3745/JIPS.2010.6.1.079

Mining Frequent Itemsets with Normalized Weight in Continuous Data Streams

Younghee Kim*, Wonyoung Kim* and Ungmo Kim*

Abstract—A data stream is a massive unbounded sequence of data elements continuously generated at a rapid rate. The continuous characteristic of streaming data necessitates the use of algorithms that require only one scan over the stream for knowledge discovery. Data mining over data streams should support the flexible trade-off between processing time and mining accuracy. In many application areas, mining frequent itemsets has been suggested to find important frequent itemsets by considering the weight of itemsets. In this paper, we present an efficient algorithm WSFI (Weighted Support Frequent Itemsets)-Mine with normalized weight over data streams. Moreover, we propose a novel tree structure, called the Weighted Support FP-Tree (WSFP-Tree), that stores compressed crucial information about frequent itemsets. Empirical results show that our algorithm outperforms comparative algorithms under the windowed streaming model.

Keywords—Frequent Itemsets, Weighted Support, Window Sliding, Weighted Support FP-Tree, Data Stream, WSFI-Mine

1. INTRODUCTION

In recent years, advances in hardware technology have facilitated the ability to collect data continuously. For many recent applications, the concept of a data stream, possibly infinite, is more appropriate than a data set [1]. Previous studies have discussed data stream mining applications, such as manufacturing flow monitoring, sensor networks, stock exchange, telecommunications data flow and performance measurement in network monitoring and traffic management. Unlike data in traditional static databases, data streams are continuous, unbounded, and arrive at high speed. In many cases, these large volumes of data can be mined for interesting and relevant information in a wide variety of applications.

In data mining and knowledge discovery technique areas, frequent pattern mining plays an important role but it does not consider different weight value of the items. On the other hand, in real world applications, specific patterns and items within the patterns have more importance or priority than other patterns. For example, the support of a diamond ring is very low compared to the support of hairpins. Hence, some items vary in importance and therefore should be given certain priority. In biomedical data analysis, some genes are more significant than others in causing particular diseases, and some genes are more effective than others in fighting diseases.

※ This work was supported by the Korea Science and Engineering Foundation (KOSEF) grant funded by the Korea government (MEST) (No.2009-0075771)
Manuscript received January 28, 2010; accepted February 17, 2010.
Corresponding Author: Younghee Kim
* School of Information and Communication Engineering, Sungkyunkwan University, Suwon, Korea (younghees@gmail.com, bluemint88@gmail.com, umkim@ece.skku.ac.kr)

Weighted frequent pattern mining functions to retrieve this hidden knowledge from data sets.

In this paper, we consider the problem of mining with weighted support over a data stream sliding window using limited memory space, called WSFI-Mine (Weighted Support Frequent Itemsets Mine). The proposed algorithm allows the user to specify the weight for each item. It can discover useful recent knowledge from a data stream by using a single scan. Based on the weighted support, we propose a new algorithm, to efficiently discover all the frequent itemsets from streams. Our method is driven by an external weight table or weight function. The proposed WSFI-Mine method is designed to mine all frequent itemsets from one scan in the data streams.

The WSFI-Mine algorithm has three phases. First, a data stream is divided into patterns of three categories such as frequent items, latent items and infrequent items. Second, we present a novel tree structure, called the WSFP-Tree (Weighted Support FP-Tree), that stores compressed crucial information about frequent itemsets. The proposed WSFP-Tree structure is an extended FP-Tree. Finally, the WSFI-Mine method discovers frequent itemsets.

The remainder of this paper is organized as follows: In Section 2, we describe related work. In Section 3, we develop our proposed method for weighted support frequent itemsets mining over data streams. In Section 4, our experimental results are presented and analyzed. Finally, conclusions are given in Section 5.

