An Obstruction-Check Approach to Mining Closed Sequential Patterns in Data Streams

Ye-In Chang, Chia-En Li, and Tzu-Lin Chin

Department of Computer Science and Engineering, National Sun Yat-Sen University, Kaohsiung, Taiwan, R.O.C.
changyi@cse.nsysu.edu.tw, {lilce, chintl@db.cse.nsysu.edu.tw}

Abstract: Online mining sequential patterns over data streams is an important problem in data mining. There are many applications of using sequential patterns in data streams, such as market analysis, network security and web tracking. For solving the problem of data mining in the time-based sliding window, Chang et al. proposed an algorithm called SeqStream. The SeqStream algorithm still scans the sliding window many times when IST (Inverse Closed Sequence Tree) needs to be updated. In this paper, we propose an obstruction-check approach to maintain the result of closed sequential patterns. Our approach is designed based on the lattice structure. Based on the lattice structure, we propose the EULB (Exact Update based on Lattice structure with Bit stream)-Lattice algorithm that is an exact method for mining data streams. We record additional information, instead of scanning the entire sliding window. The simulation results show that the proposed algorithm outperforms the SeqStream algorithm.

Keywords: closed sequential pattern, data stream, lattice, sequential pattern, sliding window

1 Introduction

In recent years, the varied data around us become more complexity and huge with the computers and information industries growing more and more rapidly. Commercial behavior, scientific statistics, natural phenomena and DNA projects are some examples to produce lots of data every day. Generally speaking, there are four kinds of problems that we have met. First, there are too much information for us to digest. Second, it is difficult to recognize the reality of the information. Third, it is hard to guarantee the security of these information. Finally, we can not deal with different forms of information easily because of much data. In order to solve these problems, people start to think how to find useful knowledge without overwhelming by information flooding. Recently, the data mining community has focused on a new challenging model, where data arrives sequentially in the form of continuous rapid streams. To support study the model, the knowledge of mining sequential patterns and mining data streams is needed.

The problem of mining sequential patterns was first introduced in [1]. The input data is a set of sequences, call datasequences. Each datasequence is a list of transactions,
where each transaction is a set of literals, called items. Each transaction is associated
with a transaction time. A sequential pattern consists of a list of sets of items. The
problem is to find all sequential patterns with a user-specified minimum support,
where the support of a sequential pattern is the percentage of data sequences that con-
tain the pattern [11]. A data stream is an ordered sequence of items that arrives in
timely order. Different from data in traditional static databases, data streams are con-
tinuous, unbounded, usually come with high speed and have a data distribution that
often changes with time [8]. There are large amounts of data streams in many appli-
cations, including network monitors, traffic monitors, ATM transaction records in
banks, sensor network monitor, web logs and web click streams, and transactions in
retail chains. Mining data in such applications is referred to as stream mining. Stream
data mining adds many complexities to traditional mining requirements, which are:
(1) the volume of continuously arriving data is massive and cannot all be stored and
scanned for mining; (2) there is insufficient memory; (3) the mining algorithm does
not have the opportunity to scan the entire original dataset more than once, as the
whole dataset is not stored; (4) a method for delivering considerably accurate result
on demand is needed; (5) in order to mine sequential patterns in streams like click
stream data, there is need to keep Customer Access Sequences (CAS) in the order
they arrive. CAS is the sequential buying or product viewing order by a customer,
e.g., (TV, radio, jean pants, TV, shirt, TV). Keeping CAS order intact for each trans-
action and mining frequent sequential patterns from them presents additional com-
plexities [5].

The rest of the paper is organized as follows. Section 2 gives a survey of several
well-known techniques for mining sequential patterns problem, including mining
sequential patterns, mining closed mining sequential patterns in data streams. Section
3 presents the proposed the EULB-Lattice algorithm. In Section 4 we study the per-
formance and make a comparison of the proposed algorithm. Finally, we give a con-
clusion.

