Mining Regular Patterns in Data Streams Using Vertical Format

G. Vijay Kumar *gvijay_73@yahoo.co.in*
School of Computing
K L University
Guntur – 522 502, India

M. Sreedevi *msreedevi_27@yahoo.co.in*
School of Computing
K L University
Guntur – 522 502, India

NVS Pavan Kumar *nvspavankumar@gmail.com*
School of Computing
K L University
Guntur – 522 502, India

Abstract

The increasing prominence of data streams has been lead to the study of online mining in order to capture interesting trends, patterns and exceptions. Recently, temporal regularity in occurrence behavior of a pattern was treated as an emerging area in several online applications like network traffic, sensor networks, e-business and stock market analysis etc. A pattern is said to be *regular* in a data stream, if its occurrence behavior is not more than the user given regularity threshold. Although there has been some efforts done in finding *regular* patterns over stream data, no such method has been developed yet by using vertical data format. Therefore, in this paper we develop a new method called VDSRP-method to generate the complete set of regular patterns over a data stream at a user given regularity threshold. Our experimental results show that highly efficiency in terms of execution and memory consumption.

Keywords: Data Streams, Temporal Regularity, Regular Pattern, Vertical Data.

1. INTRODUCTION

Unlike mining static databases, data stream mining [1, 2, 3] creates many new challenges. It is unrealistic to keep the entire stream in the main memory or even in a secondary storage device because data stream is a continuous, massive (e.g., terabytes in volume), unbounded, timely ordered series of data elements generates at a rapid rate. Incredible volumes of data streams are often generated by communication networks, Internet traffic, real-time surveillance systems, online transactions in the financial market, remote sensors, scientific and engineering experiments and other dynamic environments. Discovering knowledge in data streams is an important research area in data mining and knowledge discovery process.

Mining Frequent patterns [4, 5] from static databases has been broadly studied in Stream data mining [1, 2, 3]. Apriori algorithm [5] is a classical algorithm proposed by R. Agarwal and R. Srikanth in 1993 for mining frequent item sets for Boolean association rules. The algorithm uses prior knowledge and employs an iterative approach known as a level–wise search to generate frequent item sets. First it generates with 1-item sets, recursively generates 2-item set and then frequent 3-item set and continues until all the frequent item sets are generated. Later Han et. al [4] proposed the frequent pattern tree (FP-tree) and FP-growth algorithm to mine frequent patterns without candidate generation. The Apriori and FP-growth algorithms find outs the occurrence frequencies of a pattern i.e., support. Several algorithms have been proposed so far to mine frequent patterns in a transaction databases as well as in data streams. However, the significance of a pattern may not always depend upon the occurrence frequency of a pattern (i.e.,

support). The significance of a pattern may also depend upon other occurrence characteristics such as temporal regularity of a pattern. For example, to improve web site design the web site administrator may be interested in regularly visited web page sequence rather than heavily hit web pages only for a specific period of time. Also, in a retail market some products may have regular demand than other products. To know how regularly a product has been sold is essential rather than the occurrence frequency of a product. Therefore, finding patterns at regular intervals also plays an important role in data mining.

Recently, Tanbeer et. al [6] introduced a new problem of discovering Regular Patterns that follow a temporal regularity in their occurrence behavior. With the help of user given maximum regularity measure at which pattern occurs in a database is called a *regular* pattern. They also extended the same problem in Data Streams. They proposed a tree based data-structure, called RPS-tree [7] that captures user-given regularity threshold and mines regular patterns in a data stream with the help of FP-growth algorithm and conditional pattern bases and corresponding conditional trees. Therefore, in this paper, we propose a new method called Vertical Data Stream Regular Patterns method (VDSRP - method in short), using the same Data Stream which is in [7] to mine regular patterns using vertical data format. By using Vertical Data Format [8, 9, 10, 11], it will be able to judge the non-regular item sets before generating candidate item sets. The main idea of our new method is to develop a simple, but yet powerful, that captures the data stream content in a window by using sliding-window technique to find regular items. The experimental results show the effectiveness of VDSRP-method in finding regular patterns in a Data Stream.