2. Related Work

The problem of frequent itemsets mining is finding the complete set of itemsets satisfying a minimum support in the transaction database. Previous studies contributed to the efficient mining of frequent itemsets over data streams [2, 3, 4]. Li et al. proposed prefix tree-based single-pass algorithms, DSM-FI and DSM-MFI, to mine the set of all frequent itemsets and maximal frequent itemsets over the history of the data streams [5, 6]. Recently, a utility mining model was defined in [7]. Traditional association rules mining models assume that the utility of each item is always 1 and the sales quantity is either 0 or 1, thus it is only a special case of utility mining, where the utility or the sales quantity of each item could be any number. Chu et al. proposed THUI-Mine that can identify the temporal high utility itemsets by generating fewer temporal high transaction-weighted utilization 2-itemsets in data streams [8]. Giannella et al developed a FP-tree-based algorithm, FP-stream, to mine frequent itemsets at multiple time granularities by a novel titled-time windows technique [9]. Weighed frequent itemsets mining has been suggested to find important frequent itemsets by considering the weights of itemsets. Some weighted frequent pattern mining algorithms MINWAL [10], WARM [11], WAR [12] have been developed based on the Apriori algorithm [13]. The first FP-tree based weighted frequent pattern algorithms WFIM [14], WIP [15] show that the weighted support of an itemset does not have the property of downward closure. By using an efficient tree structure, Ahmed et al propose a sliding window based novel technique WFPMDS. It requires only a single-pass of data stream for tree construction and mining operations [16]. The existing algorithms cannot be applied for stream data mining because they require multiple scans. Moreover, they cannot extract the recent change of pattern in a data stream adaptively. This paper can mine dynamically using maintained usage patterns information from a previous sliding time.

3. Proposed Algorithm

In this section, we suggest frequent itemsets mining with normalized weight in continuous data streams. Firstly, we describe the definition of a set of terms that leads to the formal definition of the weighted support mining formula.

3.1 Preliminaries

Let $I=\{i_1, i_2, \ldots, i_m\}$be a set of items, a transaction $T= (tid, x_1x_2\ldots x_n)$, $x_i \in I$, for $1\leq i \leq n$, is a set of items, while n is called the size of the transaction, and each transaction has a unique transaction identifier TID, A transaction generated at the k^{th} turn is denoted by T_k and its transaction identifier TID is k. An itemset $X=\{x_1, x_2, \ldots, x_n\}$ is a set of items such that $X \in (2^I -\{\emptyset\})$ where 2^I is the power set of I. An itemset is a non-empty set of items. An itemset with size k is called a *k-itemset*. An itemset $=\{x_1, x_2, \ldots, x_n\}$ is also represented as $x_1, x_2, \ldots, x_n$. When a new transaction T_n is generated, where n is the latest incoming transaction T_n. i.e., $D = <T_1,T_2,\ldots, T_n\}$ and the total number of transactions in D_n is denoted by $|D|_n$. Table 1 shows an example with $\sum_I = \{a,b,c,d,e,f\}$. A window can be composed of a fixed number of non-overlapping transactions. In Table 1, we consider that one window contains four transactions, window size is $N = 4$.

Weight support of each item shows in Table 2. Each item is normalized as a weight value within a weight range, $ws_{min}(X) \leq w(X) \leq ws_{max}(X)$. The normalized minimum weighted support, $ws_{min}(X) = (support * w_{min}(X))$ is defined as the value of multiplying the support of an itemset with each minimum weight of itemsets. The normalized maximum weighted support, $ws_{max}(X) = (support * w_{max}(X))$ is defined as the value of multiplying the support of an itemset with each maximum weight of itemsets. In Table 2, $ws_{min}(X) \leq w(X) \leq ws_{max}(X)$ is $0.2 \leq w(X) \leq 0.8$.

A weighted support of an itemset is defined as the value that results from multiplying the itemsets support by the weight of the itemsets. The weighted support of the itemset, X, is given as follows:

$$X_{(support*weight)} = support(X) * weight(X) \tag{1}$$

An itemset X is called a weighted frequent itemset if the weighted support of the itemset is greater than or equal to the minimum threshold.

Definition 1 Weighted Support, $ws(X,W_t)$, the weight of item to reflect the importance of each item in the current sliding window Wt is defined as $X_{(support*weight)}$, where $w_{min}(X) \leq w(X) \leq w_{max}(X)$ is the weight range. For example, the weighted support of item "a" is $ws(\text{a}, W_1) = 2*0.3 - 0.6$, by Table 1 and Table 2.

Table 1. Transaction of data streams

window$_1$		*window*$_2$	
T_{id}	itemset	T_{id}	itemset
T_1	(a, c, d)	T_2	(b, c, e)
T_2	(b, c, e)	T_3	(a, b, c, e)
T_3	(a, b, c, e)	T_4	(b, e)
T_4	(b, e)	T_5	(a, b, c f)

Table 2. Weight values of each item ($0.2 \leq w(X) \leq 0.8$)

item	a	b	c	d	e	f
weight	0.3	0.6	0.2	0.8	0.4	0.2

Definition 2 Minimum Weighted Support, $ws_{min}(X)$, the value of multiplying the support of an itemset with each minimum weight of itemset is defined as *support* * $w_{max}(X)$. The minimum weighted support of item 'a', $ws_{min}(a)$, is $0.4=2\times0.2$.

