*International Journal of Computer Sciences and Engineering Systems*
*Vol. 10 No. 2 (December, 2016)*

# Efficient Graph Structure for the Mining of Frequent Itemsets from Data Streams

**E.R.NAGANTHAN and F. Ramesh DHANASEELAN**

Department of Computer Science & Engg., Alagappa University, Karaikudi,Tamil Nadu, India
*E-mail:ern_jo, message_to_ramesh@yahoo.com*

***Abstract:*** *In this paper, we propose a graph structure which captures important data streams. This graph can be easily maintained and mined for frequent item sets as well as various other patterns like constrained item sets. This graph captures the contents of transaction in a window and arranges nodes according to some canonical order that is unaffected by changes in item frequency. This graph structure is designed for exact stream mining of regular frequent item sets.*

***Keywords:*** *Stream Mining, Association Rule Mining, Frequent itemsets, Data Stream.*

## 1. INTRODUCTION

A data stream is an ordered sequence of items that arrives in timely order. Different from data in traditional static databases, data streams are continuous, unbounded, usually come with high speed and have a data distribution that often changes with time [1]. One example application of data stream association rule mining is to estimate missing data in sensor networks [2]. Mining from data the following two properties of streams is more challenging due to data streams, ie., the data streams are continuous and unbounded and data in the streams are not necessarily uniformly distributed; their distributions are usually changing with time. In recent years, several stream mining algorithms have been proposed, and they can be broadly categorized into two classes, ie., exact and appropriate algorithms. Exact algorithms [3] find truly frequent item sets and appropriate algorithms [4] find frequent item sets by using appropriate procedures, ie., these algorithms may find some infrequent item sets or may miss some frequent item sets. We propose a graph structure, which is designed for exact stream mining of regular frequent item sets. The graph captures the contents of relevant transactions in the streams. When the streams flow through, a fixed size user window containing the interesting portion of the streams, ie., the recent data is properly updated.

Traditional association rule mining algorithms are developed to work on static data and, thus, can not be applied directly to mine association rule in stream data [8, 9, and 15]. The first recognized frequent item sets mining algorithm for traditional databases is Apriori [5]. After that, many other algorithms based on the ideas of Apriori were developed for performance improvement [6]. Apriori-based algorithms require multiple scans of the original database, which leads to high CPU and I/O costs. Therefore, they are not suitable for a data stream environment, in which data can be scanned only once. Another category of association rule mining algorithms for traditional databases proposed by Han and Pei [7] are those using a frequent pattern tree (FP-tree) data structure and an FP-growth algorithm which allows mining of frequent item sets without generating candidate item sets. Compared with Apriori-based algorithms, it achieves higher performance by avoiding iterative candidate generations. However, it still can not be used to mine association rule in data streams [10, 13] since the construction of FP-tree requires two scans of data. The rest of the paper is organized as follows. Section 2 introduces our graph structure for stream mining. Section 3 shows experimental results. Finally, conclusions are presented in section 4.

## 2. GRAPH STRUCTURE FOR DATA STREAM

The graph structure is designed for exact stream mining [11, 14]. The construction of the graph structure only requires on scan of the streaming data. The graph structure captures the contents of transactions in each batch of streaming data.

We arrange transaction items according to some canonical order, which can be specified by the user prior to the graph construction or the mining process. For example, items can be consistently arranged in lexicographic order or alphabetical order. Alternatively, items can be arranged according to some specific order depending on the item properties-such as their price values or their validity to some constraints, which can also be determined prior to the graph construction or the mining process. We keep a list of frequency counts at each node.

Whenever a new batch of transactions flows in, we append to this list at each node its frequency count in the current batch. In other words, the last entry of the list at node X shows the frequency count of X in the current batch.

When the next batch of transactions comes in, the list is shifted forward. The last entry shifts and becomes the second –last entry; this leaves room for the newest batch. At the same time, the frequency count corresponding to the oldest batch in the window [12] is removed. This has the same effect as deleting from the window the transactions in the oldest batch.

We use a pointer to indicate the last update at each node. If the pointer points to the previous entry in the list of frequency counts at a node X, then this indicates that X has just been visited at the update of the last batch. On the other hand, if the pointer points to a much earlier entry in the list at a node Y, then this indicates that Y has not been visited since then and that the frequent counts of Y for the entries in–between should be 0s.

Since the graph structure is constructed independent of minsup, every transaction in the current window is captured. Once such a tree is constructed, we can mine frequent itemsets from it in a fashion similar to FP–growth [16] (using minsup). Since items are consistently arranged according to some canonical order, one can guarantee the inclusion of all frequent items using just upward traversals. There is also no worry about possible omission or doubly–counting of items during the mining process. Consequently, we find all and only those truly frequent itemsets because we use minsup for mining and because every transaction in the current window is captured in the graph structure.