The rest of the paper is organized as follows. Section 2 summarizes the existing tree structure to mine regular patterns. Section 3 introduces the problem definition of regular pattern mining. The method of VDSRP to find regular patterns using vertical data format are given in section 4. Section 5, our experimental results are shown. Finally, we conclude the paper in section 6.

2. RELATED WORK

In data mining, one of the most important techniques is Association rule mining. It was first introduced by Agarwal *et al*. [5]. It extracts frequent patterns, correlations, associations among sets of items in databases. The main drawback with the classical Apriori algorithm is that it needs repeated scans to generate candidate set. After that Frequent pattern tree and FP-growth algorithm [4] is introduced by Han *et al*. to mine frequent patterns without candidate generation. Periodic patterns [12], [13] and Cyclic patterns [14] are also closely related with Regular patterns. Periodic pattern mining in time-series data focuses on the cyclic behavior of patterns either in whole or some part of time-series. Although periodic pattern mining is closely related with our work, it cannot be applied directly to mine regular patterns from a data stream because it process with either time-series or sequential data.

Tanbeer *et al*. [7] have proposed a tree based data-structure, called RPS-tree that captures user-given regularity threshold and mines regular patterns in a data stream with the help of FP-growth [4] algorithm and conditional pattern bases and corresponding conditional trees. First, they constructed RPS-tree consists of one root node referred to as “null” and a set of item-prefix sub-trees called children of the root. Each node in an RPS-tree represents an itemset in the path from the root up to that node. The RPS-tree maintains the occurrence information of all transactions in the current window with the tree structure. Also RPS-tree maintains two types of nodes called ordinary nodes and tail nodes. Nodes of both types explicitly maintain parent, children and node traversal pointers. In addition each tail node maintains a tid-list and a tail-node pointer. The tail-node pointer points to either the next tail node in the tree if any, or “null”. Then they construct an item header table called RPS-table consists of each distinct item in the current window with relative regularity and a pointer pointing to the first node in the RPS-tree that carries the item. RPS-table of a RPS-tree consists of three fields, they are item name (i), regularity of i (r), and a pointer to the RPS-tree for i (p). Similar to FP-growth mining, they mine the RPS-tree of decreasing size to generate regular patterns by creating conditional pattern-bases and corresponding conditional trees.

.

3. PROBLEM DEFINITION

Let $I = \{i_1, i_2, \ldots, i_n\}$ be the set of items. A set $X = \{i_j, \ldots, i_k\} \subseteq I$, where $j \leq k$ and $j, k \in [1, n]$ is called a *pattern* (or an *itemset*). A transaction $t = (tid, Y)$ is a couple where *tid* is a transaction-*id* and Y is a *patter* or an itemset. If $X \subseteq Y$, which means that t contains X or X occurs in t. Let *size*(t) be the *size* of t, i.e., the number of items in Y.

3.1 Definition 1 (Data Stream)

A data stream DS can be defined as infinite sequence of transactions, i.e., $DS = [t_1, t_2, \ldots, t_m, \ldots]$, $i \in [1, m]$ where t_i is the i^{th} arrived transaction. A window W can be referred to as a set of all transactions between the i^{th} and j^{th} arrival of transactions, where $j > i$ and the size of W is $|W| = j - i$, i.e., the number of transactions between i^{th} and j^{th} arrival of transactions. Let each slide of window introduce and expire *slide_size*, $1 \leq slide\_size \geq |W|$, transactions into and from the current window. If X occurs in t_j, $j \in [1, |W|]$, such transactions-*id* is denoted as t_j^X, $j \in [1, |W|]$. Therefore $T_w^X = \{ t_j^X, \ldots t_k^X \}$, $j, k \in [1, |W|]$ and $j \leq k$ is the set of all transaction-*id*s where X occurs in the current window W.

