A Weight-Order-Based Lattice Algorithm for Mining Maximal Weighted Frequent Patterns over a Data Stream Sliding Window

Ye-In Chang, Chia-En Li*, Tzung-Je Chou and Ching-Yi Yen

Department of Computer Science and Engineering National Sun Yat-sen University
70 Lienhai Rd., Kaohsiung, Taiwan
+886-7-5254350, lice@db.cse.nsysu.edu.tw

Abstract

The weighted maximal frequent pattern concerning both the importance and the count is the pattern which is not the subset of any other pattern and the weighted support is large enough. To mine such a pattern based on the sliding window model, the *WMFP-SW* algorithm was proposed. However, when the new transaction comes, it always has to reconstruct its *FP-tree*. To solve the problem, we propose the *WOB* Lattice algorithm and our algorithm provides better performance than the *WMFP-SW* algorithm.

Key words: Sliding Window Model, Lattice, Weighted Maximal Frequent Itemset

Introduction

Data mining is the process of finding hidden and useful knowledge form the large databases. Moreover, data mining has been used in many areas, including geographic [1], network [2], and traffic data [3]. Recently, finding frequent patterns has become one of the most famous ways on data mining. The most important process in frequent pattern mining is mining association rules [4]. By using association rules, the computer can discover new patterns from datasets. However, items have different importance in the real world. Therefore, we have to consider the importance and the count of the items at the same time. Because of this reason, the algorithm of mining association rules cannot be followed in the weight environments and the *Apriori* algorithm [5] also cannot be used in weighted frequent pattern mining. On the other hand, frequent pattern mining has two types of deformations, maximal frequent pattern mining [6] and closed frequent pattern mining [7, 8]. Moreover, weighted frequent pattern mining has two types of deformations, weighted maximal frequent pattern mining [9, 10] and weighted closed frequent pattern mining [11]. Besides, the concept of a data stream, which may have the infinite transactions, is more appropriate than a data set for many recent applications, for example, sensor network data analysis and web click streams. By nature, a stored data set is appropriate when significant portions of the data are queried again and again, and updates are small or relatively infrequent. In contrast, a data stream is appropriate when the data is changing constantly, and it is either unnecessary or impractical to operate on large portions of the data many times.

In [12], Lee *et al.* proposed the Weighted Maximal Frequent Pattern mining over data streams based on the Sliding Window model (*WMFP-SW*) algorithm for mining maximal weighted frequent pattern in data streams within a transaction sliding window. The *WMFP-SW* algorithm uses the *WMFP-FP-tree* to store the transactions in the present window and reconstruct the *WMFP-FP-tree* by items count descending order. While window slides, this algorithm removes the old transactions, insert the new transactions and reconstruct again. To decrease the number of candidates, they find the *MaxW*, the largest weight of the item in the itemset, and calculate *minsup*/*MaxW*. If the count of the itemset is smaller than *minsup*/*MaxW*, it is not a weighted frequent pattern. Finally, they traverse the single-pass to find maximal weighted frequent patterns. It can find weighted maximal frequent pattern based on the sliding window model easily.

The rest of thesis is organized as follows. In section 2, we give a brief description of the *WMFP-SW* algorithm. In Section 3, we present the proposed Weighted-Order-Based Lattice algorithm (*WOB*) algorithm. In Section 4, we present the performance study of our algorithm and make a comparison between our algorithm and the *WMFP-SW* algorithm. Finally, Section 5 gives the conclusion.

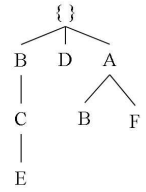
Related Work

Weighted frequent pattern mining is a very important problem in the real databases. Ryu *et al.* propose the *WMFP-SW* algorithm [12] to mine weighted maximal frequent patterns (*WMFPs*) on the sliding window. Their *WMFP-SW* algorithm performs mining operations based on a tree structure, and accordingly, they need tree structures suitable for finding *WMFPs* over sliding window-based data streams. Their *WMFP-SW* algorithm [12] performs mining operations based on the*WMFP-SW-tree* and the *WMFP-SW-array*. This tree structure is similar to the *FP-tree* [13], but the *WMFP-SW-tree* has additional weight data and is constructed with only one scan. The *WMFP-SW-array* stores a part of node data in the *WMFP-SW-tree*. Because weighted candidates do not satisfy the anti-monotone property [14] in general, weighted infrequent pattern could be a subset of a weighted frequent pattern [15]. As a result, incorrect pruning operations by the weights can cause weighted pattern losses. To solve this problem and perform efficient pruning procedures, they define a pruning condition applied in the sliding window model, named *MaxW*. If any single-path is generated in the process of the mining steps, they would calculate *WSup*, the weighted support, for the patterns. If the *WSup* value is not larger than the threshold, they would find other *WMFPs* which are the subsets of the single-path. For example, Figure 1 is an sliding window data stream, and sliding window W_1 has been scanned as shown in Figure 2-(a). Then, the tree has to be reconstructed by the descending count order in Figure 2-(b).

