An Efficient Subset-Lattice Algorithm for Mining Closed Frequent Itemsets in Data Streams

Ye-In Chang, Chia-En Li and Wei-Hau Peng
Department of Computer Science and Engineering
National Sun Yat-Sen University
Kaohsiung, Taiwan, Republic of China
changyi@cse.nsysu.edu.tw

Abstract—**There are many applications of using association rules in data streams, such as market analysis, network security, sensor networks and web tracking. Mining closed frequent itemsets is a further work of mining association rules, which aims to find the subsets of frequent itemsets that could extract all frequent itemsets. Formally, a closed frequent itemset is a frequent itemset which has no superset with the same support as it. One of well-known algorithms for mining closed frequent itemsets based on the sliding window model is the NewMoment algorithm. However, the NewMoment algorithm could not efficiently mine closed frequent itemsets in data streams, since they will generate closed frequent itemsets and many unclosed frequent itemsets. Moreover, when data in the sliding window is incrementally updated, the NewMoment algorithm needs to reconstruct the whole tree structure. Therefore, we propose the Subset-Lattice algorithm which embeds the property of subsets into the lattice structure to efficiently mine closed frequent itemsets over a data stream sliding window. Moreover, when data in the sliding window is incrementally updated, our Subset-Lattice algorithm will not reconstruct the whole lattice structure.**

Keywords-**association rules, closed frequent itemsets, data streams, frequent itemsets, sliding window**

I. INTRODUCTION

Mining association rules is a process of nontrivial extraction of implicit, previously and potentially useful information from data in databases. The most important task is to discover association rules which generates new techniques and tools.

In recent years, database and knowledge discovery communities have focused on a new data model, in which data arrives in the form of continuous streams. It is called the data stream or streaming data. Many real-world applications data is better appropriately handled by the data stream model than by traditional static databases. Therefore, mining frequent itemsets in data streams has become a research area with increasing importance [2, 3, 5, 6, 8, 9, 10, 11].

The main goal of mining association rules is to find the *frequent itemsets*, where a frequent itemset is a combination of items whose appearing times in the datasets is greater than a given threshold. In traditional algorithms of mining association rules, *e.g.*, Apriori [1], all frequent itemsets will be kept. Although the information is complete, there are many redundancies among the mining result. Take Transaction Database T in Figure 1 as an example, Figure 2 shows the frequent itemsets of Transaction Database T, where the value of the minimal support is 2. By the characteristic of a frequent itemset — all subsets of a frequent itemset are also frequent itemsets, frequent itemsets "ab", "ad", and "bd" seem very obvious and are redundant. For the reasons mentioned above, the concept of maximal frequent itemsets is proposed to reduce the redundancy. A frequent itemset is maximal frequent itemset if none of its proper supersets is frequent. For the example shown in Figure 1, "abd" is the maximal frequent itemset.

TID	Items
1	a, b
2	a, b, d
3	c, d
4	a, b, d
5	a, b, d

Figure 1. Transaction Database T

L1	a :4 b :4 d :4
L2	ab :4 ad :3 bd :3
L3	abd :3

Figure 2. Frequent itemsets of Transaction Database T with the minimal support = 2

Although maximal frequent itemsets eliminate all redundancies, they lose information. The support counts of maximal frequent itemsets are known, but the support counts of their subsets are all lost. To overcome the disadvantage of maximal frequent itemsets, a new kind of frequent itemset, called *closed frequent itemsets*, is proposed. A *closed frequent itemset* is a frequent itemset which has no superset with the same support as it. For the example shown in Figure

978-0-7695-4919-4/12 $26.00 © 2012 IEEE
DOI 10.1109/TAAI.2012.12

L1	d :4
L2	ab :4
L3	abd :3

Figure 3. Closed large itemsets of Transaction Database T with the minimal support = 2

1, Figure 3 shows the closed frequent itemsets. Frequent itemset "ab" is a closed large itemset, because all other frequent itemsets ("abd") that contain it do not have the same support count with it. However, frequent itemset "bd" is not a closed frequent itemset, since frequent itemset "abd" contains "bd" and has the same support as "bd". Closed frequent itemsets can greatly reduce the number of large itemsets, and do not lose any information.