3.2 Definition 2 (A period of *X* in *W*)

Let t^X_{j+1} and t_j^X, $j \in [1, (|W|-1)]$, be two consecutive transaction-*id*s in T_w^X. the number of transactions between t^X_{j+1} and t_j^X is defined as a period of X, say p^X (i.e., $p^X = t^X_{j+1} - t_j^X$, $j \in [1, (|W|-1)]$). For the simplicity of period computation, a "*null*" transaction with no item is considered at the beginning of W, i.e., $t_f = 0$(*null*), where t_f represents the *tid* of the first transaction to be considered. Similarly, t_l, the *tid* of the last transaction to be considered, is the *tid* of the $|W|^{-th}$ transaction in the window, i.e., $t_l = t_{|W|}$. For instance, the stream data in Table 1, consider the window is composed of eight transactions (i.e., *tid* = 1 to *tid* = 8 make the first window, say W_1). Then set of transactions in W_1 where pattern (b, c) appears in (2, 3, 5). Therefore, the periods for (b, c) are $\{(2 - t_f) = 2, (3 - 2) = 1, (5 - 3) = 2$ and $(t_l - 5) = 3\}$, where $t_f = 0$ and $t_l = 8$.

The occurrence periods of X in W defined as above will be the precise information about the occurrence behaviour of a pattern. A pattern will not be a regular in W, if it appears after large period at any stage. The largest occurrence period of a pattern can provide the upper limit of its periodic occurrence characteristic. Hence, the measure of the characteristic of a pattern of being regular in a W (i.e., the regularity of a pattern in W) can be defined as follows.

3.3 Definition 3 (Regularity of a pattern *X* in *W*)

Let in a TwX, PwX be the set of all periods of X in W i.e., PwX = { p1X , . . . , psX}, where s is the total number of periods of X in W. Then, the regularity of X in W can be denoted as regw(X) = Max(p1X, ..., psX}. For example, in DS of Table 1 regw(b, c) = 3, since Pw1{b, c} = Max(2, 1, 2, 3) = 3. Therefore a pattern is called a regular pattern in W if its regularity in W must not more than a user given maximum regularity threshold called max_reg λ, with $1 \leq \lambda \leq |W|$. The regularity threshold is given as the percentage of window size.

Therefore the regular patterns in W satisfy the downward closure property [6]. i.e., if a pattern is found to be regular, then all of its non-empty subsets will be regular. Accordingly, if a pattern is not regular, then none of its supersets can be regular. Given DS, $|W|$, and *max_reg*, finding the complete set of regular patterns in W, R_w that have regularity of not greater than the *max_reg* value is the problem of mining regular patterns in data stream.

4. MINING REGULAR PATTERNS

In contrast with traditional data sets, the continuous flow (in and out) of stream data in a computer system updates with varying rates. So, we had taken sliding window technique and vertical data format to mine regular patterns from the data stream.

Let Figure 1. be the data stream which contains transaction-*id* i.e., *tid*, and itemset i.e., *transaction* with respect to *tid.* Now consider the window size may be 8 i.e., the size of the window |W| = 8. Let the first window W_1 handles the transactions of data stream from *tid*-1 to *tid*-8, now convert this W_1 into vertical data format i.e., (itemset : tid). Then find out the *periods* for each itemset which are given in the data stream to get the regular patterns which are less than or equal to the user given regularity threshold. After finding W_1 regular patterns generate second window W_2 and repeat the same procedure to find out latest regular patterns from the data stream.

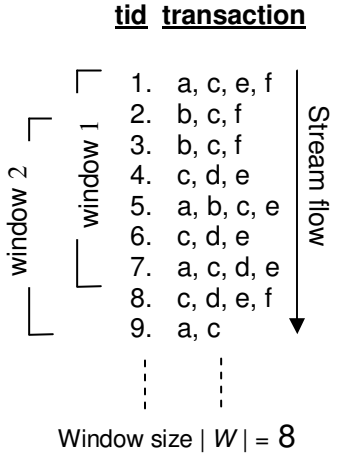

FIGURE 1: A data stream *DS*

Our proposed method is given below to mine regular patterns from the data stream with the help of sliding-window technique and vertical data format. Both the Apriori algorithm and FP-growth algorithm mine frequent patterns in Horizontal data format (i.e., {TID : itemset}), where TID is a transaction-id and itemset is the set of items in transaction TID. But the data can also be present in {item : TID-set} format where item is an item name and TID-set is the set of transactions containing the item. This is known as Vertical data format. We are going to mine regular patterns from the given data stream using vertical format.

