A New Adaptive Algorithm for Frequent Pattern Mining over Data Streams

Mahmood Deypir  
Computer Science and Engineering Department  
School of Engineering, Shiraz University  
Shiraz, Iran  
mdeypir@cse.shirazu.ac.ir

Mohammad Hadi Sadreddini  
Computer Science and Engineering Department  
School of Engineering, Shiraz University  
Shiraz, Iran  
sadeedin@shirazu.ac.ir

Abstract—Sliding window is an interesting model to solve frequent pattern mining problem since it does not need entire history of received transactions and can handle concept change by considering recent data. However, in the previous sliding window algorithms, required amount of memory and processing time with respect to limited number of transactions within window is very large. To overcome this shortcoming, this paper, introduces a new algorithm for dynamic maintaining the set of frequent itemsets over sliding window. By storing required information in a prefix tree, the algorithm does not require to store sliding window transactions. Moreover, it exploits an effective traversal strategy for the prefix tree and suitable representation for each incoming batch of transactions. Experimental results show the superiority of the proposed algorithm with respect to previous methods.

Keywords-component; frequent itemset mining; data stream; sliding window

I. INTRODUCTION

A data stream represents an input data that arrives at a rapid rate and is infinite. Examples for the sources of data streams include customer click streams, network monitoring data, telephone record calls, large sets of web pages, sensor network, scientific data, retail chain transactions and etc. Due to massive amount of incoming data in data stream mining, data elements must be only scanned once using a limited amount of main memory during the mining process [1]. Among all functions of data mining, frequent itemset mining [2] over data streams attracts a great deal of interest in data mining community since it can reveal valuable knowledge. Sliding window is an interesting model for frequent itemset mining over data streams [3-10]. In this model, only recently arrived transactions are considered for the mining purpose. Sliding window model handles concept drift within an input data stream by considering only recent transactions. Moreover, it needs limited amount of memory and processing power. In this study, a novel algorithm is proposed to adaptively maintain the set of frequent itemsets over sliding window. This algorithm not only updates the current set of frequent itemsets but also adapts itself to recent changes of the input data stream. That is, it can identify newly appeared patterns and remove infrequent ones.

The rest of the paper is organized as follows. In the next section some previous related works are reviewed. Section 3 introduces the proposed algorithm. Some experimental evaluations of the proposed algorithm and comparison with a previous method are presented in Section 4. Finally, Section 5 concludes the paper.

II. RELATED WORKS

Let \( I = \{i_1, i_2, \ldots, i_m\} \) be a set of items. Suppose that \( DS \) be a stream of transactions received in sequential order. Each transaction of \( DS \) is a subset of \( I \). For an itemset \( X \), which is also a subset of \( I \), a transaction \( T \) in \( DS \) is said to contain the itemset \( X \) if \( X \subseteq T \). A transactional sliding window \( W \) over data stream \( DS \) contains \(|W|\) recent transactions in the stream, where \(|W|\) is the size of the window. The window slides forward by inserting a new transaction into the window and deleting the oldest transaction from the window. Due to efficiency issues, instead of a single transaction, the unit of insertion and deletion can be a pane (or batch) of transactions. There are a large number of algorithm operating in the sliding window model to mine set of frequent itemsets over data streams.