The Moment algorithm [4] has been proposed for mining closed frequent itemsets over a data stream sliding window. (Note that the sliding window model emphasizes the recent data.) It defines four types of conditions to each candidate node. However, it stores information including the current closed frequent itemsets and many unclosed frequent itemsets, which consumes much memory, especially, when the support threshold is low. Moreover, the exploration and node type checking are time consuming.

The NewMoment algorithm [7] also reserves the hash table used in the Moment algorithm to check whether each candidate node is a closed frequent itemset or not. Although the NewMoment algorithm resolves the disadvantage of the time and memory space consuming of the Moment algorithm by using the technique of bit-patterns to speed up the tree construction, it still has to scan the transactions in the sliding window to generate the bit-sequence of each item and generate unclosed frequent itemsets in the search structure. Morever, the NewMoment algorithm needs to reconstruct the whole tree structure, when the window slides.

Therefore, in this paper, we propose the Subset-Lattice algorithm, which will not produce unclosed nodes into the lattice structure, to efficiently mine closed frequent itemsets. We use properties of sets to compute the closed frequent itemsets directly among transactions. When the window is sliding, there will not be too much fluctuation in the Subset-Lattice structure.

The rest of the paper is organized as follows. Section 2 gives a survey of some algorithms. Section 3 presents the proposed Subset-Lattice algorithm. In section 4, we study the performance. Finally, we give a conclusion.

II. Related Work

In this section, we describe two well-known algorithms, the Moment algorithm [4], and the NewMoment algorithm [7] for mining closed frequent itemsets in data streams.

A. The Moment Algorithm

The Moment algorithm proposed an in-memory data structure, the *closed enumeration tree* (CET) to maintain a dynamically selected small set of itemsets. Similar to a prefix tree, each node n_i represents an itemset I. A child node, n_j , is obtained by adding a new item to I. The CET only maintains a dynamically selected set of itemsets, which include (1) closed frequent itemsets, and (2) itemsets that form a boundary between closed frequent itemsets and the rest of the itemsets.

The Moment algorithm divides itemsets on the boundary into two categories. First, it corresponds to the boundary between frequent and non-frequent itemsets. Second, it corresponds to the boundary between closed and non-closed itemsets.

B. The NewMoment Algorithm

In this subsection, we describe the NewMoment algorithm. A bit vector based representation of items is used in the NewMoment algorithm to reduce the time and memory needed to slide the windows. A new data structure NewCET (New Closed Enumeration Tree) based on a prefix tree structure is developed to maintain the essential information of closed frequent itemsets in the recent transaction of a data stream.

In the NewMoment algorithm, the *bit-sequence* has three kinds of effect: (1) representation of items, (2) window sliding and (3) counting support. In the representation phase, for each item *X* in the current sliding window, a *bit-sequence* with *w* bits, is constructed. If an item *X* is in the *i*th transaction of the current window, the bit of **Bit**(*X*) is set to be 1; otherwise, it is set to be 0. In the window sliding phase, the process consists of two steps: delete the oldest transaction and append the incoming transaction. The *bit-sequence* of items is used to left-shift one bit to delete the oldest transaction. The concept of *bit-sequence* can be used to count support. For example, in Figure 4, the *bit-sequence* of 2-itemset "ab", Bit(ab), is 1010. That means transactions T_1 and T_3 contain itemset "ab".

The NewCET data structure consists of three parts. (1) The *bit-sequences* of all 1-itemsets in the current transaction-sensitive window *TransSW*. (2) A set of closed frequent itemsets in *TransSW*. (3) A hash table for checking whether a frequent itemset is closed or not, and storing all closed frequent itemsets with their supports as keys, which is similar to the one used in the Moment algorithm [4].

III. The Subset-Lattice Algorithm

One of the well-known window models for data streams is the sliding window model. The sliding window model emphasizes the recent data, which in the majority of real-world applications is more important and relevant than old data. In this section, we present our Subset-Lattice algorithm. We first define some basic definitions and notations and then discribe how to use these techniques to find and represent closed frequent itemsets and organize the closed frequent itemsets in a lattice.