	TID	Transaction
W_1	1	B, C, E
	2	D
W_2	3	A, B
	4	A, F
	5	D, F
	6	D, F, G

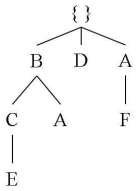
Fig. 1 An example of Transaction Database *TDB*2 in the sliding-window model

Item	Count
B	2
C	1
E	1
D	1
A	2
F	1
G	0



(a)

Item	Count
B	2
A	2
C	1
E	1
D	1
F	1
G	0



(b)

Fig. 2 *WMFP-SW-tree* for sliding window W_1 : (a) before reconstructing operation; (b) after reconstructing operation [14].

A Weight-Order-Based Lattice Algorithm

A. Data Structure

In our algorithm, we propose the weight-order-based lattice structure and the bit-pattern representation of items based on the sliding window model. We use Transaction Database *TD*1 as shown in Figure 3 to illustrate our idea. Because we need weight descending order, we have to sort the items, as shown in Figure 4-(a), by using the weighted order, as shown in Figure 4-(b). This lattice structure has two advantages. First, using the weight-order-based lattice structure, the relationship between the new transaction and present transactions can be easily understood. Second, we can update the support of the transaction efficiently.

	TID	Transaction
W_1	1	B, D, E
	2	C, D
W_2	3	A, B, C, D, E
	4	B
	5	A, B, C, D
	6	D, E

Fig. 3 Transaction Database *TD*1 without sorting

The weight-order-based lattice structure contains the root, nodes, and child-link. An example is shown in Figure 5. The root has no information. It is a start point. When a new transaction is inserted, we sort the transaction by weight descending order and search the weight-order-based lattice structure from the root. Each node records some information:

- Bit-Pattern: It represents an itemset.
- Sup: It represents the support of the itemset.

The child-link points to the subset node. With the child-link, we can check the relationship between nodes and insert the node into the weight-order-based lattice structure efficiently. Moreover, we can increase the support of the related nodes easily.

Item	Weight
A	0.4
B	0.8
C	0.3
D	0.5
E	0.6

(a)

Item	Weight
B	0.8
E	0.6
D	0.5
A	0.4
C	0.3

(b)

Fig. 4 Sorting the items by weight order: (a) before sorting; (b) after sorting.

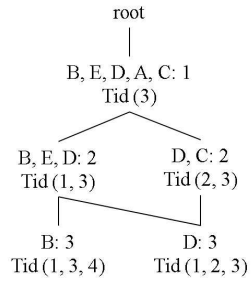


Fig. 5 The lattice of window W_1

B. Our Proposed Algorithm

In this section, we first define some basic definitions and notations and then discuss how to use these techniques to find and represent the weighted maximal frequent itemsets and organize the weighted maximal frequent itemsets in a lattice. Our algorithm has 4 main steps.

1. Transform the itemset to the bit pattern.
2. Check the relation between the new transaction and the old transaction.
3. Use the pruning strategy to prune the patterns.
4. Examine which transaction is the weighted maximal frequent itemsets.

In the first step, we use the bit-pattern representation to store the transaction. We will use weighted descending order to transform the transaction to the representation of bit-patterns. For each transaction *T* in the current window, a bit-pattern of transaction *T* is denoted as Bit(*T*), and the length of bit-pattern is the kind of items in total database. If item *X* is in this transaction and item *X* is the i-th place in the weight descending order, the *i*-th bit of Bit(*T*) is set to 1; otherwise, it is set to 0. For instance, item *D* is the third item in Figure 4-(b). Because of this reason, the bit-pattern of the transaction which has item *D* will be set to 1 in its third place.