Definition 3 Maximum Weighted Support, $ws_{max}(X)$, the value of multiplying the support of an itemset with each maximum weight of itemset is defined as *support* * $w_{max}(X)$. The maximum weighted support of item a, $ws_{max}(a)$, is $1.6=2\times0.8$.

Definition 4 Weighted Support Frequent Itemset, an itemset X is called a WSFI, if the $ws(X,W_t)$ is more than a minimum weighted support ($ws_{min}(X)$) and it is also less than a maximum weighted support ($ws_{max}(X)$).

Weighted frequent itemsets mining is attractive in that important patterns are discovered. Generally, the normalized weighted support of frequent itemsets is no less than the minimum support. Therefore, the two second factors φ and ε are each termed the user-defined minimum weighted support threshold and the minimum weighted support error threshold, respectively. So, the user-defined minimum weighted support threshold, φ, is given as follows:

$$\varphi = ((Max(ws_{min}(X)) + Min(ws_{max}(X))) / 2 \tag{2}$$

The user-defined minimum weighted support error threshold, ε, is given as follows:

$$\varepsilon = ((Min(ws_{min}(X)) + Min(ws_{max}(X))) / 2 \tag{3}$$

In our proposed method, the embedded itemsets in the data streams can be divided into three patterns: frequent, latent, and infrequent. An itemset X is a frequent itemset if $ws(X) \geq \varphi$, where ϕ is a user-defined minimum weighted support threshold. An itemset X is a latent itemset if $\varepsilon \leq ws(X) < \varphi$, where ε is the minimum weighted support error threshold in the range of $[0, \varphi]$. An itemset X is an infrequent itemset if $ws(X) < \varepsilon$. An itemset X is termed a maximal frequent itemset with weighted support if it is not a subset of any other frequent itemset.

3.2 Frequent Itemsets with Normalized weight

In this section, we illustrate how the weighted support frequent itemsets are defined by normalized weight. First, we show Table 3. Consider the first five transactions in a transaction data stream, T_1, T_2, T_3, T_4, and T_5, where a, b, c, d, e, and f are called items. Let the size of window w be 4 and a weight range is given as $0.2 \leq w(X) < 0.8$. Hence, the transaction data stream consists of two windows, $w_1 = T_1, T_2, T_3, T_4$ and $w_2 = T_2, T_3, T_4, T_5$. Each weight of itemset, $w(X)$ is a = 0.3, b=0.6, c=0.2, d=0.8, e=0.4 and f=0.2. Then, in window 1, the itemsets support are a=2, b=3, c=3, d=1, e=3 hence $ws(X)$ is a=0.6, b=1.8, c=0.6, d=0.8, e=1.2. In normalized weight range of

Table 3. The normalized weight values for each sliding window within a weight range

window$_1$				*window*$_2$			
item	$sup(X, w_1)$	$ws_{min}(X) \leq ws(X) \leq ws_{max}(X)$	$ws(X)$	item	$sup(X, w_2)$	$ws_{min}(X) \leq ws(X) \leq ws_{max}(X)$	$ws(X)$
a	[2:0.3]	$0.4 \leq ws(a) \leq 1.6$	0.6	a	[2:0.3]	$0.4 \leq ws(a) \leq 1.6$	0.6
b	[3:0.6]	$0.6 \leq ws(b) \leq 2.4$	1.8	b	[4:0.6]	$0.8 \leq ws(b) \leq 3.2$	2.4
c	[3:0.2]	$0.6 \leq ws(c) \leq 2.4$	0.6	c	[3:0.2]	$0.6 \leq ws(c) \leq 2.4$	0.6
d	[1:0.8]	$0.2 \leq ws(d) \leq 0.8$	0.8	d	[0:0.8]	$0 \leq ws(d) \leq 0$	0
e	[3:0.4]	$0.6 \leq ws(e) \leq 2.4$	1.2	e	[3:0.4]	$0.6 \leq ws(e) \leq 2.4$	1.2
				f	[1:0.2]	$0.2 \leq ws(f) \leq 0.8$	0.2