\( DSTree \) [3] and \( CPS-Tree \) [4] are two algorithms that use the prefix tree to store raw transactions of sliding window. \( DSTree \) uses a fixed tree structure in canonical order of branches while in \( CPS-Tree \) the prefix tree structure is reconstructed to control the amount of memory usage. Both of [3] and [4] perform the mining task using \( FP-Growth \) [11] algorithm that was proposed for static databases. In [5] an algorithm namely \( MPI-TransSW \) was proposed which is based on the \( Apriori \) algorithm [2]. This algorithm mines all frequent itemsets over recent window of transactions. It uses a bit string for every item to store its occurrence information within the window. All of [3, 4, 5] perform the mining task on the current window when the user requests and do not adaptively maintain and update the mining result. Therefore, after the mining, when new transactions are arrived from the stream, obtained result becomes invalid for the user and thus the mining task need to be re-executed. On the other hand, applying a mining algorithm on the whole of the window needs considerable processing and memory requirements with respect to algorithms which continuously update the mining results [6, 7, 8, 9, 10].
Lin et al. [6] proposed a new method for mining frequent patterns over time sensitive sliding window. In their method the window is divided into a number of batches for which itemset mining is performed separately. The Moment algorithm [7] finds closed frequent itemsets by maintaining a boundary between frequent closed itemset and other itemsets. The SWIM [8] is a pane based algorithm in which frequent itemsets in one pane of the window are considered for further analysis to find frequent itemsets in whole of the window. It keeps the union of frequent patterns of all panes and incrementally updates their supports and prunes infrequent ones. It stores transactions of the window in form of the prefix tree of each pane. In [9] the authors devised an algorithm for mining non-derivable frequent itemsets over data streams. This algorithm continuously maintains non-derivable frequent itemsets of the sliding window. Non-derivable and closed frequent itemsets are special types of frequent itemsets which can be viewed as a summary of all frequent itemsets. Chang and Lee proposed the estWin algorithm [10] that finds recent frequent patterns adaptively over transactional data streams using sliding window model. This algorithm uses a monitoring lattice in the form of a prefix tree to monitor the set of frequent itemsets over a data stream. Each node of the lattice represents an itemset which can be constructed using items stored in the nodes in the path from the root to the node. Last node of this path stores the information related to the itemset, e.g., support. The algorithm uses a reduced minimum support named minimum significant to identify new frequent itemset and better estimate their support. When a new transaction is arrived from the input data stream, current set of frequent itemset is updated by visiting related itemset in the monitoring lattice. Then new significant itemsets are identified by second traversal of the related paths of the monitoring lattice using the transaction. This algorithm stores the set of transactions of the window in the memory to eliminate their effect when they become expired. For removing oldest transaction from the window, related paths of the monitoring lattice must be visited again. An itemset is inserted to the prefix tree when all of its subsets have become significant and are stored in the tree. The algorithm estimates the support of a new significant itemset within previous transactions of the window using its subsets which have smaller length.

III. THE PROPOSED ALGORITHM

Panel based windows were shown to be useful for querying and mining data streams [8, 12]. In this study, a new pane based online algorithm named paWin is proposed which overcome all mentioned problems and has good runtime and memory usage. Additionally, its result is more accurate with respect to transaction based window algorithms like estWin. Our proposed algorithm is composed of two main phases, window initialization and window sliding. The algorithm uses new techniques to enhance both runtime and memory usage. Similar to the estWin method, the paWin uses a prefix tree to store and maintain frequent itemsets of sliding window.

A. Window Initialization

In our algorithm, the incoming data stream is fed into the algorithm using equal sized panes of transactions. Each pane of transaction is stored in form of Tidsets. The Tidset of each item is a set containing the Tids of transactions in which the item has appeared. If a transaction contains an item, its Tid will appear in that item’s Tidset.

After arrival of the first pane of transactions, the set of frequent itemsets of this pane is mined using the minimum support threshold. Eclat [13] algorithm is used for this purpose. The set of frequent itemsets are stored in the prefix tree structure. In the remaining process this prefix tree is used to update the mining result. Each node of prefix tree contains various information which are used during the mining process. In this prefix tree, item id, actual counts of panes of the window (ACs), pane ids (Pids) potential count (PC), id of a pane which causes the itemset to be inserted to the tree (FPids) and pointers to children of the node are important information stored in each node of the tree.

Similar to the estWin algorithm, a reduced minimum support named minimum significant is used for early monitoring of the support of the itemsets. After the first pane, by arrival of subsequent panes, the mining result is updated and new significant itemsets are identified and inserted to the prefix tree. This process continues until the window becomes complete. Subsequently, the window sliding phase starts.

B. Window Sliding

The window sliding phase consists of new pane addition and old pane deletion. A new pane, composed of Tidsets of items, is constructed and updated by arrival of a new transaction. After arrival of a number of transactions equal to the pane size, the set of Tidsets is ready for updating the prefix tree. This process includes updating of old itemset stored in the prefix tree, removing of insignificant itemsets from the tree and also insertion of newly identified significant itemsets to the prefix tree.

The Tidsets of newly arrived pane is used to perform a depth first addition process. In our algorithm conditional panes are used to update prefix tree structure at different depths. A conditional pane can be generated for a node of the prefix tree. Conditional pane of each node contains some Tidsets. Each Tidset contains Tids of transactions that contain both the item and the itemset represented by that node.