2 Related Work

For frequent sequential pattern mining, efficient algorithms, such as GSP
[11], FreeSpan [6], PrefixSpan [10] were developed. IncSPAM [7] is a stream sli-
dingwindow algorithm for mining all sequential patterns. It is based on SPAM [2]
which uses a vertical format representation of sequences. Like SPAM, IncSPAM suf-
fers from huge memory consumption. IncSpan [4] which is designed to maintain the
complete set of sequential patterns upon incremental update. There are some well-
known data mining algorithms for mining sequential patterns, including mining se-
quential patterns in traditional databases [1, 6, 10], mining sequential patterns with
high utility [9] and mining sequential patterns in data streams [3].

Discovery of mining closed sequential patterns is an important problem in the area
of data mining. A sequential pattern is closed if the sequential pattern does not have
any supersequence which has the same support. Therefore, the result of closed sequen-
tial pattern is more compact than the result of sequential pattern. Previous algorithms
to mine closed sequential patterns in data streams have been based on many different models. In [3], Chang et al. proposed a time-based sliding window model. The time-based sliding window model is constructed by data elements, where a data element records user’s transaction at this time. For solving the problem of data mining in the time-based sliding window, Chang et al. proposed an algorithm called SeqStream. It uses a data structure IST (Inverse Closed Sequence Tree) to keep the result. IST can incrementally be updated by the SeqStream algorithm. The SeqStream algorithm adopts DFS (Depth-First Search) strategy, it travels nodes in IST based on subtree of each node, recursively. Although the SeqStream algorithm has used the technique of dividing the time-based sliding window to speed up the updating of IST, it still has two disadvantages: (1) the cost of scanning the entire sliding window is too expensive, and it also cannot achieve in one pass which does not meet the requirement of data streams; (2) the SeqStream algorithm will create redundant nodes.

3 The Obstruction-Check Approach

In this section, we present an algorithm called EULB-Lattice algorithm, it is designed based on the obstruction-check approach to solve the mining closed sequential patterns problem in data streams based on the time-based sliding window model. The sliding window is one of the window models, it only retains recent data at the current time. The sliding window size is fixed. Therefore, the oldest data will be removed from the sliding window, when the sliding window is full and the new data incomes. For example, a sequence <BCDA> in the time-based sliding window with window size 4. An obsolete data element <A> is removed, and the sequence is changed to <BCD>. A new data element <B> incomes and the sequence is changed to <BBCD>. Therefore, we can use this feature to develop our algorithms.

3.1 An Extended Lattice Structure

We use an extended lattice structure to keep the result of mining sequential patterns. The lattice structure is a graph-based structure. The feature of the structure is that parent nodes are the supersequence of the child node and child nodes are subsequence of the parent node. For example, given min sup = 2 for Figure 1-(a), sequential patterns are {<A>:6, <B>:5, <C>:4, <D>:3, <E>:3, <AB>:5, <AC>:2, <AD>:3, <AE>:3, <BD>:3, <BE>:2, <CA>:2, <CB>:2, <ABD>:2, <CAB>:2} before data element e11 incomes. The result is shown in Figure 1-(b).

We introduce a virtual node which is called Base, it connects all 1-sequence sequential patterns. In this lattice structure, it has two connection relationships: (1) Dotted line: the line represents that the support of the parent node is the same as the support of the child node. For example, in Figure 1-(b), the support of node <AB> is the same as the support of its child node <B>. We say that the connection relationship is an obstruction between node <AB> and node <B>, we also call it an obstruction link. (2) Solid line: the line represents that the support of the parent node is smaller than the support of the child node. For example, in Figure 1-(b), the support of node <B>

---

518
AB > is smaller than the support of its child node < A >, we say that the connection relationship is a non-obstruction between node < AB > and node < A >. We also call it a non-obstruction link. If it is obstruction link between the parent node and the child node, we call the parent node as obstruction parent and call the child node as obstruction child. Otherwise, we call the parent node as non-obstruction parent and call the child node as non-obstruction child. In Figure 1-(b), node label represents a sequence of the node, and we utilize a number to indicate the support of a node. We define a closed sequential pattern as that the node in the lattice structure does not have any obstruction parent. In other words, all parents of the node are non-obstruction parent. For example, node < A > does not have any obstruction link; therefore, node < A > is a closed sequential pattern.