To summarize, transaction items are arranged according to some canonical order in our graph structure so that the ordering is unaffected by the changes in frequency caused by the continuous nature of streams. When the window slides, transactions in the oldest batch can be easily "detected" by shifting the list of frequency counts. The effective use of pointer at each node helps us avoid performing the expensive tree traversal of all nodes. Moreover, mining is "delayed" until it is needed. Since our graph structure is always kept up-to-date, frequent itemsets in current streams can be found effectively. By using such a "delay evaluation" scheme, we avoid lots of unnecessary computation. As streams are continuously flowing rapidly, computation spent on older batches of transactions may have been wasted if these batches get removed from the current window before the user mines for frequent itemsets. Consider the following stream of transactions:

| *Batch* | *Transactions* | *Contents* |
|---|---|---|
| First | $t_1\ t_2\ t_3$ | {a,b,c}{a} {a,c} |
| Second | $t_4\ t_5\ t_6$ | {a,c,d}{b,d}{a,b,d} |
| Third | $t_7\ t_8\ t_9$ | {b,d}{a,b,c,d}{a,c} |

Let minsup be 3 and let the window size *w* be 2 batches (indicating that only two batches of transactions are kept) Then, when the first two batches of transactions in the streams flows in, we insert the transactions into our graph structure and keep frequency counts in a list of *w* entries at each node. Each entry in the list corresponds to a batch. For example, the node a: 3: 2 in figure 1 indicates that the frequency of a is 3 in the first batch and is 2 in the second batch.

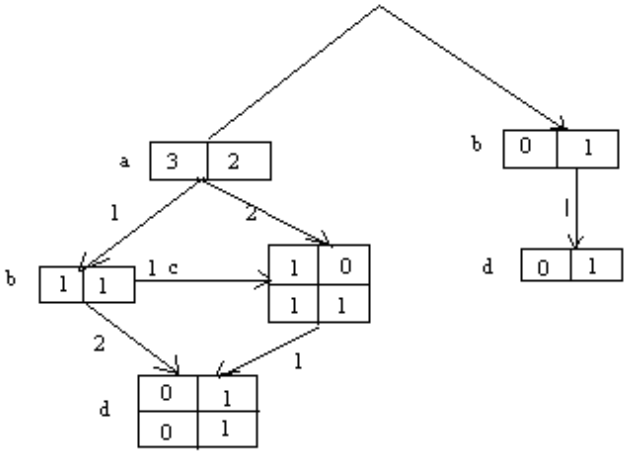


**Figure 1:** At time T (the Graph Structure Capturing 1st and 2nd Batches)

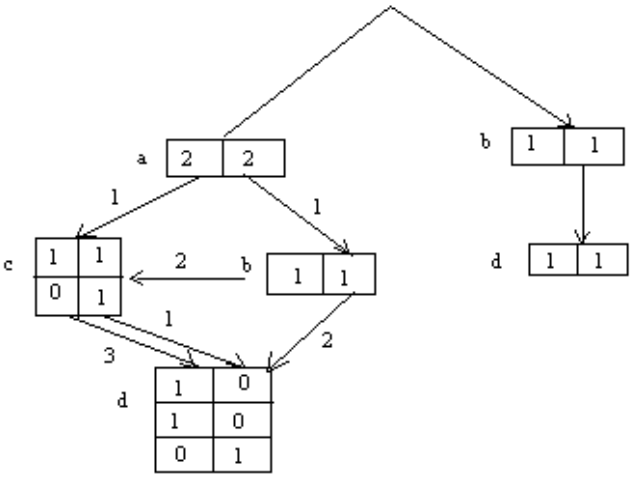


**Figure 2:** At time T′ (the graph structure capturing 2nd and 3rd batches)

Afterwards (at time T′), when the third batch of streaming data flows in, we insert the transactions in our graph structure. The list of frequency counts shifts, the frequent counts for the oldest (i.e., the first) batch are removed–leaving room for the frequency counts for the second and the third (i.e., the two newest) batches of transactions. For example, if we call the mining process at time T′, we get frequent itemsets {*a*}:4, {*a, c*}:3, {*a, d*}: 3, {*b, d*}:4, {*c*}: 3 and {*d*}: 5.

## 3. EXPERIMENTAL RESULTS

In the experiments, we mainly evaluated the accuracy and efficiency of our graph structure. In the first experiment, we

measured the accuracy of graph structure. As the graph structure captured the most recent batches of transactions in the streams, we compared the frequent item sets returned by mining directly form these transactions with those returned by mining from our graph structure. The experimental results show that mining from our graph structure led to 100% accuracy. In other words, mining from the graph structure returned all and only those truly frequent item sets. All the returned item sets were frequent, and all frequent item sets were returned. This shows that our graph structure, which is designed for an exact stream mining. The graph structure captures the contents of transactions in the streams.