VDSRP – Method

Input : DS, $\lambda = 3$
Output: Complete set of regular patterns

Procedure:

1. For each window W of size 8 in DS
2. Convert T_w into VT_w
3. For each item i in X where $X \subseteq VT_w$
4. If(Find R(i) > λ) //FindR returns max_reg
5. Delete i
6. Else
7. {
8. Result ← Result ∪ (i)
9. For each item i_{next} in X
10. {
11. If(Find R (i_{next}) > λ) }
12. Delete i_{next}
13. Else
14. Result ← Result ∪ (i, next)
15. Do “and” operation till all regular itemsets found
16. }
17. Update *W*(*i, j*)

By using the example data stream in Figure 1. We convert the first window into vertical database format and then we calculated period of X as explained in problem definition.

Itemset	Tid-set	P^X	R
a	1, 5, 7	1,4,2	4
b	2, 3, 5	2,1,2,3	2
c	1,2,3,4,5,6,7,8	1,1,1,1,1,1,1,1	1
d	4,6,7,8	4,2,1,1	4
e	1,4,5,6,7,8	1,3,1,1,1,1	3
f	1,2,3,8	1,1,1,5	5

TABLE 1: VDSRP-table with P^X and R

Finding periods is a simple procedure in our VDSRP-method. Create individual array to every itemset. Find all the periods of each itemset by subtracting i.e., $p^X = t^X_{j+1} - t^X_j$. Among number of periods that we get from the window we consider the maximum period from the tid-set of each itemset. For example in Table 1, the periods of {a} are (1, 4, 2). Its regularity is 4 because it is the maximum regularity for {a} and compare with the user given regularity threshold i.e., $\lambda = 3$. So {a} is not a regular itemset because it is greater than the user given threshold.

Itemset	Tid-set	P^X	R
(b, c)	2,3,5	2,1,1,3	3
(b, e)	5	5,3	5
(c, e)	1,4,5,6,7,8	1,3,1,1,1,1	3
(b, c, e)	5	5,3	5

TABLE 2: VDRSP-table with P^X and R

After getting 1-item set we go for 2-item set as shown in Table 2. The regular patterns in W_1 satisfy the down-ward closure property [5] i.e., if a pattern that found regular then all of its non-empty subsets will be regular, if a pattern which is not regular then none of its supersets can be regular. So we consider only the itemsets which are found regular to generate k+1 itemsets. From Table 2 we can say that itemsets (b, c), (c, e) are 2-item regular itemsets. We mine with the same procedure until no regular item set generated in the window.

Itemset	Tid-set	P^X	R
a	4,6,8	4,2,2	4
b	1,2,4	1,1,2,4	4
c	1,2,3,4,5,6,7,8	1,1,1,1,1,1,1,1,	1
d	3,5,6,7	3,2,1,1,1	3
e	3,4,5,6,7	3,1,1,1,1,1	3
f	1,2,7	1,1,5,1	5

TABLE 3: VDRSP-table with P^X and R from W_2

Generally, the regularity of patterns may change with the sliding of window i.e., with the deletion of old transaction and the insertion of new transaction. For example, in Table 1 and Table 2 the regular patterns {b} and {b, c} in W_1 become irregular patterns in W_2 because their regularity is greater than *max_reg*. Again, the irregular patterns {d} and {c, d, e} in W_1 become regular in W_2. Therefore to reflect the correct regularity of each item in the current window, we perform the refreshing operation on VDSRP-table to get new window. The process continues to W_2, W_3 and soon to find out the latest regular patterns from the data streams.

Itemset	Tid-set	P^X	R
(c, d)	3,5,6,7	3,2,1,1,1	3
(c, e)	3,4,5,6,7	3,1,1,1,1,1	3
(d, e)	3,5,6,7	3,2,1,1,1	3
(c, d, e)	3,5,6,7	3,2,1,1,1	3

TABLE 4: VDRSP-table with P^X and R from W_2

5. EXPERIMENT RESULTS

In this section we are going to present our results. All the programs are written in VC++ 6.0 and executed in Windows XP on a 2.66 GHz machine with 2GB of main memory. We used our VDSRP-method over several synthetic and real datasets which are frequently used to find out frequent pattern mining experiments. With our proposed method we present our experiment results by comparing with the existing RPS-tree. The detailed characteristics of the datasets are available in Table 5 which are obtained from [15].