![Figure 1](image-url)

**3.2 The Update Strategy**

Based on the new definition of the time-based sliding window, the data elements are updated at the same sequence. Different from the tradition stream data window, based on the new definition in time-based the sliding window; the change of the support of a node may change its children and parents. Because the change appear based on a same sequence. For example, in Figure 1-(b), we assume that there are two sequence {< CA >, < A >} in the sliding window, then there are two node in the lattice structure. If a new data element {< C >} comes, the sliding window is changed to {< CA >, < CA >}. The support of node < CA > is changed from 1 to 2, but its child node < A > does not change the support. Because the support of node < CA > is increased, the connection < A > - < CA > is changed from non-obstruction to obstruction.

We observe four cases for this feature, we describe it by Figure 2. In Case 1, when the support of a node increases and it has an obstruction link with its parent node, the connection link of the parent node may be changed to a non-obstruction link. In Case 2, when the support of a node increases and it has a non-obstruction link with its child
node, the connection link of the child node may be changed to an obstruction link. In Case 3, when the support of a node decreases and it has an obstruction link with its child node, the connection link of the child node may be changed to a non-obstruction link. In Case 4, when the support of a node decreases and it has a non-obstruction link with its parent node, the connection link of the parent nodes may be changed to an obstruction link.

We summarize those features. When the support of a node increases, we only care its obstruction parents and its non-obstruction children. When the support of a node decreases that we only care its non-obstruction parents and its obstruction children. We utilize those principles to update our lattice structure locally.

![Fig. 2. An example of four cases in data change: (a) Case 1; (b) Case 2; (c) Case 3; (d) Case 4.](image)

3.3 The EULB-Lattice Algorithm

In this section, we utilize an exact method, the EULB (Exact Update based on Lattice structure with Bit stream)-Lattice algorithm, to update the lattice structure. Our algorithm needs to expand the lattice structure for dynamically update. We embed a user table into each node. The user table is shown in Figure 3, it records two kinds of information: (1) User ID and (2) Bit Stream. We can use a simple way to construct the extended lattice structure. Since the movement of data is too quick, we only care how to efficiently update it.

In the EULB-Lattice algorithm, we utilize the lattice structure to keep results. We use a bit stream to represent the item appearing in a sequence, and the size of a bitstream is the same as the size of the sliding window. This concept is originated in the count of the tail item. A tail item is the last item of a sequence, and we define the count of the tail item which is the number of the last item appearing in a sequence. Let's see a simple example, given a user sequence, $us = \langle ABABCDBC \rangle$ in the sliding window at time $t_5$, and given a sequence, $ns = \langle ABC \rangle$. The count of the tail item of sequence $ns$ in sequence $us$ is 2. Because sequence $ns$ appears in sequences $us$ from time $t_5$ to time $t_9$, and the tail item of sequence $ns$ appears twice before time $t_5$. If the obsolete data item of sequence $us$ is removed, sequence $us$ is changed to $\langle ABABCDB \rangle$, and the count of the tail item of sequence $ns$ in sequence $us$ is de-
creased. Sequence $ns$ still exists in sequence $us$. If sequence $us$ is changed to $< ABAB >$, the count of the tail item of sequence $ns$ in sequence $us$ is 0. Sequence $ns$ does not exist in sequence $us$.

![Fig. 3. An extended lattice structure](image)

Procedure EULB-Lattice algorithm is shown in Figure 4. The variables used in our algorithm are listed in Table 1. In this structure, parents are supersequence of the child. We can update the current level based on the previous level. When the child is not frequent, its parents are also not frequent. Let level $n$ represent the $n$th level in lattice structure, level 0 only has virtual node base, level 1 has 1-sequence node $< A >, < B >, etc.$, level 2 has 2-sequence node $< AB >, < AC >, ...$, and so on.