In the second experiment, we measured the efficiency of our graph structure. We compared the runtime of mining from our graph structure with that of using the DSTree. The x-axis shows the number of batches in the current window. The y-axis shows the run time. Figure 3 shows the run time. When the number of batches increased, the run time of mining from our graph structure slightly increased. But it is better than DSTree.

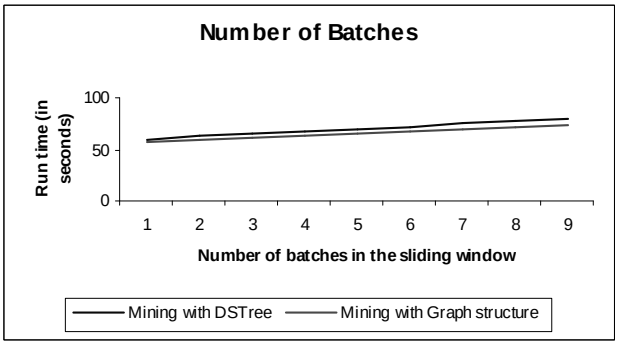


**Figure 3:** Number of Batches

The result of third experiment show that, our graph structure required less space than the DSTree.

## 4. CONCLUSIONS

This graph structure captures the contents of transactions in a window, and arranges tree nodes according to some canonical order that is unaffected by changes in item frequency. Mining from this graph structure returned all and only those truly frequent item sets. All the returned item sets were frequent, and all frequent item sets were returned. So this graph structure is suitable for exact stream mining of regular frequent item sets.

## REFERENCES

[1] Studipto Guha, Nick Koudas, Kyuseok Shine; Data stream and Histogram; *ACM Symposium on Theory of Computing*; 2001.

[2] Mihail Kalatcher and Le Gruenwald; Estimating Missing values in Related Sensor Data streams; *Int'l Conf. on Management of Data*; January 2005.

[3] Y. Chi *et al.* Moment: Maintaining Closed Frequent Item Sets Over a Stream Sliding Window. *In proc. ICDM* 2004, pp. 59-66.

[4] C. Giannella *et al.* Mining Frequent Patterns in Data Streams at Multiple Time Granularities. In Data Mining: Next Generation Challenges and Future Directions, *AAAI/MIT Press*, 2004, ch. 6.

[5] Rakesh Agarwal, Tomasz Imielinski, Arun Swami; Mining Association Rule between Sets of Items on Massive Databases; *Int'l Conf. on Management of Data*; May 1993.

[6] Jaiwei Hans, Guozhu Dong, Yiwen Yin; Efficient Mining of Partial Periodic Patterns in Time Series Database; *IEEE Int'l Conference on Data Mining*; March 1999.

[7] Jaiwei Hans, Jian Pei, Yiwen Yin; Mining Frequent Patterns without Candidate Generation: *Int'l Conf. on Management of Data*; May 2000.

[8] Mohamed Medhat Gaber, Arkady Zaslavsky and Shonali Krishnaswamy; Mining Data Streams: A Review; SIGMOD Record, Vol. 34, No. 2, June 2005.

[9] Nan Jiang and Le Gruenwald; Research Issues in Data Stream Association Rule Mining; *SIGMOID Record*, Vol. 35, No. 1, March 2006.

[10] Carson Kai-Sang Leung and Quamrul I.Khan; DSTree: A Tree Structure for the Mining of Frequent Sets from Data Streams; IEEE *Sixth Int'l. Conf. on Data Mining*, 2006.

[11] J.X. Yu *et al.*; False Positive or False Negative: Mining Frequent Item Sets from High Speed Transactional Data Streams; *In Proc. VLDB*, 2004, pp. 204-215.

[12] Chih-Hsiang Lin, Ding-Ying Chiu, Yi-Hung Wu, Arbee L.P. Chen; Mining Frequent Item Sets from Data Streams with a Time-Sensitive Sliding Window; *SIAM Int'l Conf. on Data Mining*, April 2005.

[13] Moses Charikar, Kevin Chen, Martin Farach-Colton; Finding Frequent Items in Data Streams; *Theoretical Computer Science*; January 2004.

[14] L.Golab and M.T Ozsu; Issues in Data Stream Management; *In SIGMOID Record*, Vol. 32, No. 2, June 2003.

[15] Joong Hyuk Chang, Won Suk Lee; A Sliding Window Method for Finding Recently Frequent Itemsets over Online Data Streams; *Journal of Information Science and Engineering*, July 2004.

[16] J. Han *et al.*; Mining Frequent Patterns Without Candidate Generation; *In Proc. SIGMOID* 2000, pp. 1-12.