In the second step, we check the relation between the new transaction and the old transaction. There are five cases in inserting itemsets into the lattice structure. The flowchart is shown in Figure 6. When the current window becomes full, we will delete the two oldest transactions and insert two new transactions.

In mining the maximal frequent itemsets from the data

stream, each new transaction must be discovered the related relations (equivalent, superset, subset, intersection, empty) of the present transactions. In order to check the relation efficiency, we use the bit-pattern to represent the itemset. The maximal length of the bit-length is the number of distinct items. When the item appears in the transaction, we set the related bit to 1 according to the weight descending order. In Figure 3, Tid_2 is {C, D}, and it would become [D, C] in the weight-order-based lattice structure. We set the third and fifth positions to 1. The bit-pattern is denoted as [00101]. So, the bit-pattern of window W_1 is shown in Table I. For every new transaction which is inserting, we set the support to be 1. After checking the new transaction with each present transactions, the support may be increased and new itemsets may be created. In the next section, we introduce this case in details.

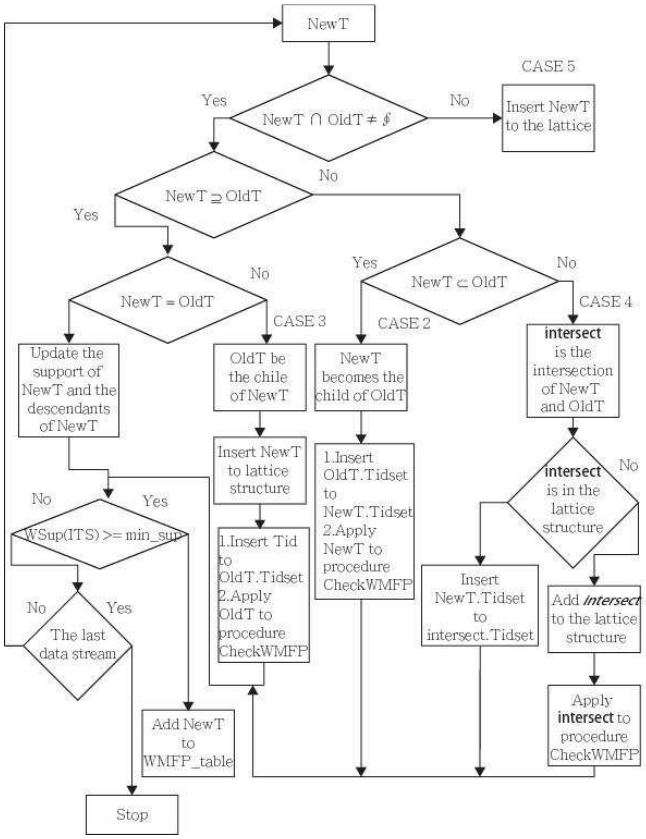


Fig. 6 The flowchart of the algorithm

TABLE I
An example of the data stream mapped to the bit-pattern

Tid	Itemset	Bit-Pattern
1	B, E, D	11100
2	D, C	00101
3	B, E, D, A, C	11111
4	B	10000

In this section, first, we will build an Appearing Table *AppearT* to store the count of the pattern in lattice. After we build the Appearing Table *AppearT*, we will use two pruning strategies: global maximal weight (*GMAX_W*) pruning strategy and local maximal weight (*LMAX_W*) pruning strategy. Note that, *GMAX_W* means the largest weight of the items among all of transactions and *LMAX_W* means the largest weight of the items among items in a certain transaction. In the global pruning strategy, we make use of *GMAX_W*, the maximum weight of item *X* among items appearing in all of transactions. For any pattern *Y*, the average weight of pattern *Y*, Weight(*Y*) must be smaller than or equal to *GMAX_W*. Let Count(*Y*) denote the count of pattern *Y*. Therefore, if *Count*(*Y*) * *GMAX_W* is smaller than the threshold, pattern *Y* will not be the result. That is, pattern *Y* can be pruned. In other words, if *Count*(*Y*) < threshold/*GMAX_W*, pattern *Y* can be pruned. In our algorithm, *Count*(*Y*) = *I*, where *AP*(*I*), the *I*'th entry of Appearing Pattern table, records every pattern *Y* with count = *I*. Therefore, we prune those patterns stored in *AP*(*I*), if *I* is smaller than threshold/*GMAX_W*. Note that *I* is an integer, while threshold/*GMAX_W* is a real number. In order to check whether the subset *X* of a pattern *Y* is frequent or not, we must record the count of subset *X*. Here, similar to the concept of the closed set, we only record the subset *X* which has different count to pattern *Y* in Appearing Table *AppearT*. Note that the definition of a closed set is that the set whose supersets are not frequent or whose count is larger than the count of its superset. Moreover, *AP*(*I*) records patterns with count = *I* in the current window. Basically, only subset *X* which has count lager than that of pattern *Y* will be recorded in *AppearT*.