Conditional pane of the root node of the prefix tree is the complete set of Tidsets of the newly arrived pane. Since all single items having at least one occurrence are maintained in Tidsets, each Tidset of this pane can be used to update the
nodes of the first depth in the prefix tree. However, a depth first traversal of the prefix tree is used in which conditional panes are generated after visiting and updating each node to process nodes at lower depth. Tidset intersections are used to construct a conditional pane. The process of updating is illustrated using an example of a prefix tree shown in Fig. 1 for its left most branch. For sake of presentation, other branches and details of the nodes in the monitoring tree are not depicted. Moreover, the count values of the nodes after updating are not shown. The full content of the pane in the root of tree is used to update all corresponding branches by constructing conditional pane of each node. For each node of the left most branch, the conditional pane that is used to update its actual count is shown. During the traversal process, for each item that resides in a node, the size of corresponding Tidset is added to the count value of the node. In Fig. 1, for the node containing item “a”, the value of 3 is added to its support and thus its value becomes 6 since the size of Tidset of item “a” in the corresponding conditional pane is 3. Subsequently, before processing its first child, the corresponding conditional pane of that child must be constructed. For this purpose, Tidset of item “a” is intersected by Tidsets of child node’s item and its subsequent Tidsets. Since node containing item “b” is first visited in the second depth, after generating conditional pane for this node, its count is added by the size of Tidset “b” (2) in conditional pane and thus its count becomes 4. This is due to the number of transactions of the complete pane that contains both items “a” and “b”, i.e., count of itemset “ab” is two.

![Figure 1. Updating prefix tree using conditional panes.](image)

For the next depth, other conditional panes are generated using set intersection of Tidset of item “b” and other Tidset. At node c, its count is added by size of c’s Tidset and the new value becomes 4. In fact, the number of transactions that contain “a”, “b” and “c” is two. That is, itemset “abc” has actual count 2 in the complete pane. For node representing itemset “abd”, the conditional pane only contains one Tidset of item “d” whose size is 1. This conditional pane is constructed by intersecting tidssets of “b” and “d”. Therefore, the count of the node “d” at third depth becomes 3. Conditional panes are generated irrespective to existing of the corresponding node. For a branch of the tree, the process continues until an empty conditional pane is constructed or if there is not any child node corresponded to a generated conditional pane. In the second case, remaining items of the conditional pane are tested for possible insertion of new node, i.e. adding a new itemset into the prefix tree. If an itemset represented by the current node and an item remaining in the conditional pane has a significant greater than the minimum significant, a new node corresponded to the item is inserted to the tree. The value of significance is composed of two other values, the actual count of the itemset in the conditional pane and the estimated count using the subsets of the itemset. The estimated count and error in this estimated count are calculated by the approach proposed in [10].

Therefore, for an itemset which corresponds to a node, all of its subsets must be visited first since the estimated support is calculated based on them. On the other hand, due to Apriori principle a new itemset is inserted once all of its subsets become significant and inserted. In order to achieve this goal, we use a reverse order depth first traversal of the prefix tree which is described later. For an existing node, its actual support is updated. For a newly inserted node its actual count is set to the size of its corresponding Tidset in the conditional pane and its potential count is set to the estimated support computed using the subsets of the itemset. Moreover, the Pid of inserted pane is stored in a newly inserted node as its FPid. This value is used to update potential count of the itemset in the pane removal process. Furthermore the Pid is also appended in the array of the Pids for both updated and inserted node. Values stored in this array are used to update the actual count of the node again in the pane removal process.

Another task performed by pane addition process is pruning of insignificant itemsets. Although, the support of an itemset increases by the updating process, however, its significant might be lower than minimum significant since it is a relative value with respect to the window size. Therefore, such nodes should be erased from the prefix tree. Additionally, based on the Apriori property, all of its descendents are also removed from the prefix tree and it is not required to visit them during the pane addition process.