itemset, ws_{min} is 0.2 and ws_{max} is 0.8. Therefore, ws_{min} is a=0.4=(2*0.2), b=0.6=(3*0.2), c=0.6=(3*0.2), d=0.2=(1*0.2), e=0.6=(3*0.2). The resulting maximum $Max\ (ws_{min}(X))$ is 0.6 and $ws_{max}(X)$ is a=1.6= (2*0.8), b=2.4=(3*0.8), c=2.4=(3*0.8), d=0.8=(1*0.8), e=2.4= (3*0.8). The resulting minimum of X, $Min\ (ws_{max}(X))$ is 0.8. By observation, we obtain that $Max\ (ws_{min}(X)) = 0.6$, $Min\ (ws_{max}(X)) = 0.8$ and $Min(ws_{min}(X)) = 0.2$.

From equations (2) and (3), we conclude that $\not\phi$ is 0.7=(0.6+0.8)/2 and ε is 0.5=(0.2+0.8)/2. As a result, an itemset X is a frequent pattern if $ws(X) \geq 0.7$, where b, d, and e. The itemsets a and c represent a latent pattern, with $0.5 \leq w(X) < 0.7$. Itemset X is pruned as an infrequent pattern, $ws(X) < 0.5$.

3.3 WSFI-Mine Method

With streaming databases, memory is often limited. It is hard to store large itemsets in memory. In this section, we propose a WSFI-Mine that can mine dynamically maintained usage patterns using information from a previous sliding time that can be updated in real time. The WSFI-Mine algorithm has three phases: the normalization of weight support and dividing patterns into three categories, the construction of the WSFP-Tree, and a frequent itemset discovery scheme. Construction of a WSFP-Tree ensures that frequent pattern mining can be performed efficiently. A WSFP-Tree is a data structure based on an extended FP-tree. It serves to store compressed crucial information about frequent patterns. WSFP-Tree construction is described as follows: We scan the stream database only once, counting the support for each item and checking the weight of each item. Then, the product of multiplying the item support by the weight of each item must be sorted in descending order. The window slide has a fixed number of transactions, w. T_1, T_2, T_3, T_4, T_5, and T_6 are transactions; a, b, c, d, e, f, and g are items. Let the size of the sliding window w be 4: the transaction data stream consists of three sliding windows, w_1, w_2 and w_3. As shown in Figure 1, the weighted support of items of each window in the sliding phase is shown in Table 4. For example, consider the weight support of each item in the windowsliding phase, and let $\not\phi$ and ε, be 0.7 and 0.5 respectively. Notice Table 4. First, the first sliding window w1 consists of four transaction data streams: [T_1, <acd >], [T_2, <bce>], [T_3, <abce>], and [T_4, <be>]. Here, item ‘a’ appears in T_1 and T_3 of window slide w_1: the weight of ‘a’ is 0.3. Hence, $ws(a)$ is 0.6, the weighted support of ‘a’. Similarly, $ws(b) = 1.8$, $ws(c) = 0.6$, and $ws(e) = 1.2$. Next, we sort the items by weight support in descending order. The result is a descending ordered list <bedac> in sliding window w_1. Second, w_2 generates a descending list <beacf>, in the sliding window w_2, after T_1 is removed from w_1: T_5 is appended to w_2.

Next, the new items f and g appear in the third sliding window w_3 from the data stream; item

Table 4. Support and weighted support in each window sliding phase

Item	*window*$_1$		*window*$_2$		*window*$_3$	
	w_1		w_1-T_1+T_5		w_1-T_2+T_6	
	<b e d a c>		<b e a c f>		<b a g d e c>	
	sup(*x*)	*ws*(*x*)	*sup*(*x*)	*ws*(*x*)	*sup*(*x*)	*ws*(*x*)
[a:0.3]	2	0.6	2	0.6	3	0.9
[b:0.6]	3	1.8	4	2.4	3	1.8
[c:0.2]	3	0.6	3	0.6	3	0.6
[d:0.8]	1	0.8	0	0.0	1	0.8
[e:0.4]	3	1.2	3	1.2	2	0.8
[f:0.2]			1	0.2	1	0.2
[g:0.9]					1	0.9