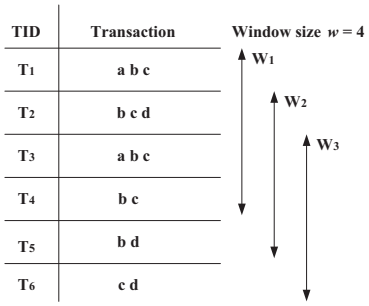

Figure 4. Example of Transaction-Sensitive Window

Table I
VARIABLES

Notation	Description
w	The sliding window size
Tid	The transaction ID of data streams
$NewT$	A transaction which is going to be inserted into the Subset-Lattice
$OldT$	A transaction which is already in the Subset-Lattice
$InterX$	A common subset of transactions $NewT$ and $OldT$
$ODes$	A descendant of transaction $OldT$
$ChildT$	A child lattice of transaction $OldT$
$InterY$	A common subset of transactions $InterX$ and $ChildT$
$DelT$	A transaction which is going to be deleted from the Subset-Lattice
$DesT$	A child lattice of transaction $DelT$
$ParT$	The parent lattice of transaction $DelT$
$Tidset$	A set of Tid

In the Subset-Lattice algorithm, we first transfer the transaction into bit-representation. Take transaction {CD} as an example. We use the maximum length of itemset as the length of bit-length and itemsets are ordered as lexical order. When the item appears in the transaction, we set the bit to 1 according to the lexical order position. The bit-represent of {CD} is denoted as $Bit(0011)$, since there are four items {ABCD}. By way of the sliding window approach, we maintain the essential information of closed frequent itemsets in the recent w transactions of a data stream. The sliding process consists of two steps: delete the oldest transaction and append the incoming transaction. Second, we use set-relation checking to deal with each transaction. The variables used in our algorithm are listed in Table I.

The proposed data structure, called Subset-Lattice, is an extended Lattice structure. The Subset-Lattice consists of two parts.

1) A lattice node which contains the information of transaction itemset and transaction ID.
2) A pointer which represents the relation between the superset and the subset. In the Subset-Lattice, a pointer is pointed to the subset from the superset.

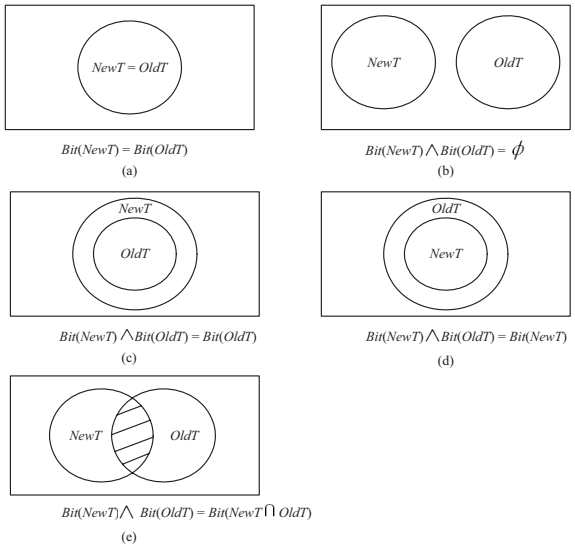

Figure 5. Conditions of subsumption checking: (a) Case 1; (b) Case 2; (c) Case 3; (d) Case 4; (e) Case 5.

A. Data Insertion

In this step, we focus on recent transactions in data streams. The proposed algorithm is processed as follows. First, we insert the transaction into the Subset-Lattice including the transaction itemset X and transaction identifier Tid. When the next transaction comes, we start to process the subsumption checking. In the subsumption checking step, conditions are considered based on the relation between the new transaction $NewT$ and the old transaction $OldT$. We use logical AND and XOR operators to make subsumption checking more efficient.