![Fig. 4. The EULB-Lattice algorithm](image)

```
1: procedure EULB(D, min_sup, base, currentT);
2: Input: (1) the current sliding window $D$
3: (2) the minimum support min_sup
4: (3) the base node of the extended lattice structure base
5: (4) the current time currentT
6: Output: An updated extended lattice structure
7: begin
8: \* the data preprocessing phase \*
9: create INSTravelers based on $D$;
10: create RMTravelers based on $D$;
11: create nextRMNs based on RMTravelers and base;
12: \* the update phase \*
13: while (nextRMNs or nextINSNs or nextCANs is not empty) do
14: begin
15: currentRMNs := nextRMNs;
16: currentINSNs := nextINSNs;
17: \* bit stream update step \*
18: delete_step(RMTravelers, currentRMNs, nextRMNs, currentT);
19: insert_step(INStravelers, currentINSNs, nextINSNs, currentT);
20: \* node state update step \*
21: state_update_step(min_sup);
22: node_create_step(min_sup, currentT);
23: end;
```
**The data preprocessing phase:** In this phase, we collect the new data element and the obsolete data element to a set, we call it Traveler. The traveler is constructed with an item label and a set of user ID, it represents users which are contributed by the same item at the same time, and it is denoted by item traveler. For example, the new data element incomes at time $t_{11}$. There are three item elements $\{<1, C>, <5, E>, <6, C>\}$ in the new data element. We collect them to $\{<1, C>, <5, E>, <6, C>\}$.

**The update phase:** In this phase, we update the lattice structure by travelers which are constructed by the data preprocessing phase. The update phase has two main steps: (1) the bit stream update step: In this step, we update the bit stream of user nodes and the support of nodes on the current level; (2) the node state update step: In this step, we update connections of nodes and delete the node from the lattice structure, when the support of the node is smaller than $\minsup$.

At line 13, we must check whether a set of nodes is not empty. If we can create any set of nodes for travelers, we execute this procedure to update the lattice structure. Otherwise, we stop the procedure. We give an example based on Figure 1-(a), the new data element $e_{11}$ incomes and the obsolete data element $e_{3}$ is removed from the sliding window. We can create a set of travelers based on element $e_{11}$ elements $\{<1, C>, <5, E>, <6, C>\}$ and a set of travelers based on element $e_{3}$ elements $\{<D: 1, 3>, <C: 4>\}$. We call them $\text{INSTTravelers}$ and $\text{RMTravelers}$, respectively.

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$D$</td>
<td>The current sliding window database</td>
</tr>
<tr>
<td>$\minsup$</td>
<td>The minimum support</td>
</tr>
<tr>
<td>$\text{base}$</td>
<td>The base node of the extended lattice structure</td>
</tr>
<tr>
<td>$\text{INSTTravelers}$</td>
<td>A set of travelers which are created based on the new data element of $D$</td>
</tr>
<tr>
<td>$\text{RMTravelers}$</td>
<td>A set of travelers which are created based on the obsolete data element of $D$</td>
</tr>
<tr>
<td>$\text{nextINSNs}$</td>
<td>A set of nodes which are created based on $\text{INSTTravelers}$ and $\text{currentINSNs}$</td>
</tr>
<tr>
<td>$\text{nextRMNs}$</td>
<td>A set of nodes which are created based on $\text{RMTravelers}$ and $\text{currentRMNs}$</td>
</tr>
<tr>
<td>$\text{currentINSNs}$</td>
<td>A set of nodes which need to be increased on the current level</td>
</tr>
<tr>
<td>$\text{currentRMNs}$</td>
<td>A set of nodes which need to be decreased on the current level</td>
</tr>
<tr>
<td>$\text{NS}$</td>
<td>A set of nodes which are created based on each traveler</td>
</tr>
<tr>
<td>$\text{cflag}$</td>
<td>A flag which is to record the node state</td>
</tr>
<tr>
<td>$\text{oldSup}$</td>
<td>The initial support of the node</td>
</tr>
<tr>
<td>$\text{US}$</td>
<td>A set of users that each user in the set exists in the traveler and the node</td>
</tr>
<tr>
<td>$m$</td>
<td>A node which needs to be updated</td>
</tr>
<tr>
<td>$\text{modifyNS}$</td>
<td>A set of nodes that the support of some nodes in the set are changed</td>
</tr>
<tr>
<td>$\text{om}$</td>
<td>A node that we can find it in $\text{modifyNS}$ and it is the same as node $m$</td>
</tr>
<tr>
<td>$\text{currentT}$</td>
<td>The current timestamp</td>
</tr>
</tbody>
</table>

We adopt BFS (Breadth-First Search) strategy in order to utilize the feature of the lattice structure. It can avoid redundant update. Therefore, if the node is not frequent, which represents that we do not need to update its ancestors, we can delete the node and its ancestors from the lattice structure. When a node on the current level has been updated, we will find its parents which need to be updated on the next level.
el, nextRMNs, nextINSNs and nextCANs are filled based on procedures delete step, insert step, and node create step, respectively. The lattice structure has been updated at the current time.