Dataset	#Trans	#Items	MaxTL	AvgTL	Type
Kosarak	9,90,000	41,270	673	8.10	Real
Mushroom	8,124	119	23	23	Real
T1014D100K	1,00,759	870	30	10.10	Synthetic

TABLE 5: Database Characteristics

The above table shows some statistical information about the datasets. We consider the slide_size = 1 for all the experiments. We report the results on Kosarak dataset which contain 9,90K transactions, 41,270 items and 8.10 average transaction length. We also report on T1014D100K dataset which contains 1,00,759 transactions, 870 items, 10.10 average transaction length and also on mushroom dataset which contains 8,124 transactions, 119 items and 23 is the average transaction length.

5.1 Memory Efficiency

The memory requirements for our VDSRP-method on different datasets with different window sizes are shown in Table 6. For example, in kosorak dataset when window size is 100K, the memory required on an average of 3.57MB and when window size is 500K, it requires on an average of 16.83 MB. Hence from table 6 it can be observed that VDSRP-method is memory efficient on different real and synthetic datasets.

Kosorak	W1 (100K)	W2 (300K)	W3 (500K)	W4 (700K)	W5 (900K)
	3.57 MB	10.71 MB	16.83 MB	24.96 MB	32.1 MB
Mushroom	W1 (1 K)	W2 (3 K)	W3 (5 K)	W4 (7 K)	W5 (8 K)
	0.7 MB	0.14 MB	0.31 MB	0.48 MB	0.56 MB
T1014D100K	W1 (20 K)	W2 (40 K)	W3 (60 K)	W4 (80 K)	W5 (100 K)
	0.82 MB	1.74 MB	2.57 MB	3.31 MB	4.12 MB

TABLE 6: Memory Requirement for different window sizes

5.2 Runtime Efficiency

From figures 2(a) and 2(b) we can see that our proposed method runs faster than RPS-tree under various regularity thresholds and with different window sizes respectively. We conducted experiments on kosarak dataset with window size 500K on different max_reg(%) values. In figure 2(a) y-axis shows different regularity threshold values and x-axis shows the average total time taken to convert the data into vertical format and mining time as well. Our proposed method is taking on an average 38 seconds time when max_reg is 0.04%. If the max_reg value increases, the execution time also increases to mine regular patterns from the window.

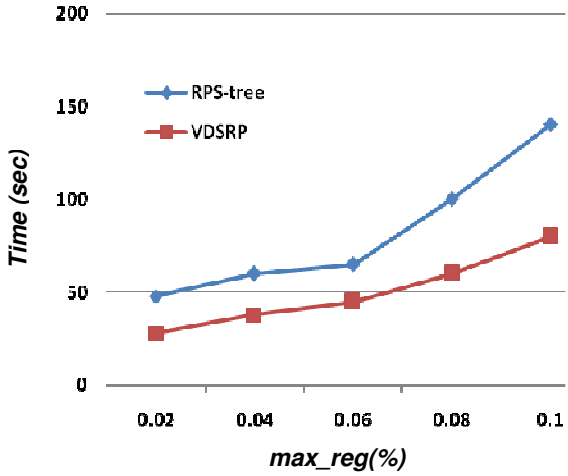

FIGURE 2 (a) On Kosarak (|W|)= 500K

In figure 2(b) the graph shows x-axis with different window sizes and y-axis with the average time taken in seconds to mine regular patterns at max_reg is 0.06%. Our proposed method takes less average time when compare with RPS-tree on different window sizes. For example, when window size is 300K the average time taken is only 45 seconds.