In Figure 5, Case 1 indicates the bit-operation $Bit(NewT) \otimes Bit(OldT) = \emptyset$; that is, itemset $NewT$ and itemset $OldT$ are the same itemset. Case 2 indicates the bit-operation $Bit(NewT) \wedge Bit(OldT) = \emptyset$; that is, there is no relation between $NewT$ and $OldT$. Case 3 indicates the bit-operation $Bit(NewT) \wedge Bit(OldT) = Bit(OldT)$; that is, itemset $NewT$ is the superset of itemset $OldT$. Case 4 indicates the bit-operation $Bit(NewT) \wedge Bit(OldT) = Bit(NewT)$; that is, itemset $OldT$ is the superset of itemset $NewT$. Case 5 indicates the bit-operation $Bit(NewT) \wedge Bit(OldT) = Bit(NewT \cap OldT)$; that is, itemset $NewT$ and itemset $OldT$ have some common items.

We use an example to illustrate the data appending procedure. Figure 6 shows an example of stream data and window size = 5. When the data stream comes, we process procedure $Insert$. Transaction Tid_1 is the first transaction of the data stream, The itemset {CD} and $Tid(1)$ are inserted into the Subset-Lattice directly. Transaction Tid_2 will call function $CheckSet$ to check the relation among transaction Tid_2 and previous transactions. Transaction Tid_2 will process Case 2, because {AB} ∩ {CD} is empty. Thus, a new node for

Tid	Item	Bit-represent
1	CD	0011
2	AB	1100
3	AB	1100
4	ABCD	1111
5	ABD	1101
6	AC	1010

Figure 6. An example of the data stream

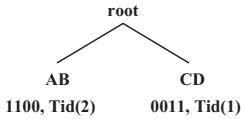

Figure 7. Subset-Lattice of transaction Tid(1) and transaction Tid(2)

itemset {AB} is created and transaction $Tid(2)$ is inserted into the Subset-Lattice. Figure 7 shows the result of inserting transactions Tid_1 and Tid_2.

Transaction Tid_3 will call function $FindT$ to search the Subset-Lattice, to check whether it can find the itemset in the Subset-Lattice or not. In function $FindT$, we will process Case 1. By way of function $FindT$, we can find itemset {AB}, which is already in the Subset-Lattice. Hence, we only update $Tid(3)$ into itemset {AB}'s $Tidset$ as shown in Figure 8. Transaction Tid_4 is the condition of Case 3. Itemset {ABCD} is the superset of itemset {AB} and itemset {CD}. Therefore, we will process function $CheckSet$. Itemsets {AB} and {CD} become itemset {ABCD}'s child. We then insert transaction $Tid(4)$ into the $Tidset$ of itemsets {AB} and {CD} as shown in Figure 9.

Transaction Tid_5 will pass through conditions of Case 4 and Case 5. First, itemset {ABD} encounters itemset {ABCD} and by way of function $CheckSet$, itemset {ABD} becomes a child of itemset {ABCD} and a superset of itemset {AB}(Case 4). We need to insert $Tidset$ of

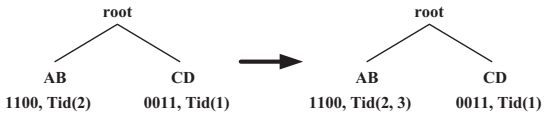

Figure 8. Subset-Lattice of transaction Tid(3)

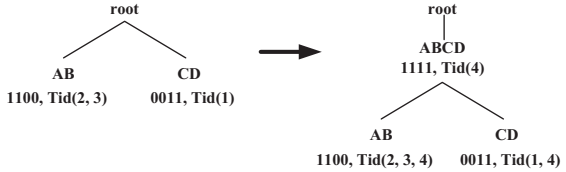

Figure 9. Subset-Lattice of transaction Tid(4)

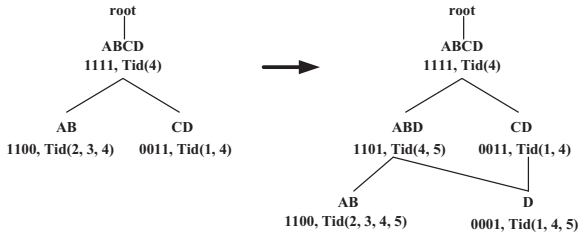

Figure 10. Subset-Lattice of transaction Tid(5)

itemset {ABCD} into itemset {ABD} and insert $Tidset$ of itemset {ABCD} into itemset {AB}. Second, through function $CheckSet$, itemset {ABD} and itemset {CD} have a common itemset {D}(Case 5). We create a new lattice node for itemset {D}, then insert the $Tidset$ of itemset {ABD} and itemset {CD} into the $Tidset$ of itemset {D} as shown in Figure 10.