In the sliding window model, we have to delete the old transactions and insert the new transactions when the window slides. In this section, we describe how to delete the old transaction from the lattice structure and the current window. Step 1: We remove the two old set transaction from the lattice structure. When a transaction is out of the current window, it should be deleted from the lattice structure. In the lattice structure, we need to traverse the nodes which are relevant to the deleted transaction. If a node is deleted from the lattice structure, a subset of the deleted node will be influenced. The reason is that the subset lattice node could be created by two itemsets. Step 2: If the itemset or its subsets which is going to be deleted is already in *WMFP*-table, then the support of the itemset in *WMFP*-table is decreased. However, we will not calculate the support of the itemset. Because it may be inserted again in the next two transactions. For example, a *WMFP* [B, D] will be decreased, since T_1 [B, E, D] is deleted and [B, D] will be less than threshold. However, when T_5 [B, D, A, C] is inserted into the lattice structure, [B, D] become the *WMFP* again. When the fifth transaction T_5 [B, D, A, C] and the sixth transaction T_6 [E, D] come, the current window is full. We have to remove the oldest two transactions from the current window and insert the new two transactions to the current window. The first transaction T_1 [B, E, D] and the second transaction T_2 [D, C] should be removed from the current window. We remove Transaction *Tid* (1) and *Tid* (2) from *Tid* set of T_1 and T_2 and all of the subsets. Because T_1 [B, E, D] has two *WMFPs* [B, E] and [B, D], we have to decrease the count of these two *WMFPs*.

In this section, we will show how to insert transactions into the lattice structure. We process procedure *Insert_T*. When the next transaction T_5 [B, D, A, C] comes, *Insert_T* will call Function *Find_T* to check whether the itemset is in the lattice structure or not. Because of this checking step, we can know that the transaction T_5 [B, D, A, C] is not in the lattice structure. Then, we will call Procedure *CheckCase*.

Performance

In this section, we show the results of the real data. Table II shows the comparison of the processing time of the both algorithms for the Retail dataset under the change of the threshold. From this table, we show that the *WOB-Lattice* algorithm is faster than the *WMFP-SW* algorithm. We observe that the processing time of the *WMFP-SW* algorithm and the *WOB-Lattice* algorithm decreases, when the threshold increases. Because when the threshold increases, the number of the *WMFP* decreases. Figure 7 shows the comparison of processing time of the both algorithms for the Mushroom dataset under the change of the threshold. The detailed results are shown in Table III. From this table, we show that the *WOB-Lattice* algorithm is faster than the *WMFP-SW* algorithm. We observe that the processing time of the *WMFP-SW* algorithm and the *WOB-Lattice* algorithm decreases, when the threshold increases. Because when the threshold increases, the number of the *WMFP* decreases.

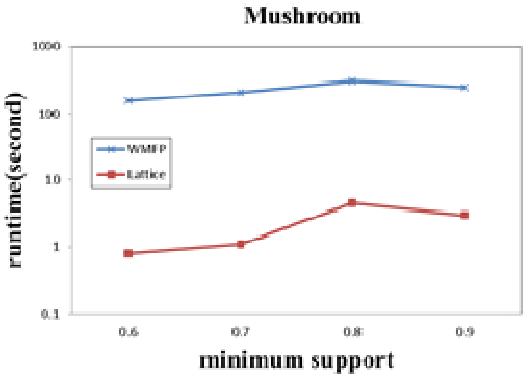


Fig. 7 A comparison of the processing time of the Mushroom dataset under the change of the threshold

TABLE II
A comparison of processing time for the Retail dataset under the change of the threshold

Minimum support	WMFP-SW (msec)	WOB-Lattice (msec)
0.66	2555144	71143
0.68	1000627	60832
0.70	472767	45037
0.72	213852	25147
0.74	100705	11474
0.76	50280	4600
0.78	31624	2756
0.80	21143	2043