As mentioned previously, updated subsets of an itemset is required for adding the corresponding node. Therefore, the pane addition process must visit all subset of an itemset before the itemset itself. In an ordered prefix tree, some subsets of an itemset are placed after the itemset itself. That is, if we perform a depth first traversal, some of subsets resided in nodes of the tree are visited after the node containing the itemset. The estWin algorithm overcomes this shortcoming, by traversing the prefix tree twice, once for updating of the itemsets and the other for insertion process. However, this violates the single processing of elements rule and diminishes the performance. Therefore, we propose a new traversal strategy of the prefix tree during the pane addition process for both nodes' update and new nodes insertion processes by only one traversal. In this strategy, the prefix tree is traversed in a depth first reverse order manner. In this way, for an itemset, all of its subsets are visited first.
Therefore, for estimating the support of a new significant itemset, all updated support values of its subsets are available. Moreover, some subsets of the itemsets might be inserted to the prefix tree. This facilitates early insertion of these itemsets. For support estimation of a k-itemset as described in [10] only (k-1)-itemsets are required which are obviously visited before the itemset. Therefore, by using this strategy both updating and insertion are performed in one traversal of the prefix tree using the new pane. It is important to note that, those nodes in the monitoring prefix tree are visited that related itemset are included in the newly inserted pane and whole of the tree does not require to be visited.

After new pane addition, a removal process is required to delete oldest pane of the window. In this way, fixed size window is preserved. In contrast to the Addition process, deletion does not require using the oldest pane of transactions and the action is performed only on the prefix tree using stored pane information. When the window advances, the oldest pane of the window should be removed. As mentioned previously, when a pane causes a node to be inserted to the tree, its Pid and Pids of subsequent panes, when they will arrive, are stored in the node. In another array (ACs), for each of these panes, corresponding actual counts are also stored. Due to the stored pane information, it is not required to store transactions of the window in order to remove their effects when they become obsolete. For oldest pane, its contribution should be removed from the set of significant itemset represented as the nodes of the prefix tree. Removing the oldest pane from the window is performed by visiting the nodes of the prefix tree. For each node of the tree if the list of Pids contains the oldest pane id, it is removed. Moreover, the actual count of the pane is removed from ACs list and it is also subtracted from the actual count of the itemset in the window (AC).

On the other hand if the list of Pids does not contain the removed pane id and FPid of the node is greater than the removed Pid, only the value of potential count is corrected. The PC is replaced by the value of formula 1, if the value is smaller than the current PC value.

\[(FPid - Wid) \times S_{sig}\] (1)

Where FPid and Wid are the Pid causes the itemset to be inserted to the tree and Pid of the first pane of the window, respectively. If the FPid of the node becomes equal to the Wid due to the pane removal process, the value of the error (ER) and potential count (PC) of the node become zero since we have the actual count of the itemset in whole window. Another task performed during the pane deletion process is erasing the nodes which become insignificant. For each insignificant node, all descendants are also removed since they belong to supersets of its itemset. Similarly, this is performed according to the A priori property since any superset of an insignificant itemset is also insignificant.

IV. EXPERIMENTAL RESULTS

In this section, the proposed algorithm is experimentally evaluated and compared to the estWin and SWIM algorithms. These algorithms are selected since the first algorithm is an approximate algorithm and second is a pane based method. We have implemented estWin, SWIM and the paWin algorithms using C++ language and Dev C++ IDE. All experiments are executed on an Intel Pentium 4 CPU 3.0-GHz machine with 2-GB RAM running on Windows XP.

Since the memory and processing time are two main factors of every data stream mining algorithm, we compare the algorithms in terms of these factors. Similar to the estWin algorithm, our algorithm is an approximate algorithm. That is, it might miss some frequent itemsets or identify some infrequent patterns as frequent. Therefore, in the last experiment, the accuracy of the result is evaluated and compared to the estWin method in terms of false negatives and false positives. We select a synthetically generated dataset [2] T40I10D100K to accomplish empirical evaluations. This dataset contains 100,000 transactions and 1000 items. The average transaction length is 40.

A. Runtime

The parameter that impacts the runtime of all three algorithms is minimum support threshold. In this experiment, window size is set to 40K. Pane size is set to 10K for paWin and SWIM. Based on the user interest, other values for these parameters are allowed. Force pruning is set to 10K for paWin and estWin and vary the value of minimum support threshold for all algorithms to see the impact of minimum support threshold on the runtime. The results are shown in Fig. 2. Vertical and horizontal axises show runtime and minimum support respectively.