B. Data Deletion

In this step, we describe how to delete the oldest transaction from the Subset-Lattice. When a transaction is out of the current window, it should be deleted from the Subset-Lattice. In the Subset-Lattice, We also need to traverse the nodes which are relevant to the deleted transaction, and update their $Tidset$. If a node is deleted from the Subset-Lattice, a subset of the deleted node will be influenced. Because the subset lattice node could be created by two itemsets. For example, itemset {D} is created by itemset {ABD} and itemset {CD}. Both itemset {ABD} and itemset {CD} contain itemset {D}; therefore, the support of itemset {D} is larger than that of its superset itemset {ABD} and itemset {CD}. In this case, any itemset of these two itemsets is deleted, and the subset lattice node will be deleted in the meantime.

Based on the properties of closed frequent itemsets, any lattice node in the Subset-Lattice is a closed frequent itemset. We use an example shown in Figure 11 to describe the procedure of deletion. In this example, itemset {CD} is the oldest transaction in the current window. According to the window table, where a window table is the tilted-time window structure inserted into each node, we can directly find the location of itemset {CD}. First, we remove transaction $Tid(1)$ from $Tidset$ of itemset {CD} and all of its subsets recursively as shown in Figure 11-(b). Second, we compare $Tidset$ of itemset {D} with that of its superset {ABD} and superset {CD}. There is a critical key point in this step, if the $Tidset$ of a subset is equal to its superset, then the subset and superset must occur in the same transaction; that is, the subset becomes a unclosed frequent itemset. Therefore, itemset {D} should be deleted from the Subset-Lattice. The itemset {CD} should be deleted in the same way as shown in Figure 11-(c) and Figure 11-(d).

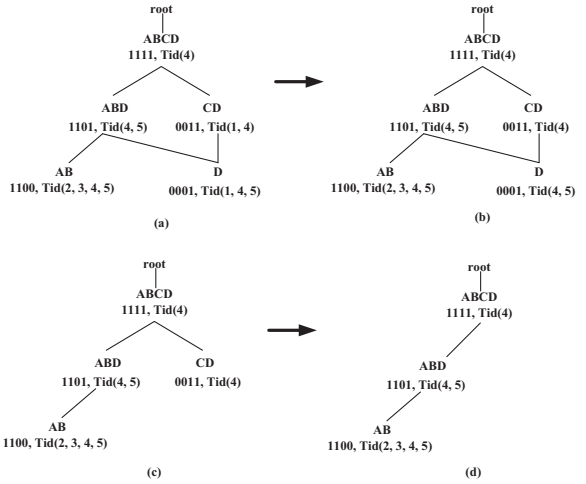

Figure 11. Subset-Lattice after deleting itemset {CD}: (a) the original lattice; (b) updating the Tidset of itemset {CD} and itemset {D}; (c) deleting itemset {D}; (d) deleting itemset {CD}.

Table II
PARAMETERS USED IN THE EXPERIMENT

Parameters	Meaning
$\|T\|$	The average size of transactions
$\|MT\|$	The maximum size of transactions
$\|I\|$	The average size of maximal potentially frequent itemsets
$\|D\|$	The number of transactions
$\|MI\|$	The maximum size of potentially frequent itemsets
$corr$	The number of the correlation level
$\|L\|$	The number of the maximal potentially frequent itemsets
$\|N\|$	The number of items
$\|SW\|$	The size of the sliding window
$\|S\|$	The minimum support

IV. PERFORMANCE

In this section, first, we show how to generate the synthetic data which will be used in the simulation. Second, we study the performance of the NewMoment algorithm and our proposal Subset-Lattice algorithm.

A. The Simulation Model

In this subsection, we describe the way to generate synthetic transaction data from the IBM synthetic data developed by Agrawal *et al*. [1]. These synthetic transaction data sets are used to evaluate the performance of algorithms for mining closed frequent itemsets. The parameters used in the generation of the synthetic data are shown in Table II.