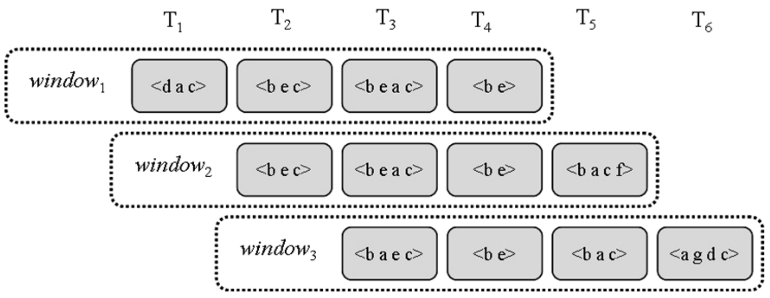

Fig. 1. The itemsets list after descending order

support is f: 0.2 and g: 0.9, where w_3 is carried from w_2 - T_2+ T_6, then, w_3 generates a descending list <bagdec>. Next, the weight support of item f, $ws(f) = 0.2$ is deleted based on the pruning conditions, as the infrequent pattern is less than the minimum weighted support error threshold ε, i.e., $ws(f) < 0.5$.

3.4 WSFP-Tree Construction

In the previous observations, the construction of a WSFP-Tree is described as follows: First, we determine the descending list by reading the data stream in w_1. The frequent items in each transaction data stream are listed accordingly as in Figure 1. The descending list items are stored in descending order of weight support in the header table. The header table of the WSFP-Tree has three columns, the item-id, weight support of each item, and node link. The FP-Tree proposed by Han et al [17] stores the item id in ascending order; while in our WSFP-Tree, the item ids are mapped to a descending order list. Each node of the WSFP-Tree contains an array of counts for items with weight. The following is a more detailed analysis of this aspect. We use a header table to store all the descending list items by weight support. To build a WSFP-Tree, first we create a root node as "null". Next, by scanning the descending ordered transaction data streams, the WSFP-Tree is constructed as follows: Let the descending ordered list of weighted support items in sliding window w_1 be <bedac> and the minimum weighted support error threshold be 0.5 (i.e., $\varepsilon = 0.5$). Thus, after a scan of the first sliding window leads to the construction of Figure 2, each node in the WSFP-Tree has four fields: item name, count, weight,

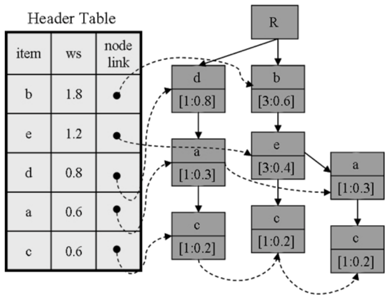

Fig. 2. WSFP-Tree after inserting first sliding window w_1

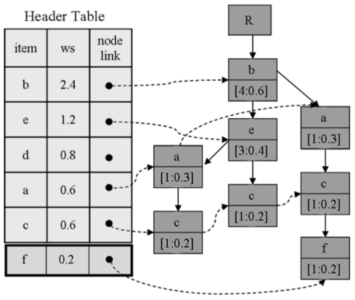

Fig. 3. WSFP-Tree after deleting T_1 and inserting T_5 in the second sliding window w_2

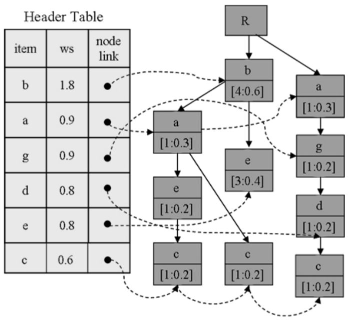

Fig. 4. WSFP-tree depicting pruning item f after and deleting T_2 inserting T_6 in the third sliding window w_3

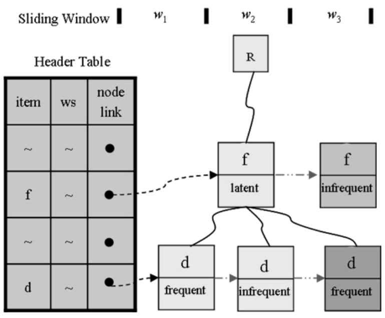

Fig. 5. WSFP-tree by inserting and deleting In window

and node-link. In Figure 3, for the second sliding window, w_2, a new transaction T_5 is added to the window stream. Then, item d in T_1 does not appear in the WSFP-Tree and a new item f will be updated in the header table and added to the WSFP-Tree. Next, the scan of the third sliding window w_3 leads to the construction of Figure 4. It depicts the pruning of infrequent item f after deleting T_2 and inserting T_6. Data changes usually with time in the streams. A currently infrequent pattern may become frequent in the future.