As shown in Fig. 2, the proposed algorithm is faster than the estWin and SWIM algorithms. The performance gain of our algorithm is considerable for all values of minimum support thresholds. This is due to batch processing of transactions and also performing fast removal of old information belonging to the removed pane from the set of frequent patterns. Moreover, in paWin both of the pane removal and pruning of insignificant itemsets are performed simultaneously while the estWin algorithm has a separate phase of pruning named force pruning which is performed periodically. Furthermore, by using the reverse order depth first traversal, both updating of current itemset and insertion
of new patterns are performed simultaneously. On the other hand, in the estWin algorithm, related paths for every incoming transaction are visited twice, one for itemset updating and the other for possible itemset insertion to the tree. In our algorithm, by arrival of a pane, new itemset identification and support updating for previously found frequent itemsets are performed together while the SWIM algorithm applies the FP-Growth algorithm on new pane to detect frequent itemset of the pane and then separately verifies the support of previous frequent itemsets. Our algorithm computes the support of new itemset within new pane and estimate the support on previous pane of the algorithm computes the support of new itemset within new pane and then separately verifies the support of previous frequent itemsets. Our algorithm computes the support of new itemset within new pane and estimate the support on previous pane of the window while in SWIM, the support of new itemset is verified in all panes of the window which is a time consuming process.

B. Memory Usage

In this subsection the memory usage of all three algorithms are compared together. Memory required by the estWin algorithm is composed of memory of the prefix tree and required memory to store raw transactions of windows. In SWIM, FP-Trees for transactions of each pane, and prefix tree to store union of frequent itemsets of all panes are required data structures. On the other hand, the paWin algorithm only maintains the prefix tree structure for itemsets and does not physically store transactions of the window neither in the form of FP-Tree nor in the form of raw transactions. Therefore, paWin should have lowest memory usage. For different value of minimum supports and fixed minimum significant of 0.5 (for estWin and paWin), average memory required for all active windows are computed and the results are plotted in Fig. 6. This figure shows that the paWin has lowest memory requirement.

Although, the paWin stores extra information in each prefix tree node (including pane id and support information of each pane), it does not maintain transactions of the window. This is the reason of memory usage superiority of the paWin algorithm with respect to the estWin and SWIM.

C. Accuracy

Since the new algorithm produces an approximate result for an input data stream, the quality of the result must be measured. The number of false positives and false negatives are important for approximate frequent itemset mining algorithms. False positives waste memory and processing time. On the other hand false negatives reduce the quality of the mining results. Vast numbers of false positives and negatives cause the results to be undependable. We report these numbers for three different minimum support thresholds and fixed minimum significance 0.5 for estWin and paWin algorithms. The SWIM is an exact algorithm and does not have any false positive or negative. Total number of false positives and false negatives are enumerated and reported in Table 1. As shown in this table, for all minimum support values, our algorithm has smaller number of false positives and false negatives. Our algorithm does not have any false negative and the number of false positives for each minimum support value is negligible.

<table>
<thead>
<tr>
<th>MinSup(%)</th>
<th>FP</th>
<th>FN</th>
<th>FP</th>
<th>FN</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>19</td>
<td>260719</td>
<td>9</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>5820</td>
<td>25</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>916</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Therefore, the devised algorithm has better accuracy and its mining result is more dependable for the user. The reason of this supremacy is that in our algorithm a new candidate itemset is evaluated, for possible insertion to the prefix tree, based on their actual support in the new pane and estimated support in the previous panes of the window. The new pane constitutes a considerable number of transactions of the window. On the other hand, in the estWin algorithm, actual support of a new itemset restricted to the newly arrived transaction and its support is mainly computed based on estimation using old transactions of the window. In our algorithm, having a wider range of actual support for an itemset, the computed support becomes more realistic and the probabilities of being false positive or false negative are decreased.

V. CONCLUSION

In this study, an efficient algorithm for mining set of frequent itemsets over data streams was proposed. Although, this algorithm operates under sliding window model, it does not store sliding window transaction. Moreover, by batch processing of transactions and using new techniques it becomes a fast algorithm which is suitable for high speed data streams. These techniques facilitate updating of the frequent itemsets by new transactions and removing the effect of old transactions from the mining results. In this algorithm, always the mining result of the active window is available for the user and is continuously updated. Experimental evaluations show that, this algorithm has remarkably better runtime and memory usage with respect
to previously proposed estWin and SWIM methods. Moreover, it has better mining result quality with respect to the estWin as an approximate algorithm.

REFERENCES


[11] J. Han, J. Pei, Y. Yin, and R. Mao, “Mining Frequent Patterns without Candidate Generation: A Frequent-Pattern Tree Approach”, Data Mining and Knowledge Discovery, Vol. 8, 2004, pp. 53-87.