4 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 SeqStream algorithm and our proposal algorithm. All experiments were done on the Core 2 Duo processors clocked at 2 GHz. The server has 2GB physical memory and runs Microsoft Windows XP SP2 and we use Java to implement the SeqStream algorithm and our algorithm. The program is compiled by JDK 1.6.

4.1 The Simulation Model

The synthetic data sets is produced by referring the time-based sliding window model [3]. These synthetic transaction data sets are used to evaluate the performance of algorithms for mining closed sequential patterns. The parameters used in the generation of the synthetic data are shown in Table 2. The length of a sequence is not larger than the size of the sliding window. An item in data element denotes \{UID, item\}, the data element size \(S\) is not larger than the number of users \(D\). If \(S = 50\) and \(S < D\), it denotes that a data element contains at most 50 items in the current time. We test the EULB-lattice algorithm and the SeqStream algorithm with the minimum support threshold \(MS\). If the number of users \(D\) is 100 (the maximum number of sequences is 100), the minimum support threshold \(MS\) is 30%. Then the minimum support value is 100*0.3.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>(T)</td>
<td>The number of data elements</td>
</tr>
<tr>
<td>(D)</td>
<td>The number of users</td>
</tr>
<tr>
<td>(C)</td>
<td>The maximum length of a items</td>
</tr>
<tr>
<td>(N)</td>
<td>The number of items</td>
</tr>
<tr>
<td>(S)</td>
<td>The size of data elements</td>
</tr>
<tr>
<td>(MS)</td>
<td>The minimum support threshold</td>
</tr>
<tr>
<td>(ES)</td>
<td>The maximum error support threshold</td>
</tr>
<tr>
<td>(SW)</td>
<td>The size of the sliding window</td>
</tr>
</tbody>
</table>

4.2 Experiment Results

We show the experiment result of the EULB-Lattice algorithm and the SeqStream algorithm. In Figure 5-(a), we compare the processing time with different minimum supports \(MS\) which is changed from 28% to 20%. The synthetic data set is \(T10kD50C20N500S30\) with the windowsize \(SW\) 20. The experiment results shows that the processing time of the EULB-Lattice algorithm is faster than the processing
time of the SeqStream algorithm. Figure 5-(b) shows the comparison of the processing time of the window sliding with different window size $SW$. The synthetic data set $T_{10k}D_{100}C_{30}N_{500}S_{50}$ with the minimum support $MS$ 23% is used. The window size $SW$ is changed from 20 to 30. The experiment results shows that the EULB-Lattice algorithm is faster than the SeqStream algorithm.

![Fig. 5. A comparison: (a) a comparison of the processing time under different minimum supports; (b) a comparison of the processing time under different sliding window size.](image)

Figure 6-(a) shows the comparison of the processing time of the window sliding with different number of data elements $T$. The synthetic data set $T_{10k}D_{100}C_{30}N_{500}S_{80}$ with the minimum support $MS$ 30% is used, and the window size $SW$ is 20. The experiment results shows that the growth rate of processing time of the EULB-Lattice algorithm is smaller than the growth rate of the processing time of the SeqStream algorithm. Figure 6-(b) shows the comparison of the processing time of the window sliding with different number of users $D$. The synthetic data set $T_{10k}D_{[200–600]}C_{30}N_{500}S_{100}$ with the minimum support $MS$ 30% is used, and the window size $SW$ is 20. The experiment results shows that the effect of number of users for executing algorithm is small.

![Fig. 6. A comparison: (a) a comparison of the processing time under different number of data elements; (b) a comparison of the processing time under different number of users.](image)
5 Conclusion

In this paper, we have proposed the EULB-Lattice algorithm, and made a comparison with the SeqStream algorithm by simulation. The EULB-Lattice algorithm is an efficient algorithm for the problems of mining closed sequential patterns in data streams. We have conducted several experiments using different synthetic data. The simulation results have shown that the proposed EULB-Lattice algorithm outperforms the SeqStream algorithm in all relational data settings.

Acknowledgments

The 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