Hence, as shown Figure 5, we have to be careful not to prune infrequent itemsets too early. (i.e., item f in w_2 (latent pattern) → item f in w_3 (infrequent pattern). The algorithm to construct the WSFP-Tree is as follows. This structure has several advantages. First, we will not miss itemsets even if they were infrequent items in the previous sliding window. (e.g., item d). Second, we will save memory. It does not need to have the header table for itemsets with less than the weighted support. (e.g., item f).

3.5 WSFI-Mine Algorithm

The algorithm for the WSFI-Mine is described as follows: Initially, the WSFI-Mine reads a stream database TDS from the current window. Then, the WSFI-Mine processes the weight of each item and sorts the weighted support itemsets in each sliding window intodescending order.

```
Input:
    (1) A Stream Database (TDS)
    (2) Normalized minimum weighted support ( $\varphi$ )
    (3) Normalized minimum weighted error support ($\varepsilon$)
    (4) Weighted Range
Output:
    WSFP-Tree, A set of weighted support frequent itemsets.
Procedure:
    1. Scan a Stream Database and count support for each item.
    2. Multiply item support by the weight of each item.
    3. Sort them into a descending order list in a sliding window.
    4. Create the root of a WSFP-Tree. Next, each transaction performs as follows:
      4-1. Select the descending order frequent item and call:
           insert_wsfp_tree (dsitem_list, T).
      4-2. The function insert_wsfp_tree (dsitem_list, T) is performed as follows:
           (1) If T has a child node such that node.item =dsitem_list.item then increment the
                 node's count by 1 or create a new node with its count initialized to 1.
           (2) Link its parent to T and link its node-link to the nodes with the same item name via
               the node-link.
           (3) If $\varepsilon \le$ weighted support of node.item $\le \varphi$ then do not remove it from WSFP-Tree
               (latent pattern) or
               If the weighted support of node.item $\le \varepsilon$ then remove it from the WSFP-Tree in the
               next phase (infrequent pattern)
    5. The construction process of WSFP-Tree with respect to previous sliding window tree re-
       sult is the same as in step 4 recursively.
    6. End.
```

Fig. 6. Pseudo code for the WSFI-Mine

Next, a WSFP-Tree is constructed from the descending order list. Finally, frequent itemsets mining is usually performed.

4. EXPERIMENTAL RESULTS

4.1 Performance Comparison

We evaluate the performance of our WSFI-Mine algorithm by varying the usage of the memory space. We also analyze the execution time. The simulation is performed in Visual C++ and conducted in a machine with a 3GHz CPU and 1GB memory. We use two sets of synthetic databases from an IBM Quest data generator. Table 5 illustrates some of the parameters that we have controlled: the size of the sliding window 10K, the average size of the transaction T, the average size of the frequent itemsets I and we randomly generate the weight of each item in transaction, ranging from 0.2 to 0.8. Synthetic dataset T10I4D100K denotes the average size of the transactions and I the average number of frequent itemsets. A Mushroom database has been used extensively in the AI area.

Table 5. DATA SET

Database	**Parameter**			
	No. of items	***Average length***	***No. Records***	***Window size***
T10I4D100K	1,000	10	100,000	10K
Mushroom	120	23	8124	10K