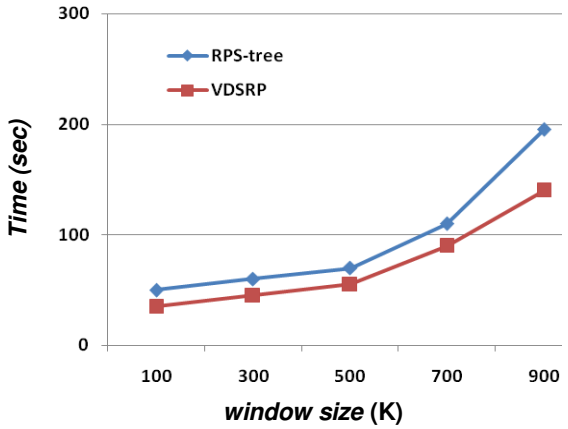

FIGURE 2 (b) On Kosarak (max_reg = *0.06%)*

6. CONCLUSION

In this paper we presented a VDSRP method which is much better than the existing RPS-tree algorithm because it uses sliding-window technique and the advantages of Vertical Database Format. This method is very simple to use with simple operations like arrays, unions, intersection, deletion etc. to find out regular patterns over data streams. Our experiment results outperforms in both execution and memory consumption.

7. REFERENCES

[1] S.K. Tanbeer, C.F. Ahmed, B.-S. Jeong, Y.-K. Lee "Sliding Window-based Frequent Pattern Mining over Data Streams. Information Sciences", 179, 2006, pp. 3843-3865.

[2] C.K.-S. Leung, , Q.I. Khan "DSTree: A Tree Structure for the mining of Frequent Sets from Data Streams." In: ICDM, 2006, pp. 928-932.

[3] H.-F. Li, S.-Y. Lee "Mining Frequent Itemsets over Data Streams Using Efficient Window Sliding Techniques." Expert Systems with Applications 36, 2009, pp. 1466-1477.

[4] J. Han, J. Pie, Y. Yin "Mining Frequent Patterns without candidate generation", In Proc. ACM SIGMOD international Conference on management of Data, 2000, pp. 1-12.

[5] R. Agarwal, and R. Srikanth, "Fast algorithms for mining association rules in Large databases", In Proc. 1994 Int. Conf. Very Large Databases VLDBA'94, Santiago, Chile, Sept. 1994, pp. 487- 499.

[6] S. K. Tanbeer, C. F. Ahmed, B.S. Jeong, and Y.K. Lee, "Mining Regular Patterns in Transactional Databases", IEICE Trans. On Information Systems, E91-D, 11, 2008, pp. 2568-2577.

[7] S.K. Tanbeer, C.F. Ahmed, B.S. Jeong. "Mining regular patterns in data streams." In: DASFAA. Volume 5981 of LNCS., Springer 2010, pp. 399-413.

[8] J. Han, M. Kamber, "*Data Mining :Concepts and Techniques*", 2nd ed. An Imprint of Elsevier, Morgan Kaufmann publishers, 2006, pp. 468-489.

[9] G. Yi-ming, W. Zhi-jun, "A Vertical format algorithm for mining frequent item sets", IEEE Transactions, pp. 11-13, 2010.

[10] M. J. Zaki, K. Gouda. "Fast Vertical Mining using Diffsets", *SIGKDD* '03, Copyright 2003 ACM 1-58113-737-0/03/0008, August' 24 – 27, 2003.

[11] G. Vijay Kumar, M. Sreedevi, NVS. Pavan Kumar. "Mining Regular Patterns in Transactional Databases using vertical Format", International Journal of Advanced Research in Computer Science, vol. 2, pp. 581-583, Sep-Oct 2011.

[12] M.G. Elfeky, W.G. Aref, A.K. Elmagarmid "Periodicity detection in time series databases." IEEE Transactions on Knowledge and Data Engineering 17(7), pp. 875-887 2005.

[13] G. Lee, W. Yang, J-M Lee. "A Parallel algorithm for mining partial periodic patterns." Information Society 176, pp. 2006, pp.3591-3609.

[14] B. Ozden, S. Ramaswamy, A. Silberschatz. "Cyclic Association Rules." In.: 14th International conference on Data Engineering, 1998, pp. 412-421.

[15] Frequent Itemset Mining Dataset Repository http://fimi.cs.helsinki.fi/data/ and UCI machine learning repository (University or California).