B. Experimental Results

In this subsection, we show the experiment results. All experiments are done on a 2.4 GHz Intel Core2 Quad PC with 2GB memory and running with Windows XP system. The proposed Subset-Lattice algorithm was implemented in Java and compiled by JDK 1.5.0.

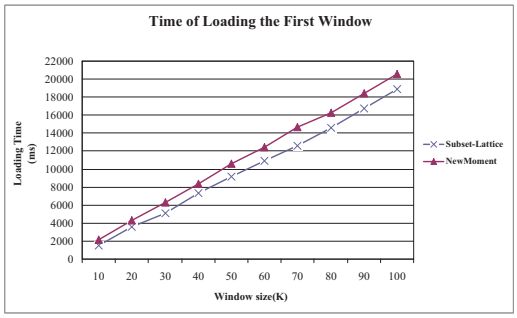

Figure 12. A comparison of the processing time of loading the first window with different sliding window sizes

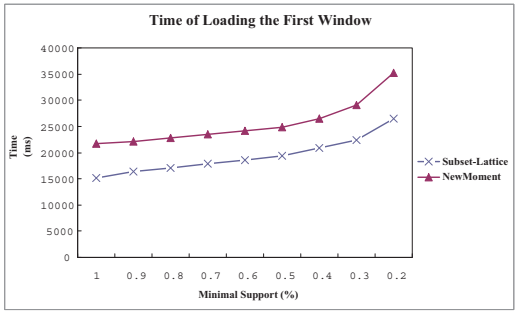

Figure 13. A comparison of the processing time of loading the first window with different minimum supports

We show the comparison of the processing time between our Subset-Lattice algorithm and the NewMoment algorithm, when using the synthetic datasets as the input. Figure 12 shows the comparison of the processing time of loading the first sliding window with different sliding window sizes. The window size is changed from 10K to 100K, and the minimum support threshold is 1%. In the first window, the NewMoment algorithm needs to build a prefix (lexicographic) tree to reserve the candidate itemsets. We can observe that the Subset-Lattice algorithm is faster than the NewMoment algorithm. The reason is that the Subset-Lattice algorithm does not have to generate candidates and can judge the closed frequent itemsets immediately.

Figure 13 shows the comparison of the processing time of loading the first sliding window with different minimum supports. The synthetic data is T5.MT10.I10.MI20.D100k. The minimum support threshold is changed from 1% to 0.1%, and the sliding window size is 100K. When the minimum support decreases, the number of closed frequent itemsets and the processing time increase. From this figure, we can notice that if the support is small enough, the number of candidates of the NewMoment algorithm increases rapidly. Therefore, the NewMoment algorithm takes longer processing time than the Subset-Lattice algorithm.

Figure 14 shows the comparison of the average processing time of the window sliding in each transaction

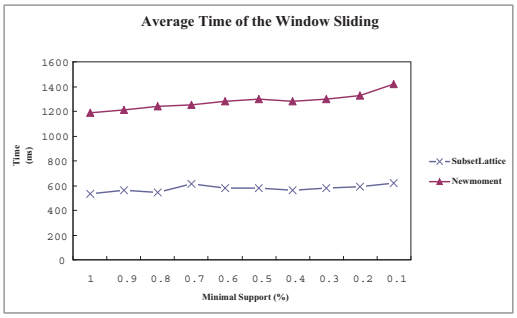

Figure 14. A comparison of the average processing time of the window sliding with different minimum supports

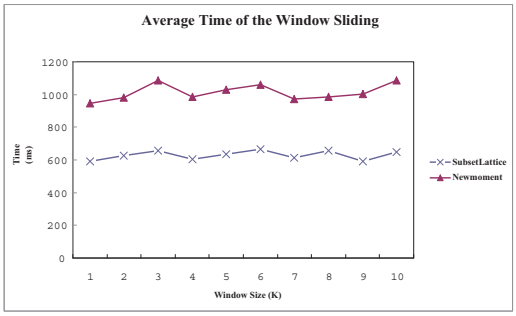

Figure 15. A comparison of the average processing time of the window sliding with different sliding window sizes