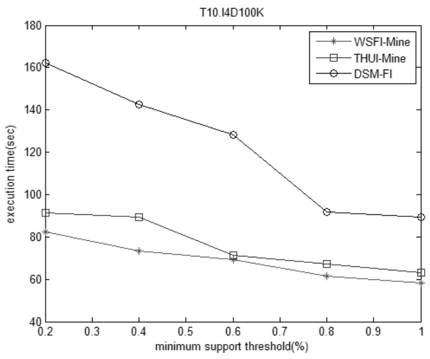

Fig. 7. Execution time on T10I4D100K

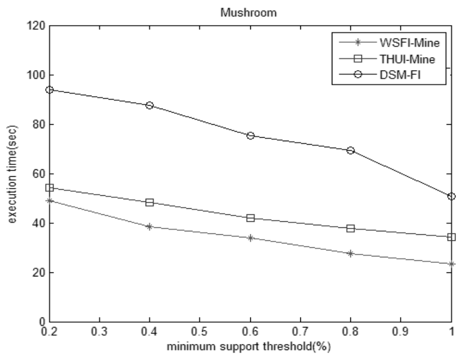

Fig. 8. Execution time on Mushroom dataset

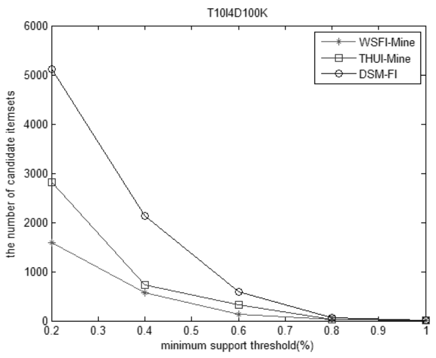

Fig. 9. The number of candidate itemsets on T10I4D100K

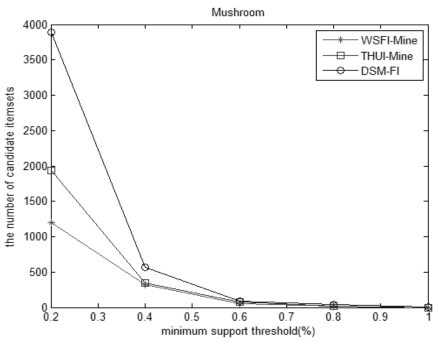

Fig. 10. The number of candidate itemsets on Mushroom

In this experiment, we compare only the relative performance of the THUI-Mine [8], DSM-FI [5] and WSFI-Mine. Figures 7 and 8 show the execution times for the three algorithms on dataset T10I4D100K and the Mushroom database, respectively, as the minimum support threshold is increased from 0.2% to 1%. As shown in figure 7 and figure 8, our algorithm leads to prominent performance improvement under different sizes of transactions. As shown in figures 9 and 10, when the minimum support threshold is decreased from 1% to 0.2%, our algorithm WSFI-Mine always generates far fewer candidates compared to the THUI-Mine and DSM-FI for various kinds of databases. Thus, we can conclude that the execution time is proportional to the number of candidates generated during the mining process.

4.2 Scalability

To test the scalability with the number of transactions, the T10.I4.DxxxK datasets were used. The WSFI-Mine is compared with the THUI-Mine. In Figure 11, both the WSFI-Mine and THUI-Mine show linear scalability with the number of transactions from 100K to 1000K. We tested different minimum support 0.2% and weight support 0.2 to 0.8. In Figure 11, we can see that the WSFI-Mine has good scalability in terms of number of transactions and becomes better

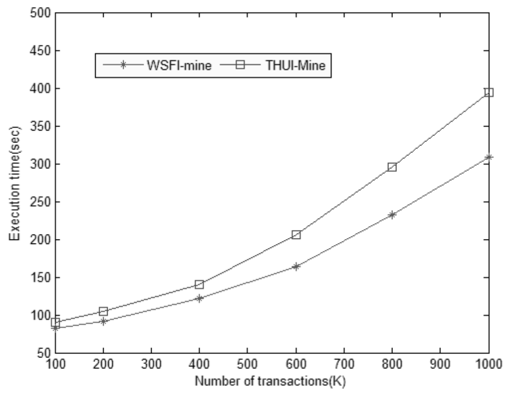

Fig. 11. Scalability in WSFI (T10.I4.DxxxK, min_sup = 0.2%, size: D100K~D1000K)

as the minimum support is increased. In comparison with other algorithms, the WSFI-Mine not only runs faster, it also has much better scalability in terms of base size.

5. CONCLUSION

This paper proposed a novel method to mine frequent itemsets from streams of datasets efficiently and effectively in memory and runtime. The proposed WSFI-Mine algorithm can mine all frequent itemsets in one scan from the data stream. The WSFI-Mine's contribution is to effectively execute frequents by generating constraint candidate itemsets. The proposed WSFP-Tree is an extended FP-tree based data structure. It is an extended prefix-tree structure to store compressed, crucial information about frequent patterns. The evaluation demonstrates that the WSFI-mine outperforms the THUI-Mine and DSM-FI in mining frequent itemsets over data streams.