TABLE III
The processing time of the Mushroom dataset under the change of the threshold

Minimum support	WMFP-SW (msec)	WOB-Lattice (msec)
0.6	150878	800
0.7	206722	1112
0.8	301995	4520
0.9	244965	3045
1.0	68163	2341

Conclusion

In this paper, we have proposed a *WOB* Lattice algorithm which can avoid the time of reconstructing the tree, while the window slides. In our algorithm, we only need to delete the old transaction and join the new transaction, instead of reconstructing the whole tree shown as in *WMFP-SW* algorithm. Besides, we proposed *GMAXW* pruning rule to decrease the time of mining maximal weight pruning step.

Acknowledgments

This research was supported in part by the Ministry of Science and Technology of Republic of China under Grant No. MOST-106-2221-E-110-079.

References

[1] V. Bogorny, et. al., "Mining Maximal Generalized Frequent Geographic Patterns with Knowledge Constraints," *Proc. of the 6th Int. Conf. on Data Min.*, pp. 813-817, 2006.

[2] G. Fang, et al., "Network Traffic Monitoring Based on Mining Frequent Patterns," *Proc. of the 6th Int. Conf. on Fuzzy Syst. and Knowl. Discov.*, Vol. 7, pp. 571-575, 2009.

[3] W. Liu, et al., "Discovering Spatio-Temporal Causal Interactions in Traffic Data Streams," *Proc. of the 17th ACM SIGKDD Int. Conf. on Knowl. Discov. and Data Min.*, pp. 1010-1018, 2011.

[4] R. Agrawal, et al., "Mining Association Rules Between Sets of Items in Large Databases," *Proc. of the 1993 ACM SIGMOD Int. Conf. on Manag. of Data*, Vol. 22, No. 2, pp. 207-216, June 1993.

[5] R. Agrawal and R. Srikant, "Fast Algorithms for Mining Association Rules," *Proc. of the 20th Int. Conf. Very Large Data Bases, VLDB*, Vol. 1215, pp. 487-499, 1994.

[6] D. Burdick, et al., "MAFIA: A Maximal Frequent Itemset Algorithm for Transactional Databases," *Proc. of the 17th Int. Conf. on Data Eng.*, pp. 443-452, 2001.

[7] L. Chang, et al., "Efficient Algorithms for Incremental Maintenance of Closed Sequential Patterns in Large Databases," *Data Knowl. Eng.*, Vol. 68, No. 1, pp. 68-106, Jan. 2009.

[8] Y. Chen, et al., "A New Approach for Maximal Frequent Sequential Patterns Mining over Data Streams," *Int. J. of Digit. Content Technol. and Its Appl.*, Vol. 5, No. 6, June 2011.

[9] B. Vo, et al., "A New Method for Mining Frequent Weighted Itemsets based on WIT-trees," *Expert Syst. with Appl.*, Vol. 40, No. 4, pp. 1256-1264, March 2013.

[10] J. Wang and Y. Zeng, "DSWFP: Efficient Mining of Weighted Frequent Pattern over Data Streams," *Proc. of the 8th Int. Conf. on Fuzzy Syst. and Knowl. Discov. (FSKD)*, Vol. 2, pp. 942-946, 2011.

[11] U. Yun, "Mining Lossless Closed Frequent Patterns with Weight Constraints," *Knowl.-Based Syst.* Vol. 20, No. 1, pp. 86-97, Feb. 2007.

[12] G. Lee, et al., "Sliding Window Based Weighted Maximal Frequent Pattern Mining over Data Streams," *Expert Syst. with Appl.*, Vol. 41, No. 2, pp. 694-708, Feb. 2014.

[13] J. Han, et al., "Mining Frequent Patterns without Candidate Generation: A Frequent-Pattern Tree Approach," *Data Min. and Knowl. Discov.*, Vol. 8, No. 1, pp. 53–87, Jan. 2004.

[14] R. T. Ng, et al., "Exploratory Mining and Pruning Optimizations of Constrained Associations Rules," *ACM SIGMOD Record*, Vol. 27, pp. 13-24, 1998.

[15] U. Yun, et al., "An Efficient Mining Algorithm for Maximal Weighted Frequent Patterns in Transactional Databases," *Knowl.-Based Syst.*, Vol. 33, No. 1, pp. 53-64, Sept. 2012.