with different minimum supports. The synthetic data is T5.MT10.I10.MI20.D100k. The minimum support threshold is changed from 1% to 0.1%, and the sliding window size is 10K. Figure 15 shows the comparison of the average processing time of the window sliding in each transaction with different sliding window sizes. The synthetic data is T5.MT10.I10.MI20.D100k. The sliding window size is changed from 1K to 10K, and the minimum support threshold is 1%. The number of items of these two experiment results are 1000. The time of the window sliding in these two experiment are almost the same. In general, the processing time of the Subset-Lattice algorithm is faster than that of the NewMoment algorithm. The processing time of the window sliding in the Subset-Lattice algorithm is saved about 50%. The reason is that the NewMoment algorithm will regenerate the nodes of k-level ($k > 1$) in the CET-Tree twice, when the window is sliding. The Subset-Lattice algorithm only needs to update the nodes that are already in the lattice structure.

V. CONCLUSION

In this paper, we have designed an efficient algorithm for the problems of mining closed frequent itemsets in data streams. Using the subsumption operation, the Subset-Lattice algorithm can directly judge the closed frequent itemsets from the data streams. When window slides, the Subset-Lattice algorithm will not reconstruct the structure, but the NewMoment algorithm has to reconstruct the whole tree structure. The simulation results have shown that the proposed Subset-Lattice algorithm outperforms the NewMoment algorithm in all relational data settings.

ACKNOWLEDGEMENT

This research was supported in part by the National Science Council of Republic of China under Grant No. NSC-101-2221-E-110-091-MY2.

REFERENCES

[1] R. Agrawal and R. Srikant, "Fast Algorithms for Mining Association Rules in Large Databases," *Proc. of the 20th Int. Conf. on Very Large Data Bases*, pp. 490–501, 1994.

[2] T. Calders, N. Dexters, and B. Goethals, "Mining Frequent Itemsets in a Stream," *Proc. of IEEE Int. Conf. on Data Mining*, pp. 83–92, 2007.

[3] J. H. Chang and W. S. Lee, "A Sliding Window Method for Finding Recently Frequent Itemsets over Online Data Stream," *Journal of Information Science and Engineering*, Vol. 4, No. 11, pp. 753–762, July 2004.

[4] Y. Chi, H. Wung, P. S. Yu, and R. R. Muntz, "Moment: Maintaining Closed Frequent Itemsets over a Stream Sliding Window," *Proc. of the 4th IEEE Int. Conf. on Data Mining*, pp. 59–66, 2004.

[5] C. Giannella, J. Han, J. Pei, X. Yan, and P. Yu, "Mining Frequent Patterns in Data Streams at Multiple Time Granularities," *Proc. of the NSF Workshop on Next Generation Data Mining*, pp. 191–211, 2002.

[6] J. L. Koh and S. N. Shin, "An Approximate Approach for Mining Recently Frequent Itemsets from Data Streams," *Computer Science Data Warehousing and Knowledge Discovery*, Vol. 4081, No. 1, pp. 352–362, Sept. 2006.

[7] H. F. Li, S. Y. Lee, and M. K. Shan, "DSM-PLW: Single-Pass Mining of Path Traversal Patterns over Streaming Web Click-Squences," *Proc. of Computer Networks on Web Dynamics*, pp. 1474–1487, 2006.

[8] J. Li and S. Gong, "Top-k-FCI: Mining Top-k Frequent Closed Itemsets in Data Streams," *Journal of Computational Information Systems*, Vol. 7, No. 13, pp. 4819–4826, 2011.

[9] K. C. Lin, I. E. Liao, and Z. S. Chen, "An Improved Frequent Pattern Growth Method for Mining Association Rules," *Expert Systems with Applications*, Vol. 38, No. 5, pp. 5154–5161, 2011.

[10] C. Raissi, P. Poncelet, and M. Teisseire, "Towards a New Approach for Mining Frequent Itemsets on Data Stream," *Intelligent Information Systems*, Vol. 28, No. 1, pp. 23–36, Dec. 2007.

[11] P. S. Tsai, "Mining Top-K Frequent Closed Itemsets over Data Streams Using the Sliding Window Model," *Expert Systems with Applications*, Vol. 37, No. 10, pp. 6968–6973, 2010.