REFERENCE

[1] M.M. Gaber, et al, "Mining data streams: a review", ACM SIGMOD record 34(2), pp.18-26, 2005.
[2] J. Chang, W. Lee, "A Sliding Window Method for Finding Recently Frequent Itemsets over Online Data Streams", Journal of Information Science and Engineering, Vol.20, No.4, July, 2004.
[3] G.S. Manku, R. Motwani, "Approximate Frequency Counts Over Data Streams", In Proceedings of the 28th International Conference on Very Large Data Bases, pp.346-357, 2002.
[4] C.H. Lee, C.R. Lin, M.S. Chen, "Sliding window filtering: An efficient method for incremental mining on a time-variant database", Information Systems, 30, pp.227-244, 2005.
[5] H.F Li, S.Y. Lee, M.K. Shan, "An Efficient Algorithm for Mining Frequent Itemsets over the Entire History of Data Streams", In Proceedings of First International Workshop on Knowledge Discovery in Data Streams 9IWKDDS, 2004.
[6] H.F Li, S.Y. Lee, M.K. Shan, "Online Mining (Recently) Maximal Frequent Itemsets over Data Streams", In Proceedings of the 15th IEEE International Workshop on Research Issues on Data Engineering (RIDE), 2005.

[7] H. Yao, H.J. Hamilton, C.J. Butz, "A Foundational Approach to Mining Itemset Utilities from Databases", In Proceedings of the 4th SIAM International Conference on Data Mining, Florida, USA, 2004.
[8] C.J Chu, V.S. Tseng, T. Liang, "An efficient algorithm for mining temporal high utility itemsets from data streams", The Journal of System and Software 81, pp.1105-1117, 2008.
[9] C. Giannella, J, Han, J. Pei, X. Yan, P.S. Yu, "Mining Frequent Patterns in Data Streams at Multiple Time Granularities", Next Generation Data Mining, 2003.
[10] C.H. Cai, A.W. Fu, C.H. Cheng, W.W. Kwong, "Mining association rules with weighted items", In Proceedings of the International Database Engineering and Applications Symposium, IDEAS98, pp.68-77, Cardiff, Wales, UK, 1998.
[11] F. Tao, "Weighted association rule mining using weighted support and significant framework", In Proceedings of the 9th ACM SIGKDD, Knowledge Discovery and Data Mining, pp.661-666, 2003.
[12] W. Wang, J. Yang, P.S, Yu, "WAR: weighted association rules for item intensities", Knowledge Information and Systems, Vol.6, pp.203-229, 2004.
[13] R. Agrawal, R. Srikant, "Fast Algorithms for Mining Association Rules", In Proceedings of the 20th VLDB conference, pp.487-499, 1994.
[14] U. Yun, J.J. Leggett, "WFIM: weighted frequent itemset mining with a weight range and a minimum weight", In Proceedings of the 15th SIAM International Conference on Data Mining (SDM'05), pp.636-640, 2005.
[15] U. Yun, "Efficient Mining of weighted interesting patterns with a strong weight and/or support affinity", Information Sciences, Vol.177, pp.3477-3499, 2007.
[16] C.F. Ahmed, S.K. Tanbeer, B.S. Jeong, "Efficient Mining of Weighted Frequent Patterns Over Data Streams", 2009 11th International Conference on High Performance Computing and Communications, pp.400-406, June, Seoul, Korea, 2009.
[17] J. Han, J. Pei, Y. Yin, R. Mao, "Mining Frequents without Candidate Generation: A Frequent-Pattern Tree Approach", Data Mining and Knowledge Discovery, No.8, pp.53-87, 2004.

Younghee Kim

She received a Ph.D. degree in Information Engineering from Sungkyunkwan University, Korea, in 2010. She received BS and MS degrees in Computer Science from Soonchunhyang University, in 1993 and 1998, respectively. Currently, she is a post-doctorate at the Sungkyunkwan University Department of Computer Engineering. Her research interests include Data Mining, Data Stream Analysis and Knowledge-based Systems and RFID.

Wonyoung Kim

He received his B.S. degree in Information and Communication Engineering from the Sungkyunkwan University in 2009. He is currently a master student in Computer Engineering at Sungkyunkwan University. His research interests include data mining, opinion mining, and web mining.

Ungmo Kim

He received a BS degree in Mathematics from Sungkyunkwan University, Korea, in 1981 and an MS degree in Computer Science from Old Dominion University, U.S.A. in 1986. His Ph.D. degree was received in Computer Science from Northwestern University, U.S.A., in 1990. Currently he is a full professor of School of Information and Communication Engineering, Sungkyunkwan University, Korea. His research interests include Data Mining, Database Security, Data Warehousing, and GIS.