Incremental Mining of Across-streams Sequential Patterns in Multiple Data Streams

Shih-Yang Yang
Department of Media Art, Kang-Ning Junior College of Medical Care and Management, Taipei, Taiwan 114, R.O.C.
Email：shihyang@knjc.edu.tw

Ching-Ming Chao
Department of Computer Science and Information Management, Soochow University, Taipei, Taiwan 100, R.O.C.
Email：chao@csim.scu.edu.tw

Po-Zung Chen and Chu-Hao Sun
Department of Computer Science and Information Engineering Tamkang University, Tamsui, Taiwan 25137, R.O.C.
Email：pozung@mail.tku.edu.tw, 894190320@s94.tku.edu.tw

***Abstract*—Sequential pattern mining is the mining of data sequences for frequent sequential patterns with time sequence, which has a wide application. Data streams are streams of data that arrive at high speed. Due to the limitation of memory capacity and the need of real-time mining, the results of mining need to be updated in real time. Multiple data streams are the simultaneous arrival of a plurality of data streams, for which a much larger amount of data needs to be processed. Due to the inapplicability of traditional sequential pattern mining techniques, sequential pattern mining in multiple data streams has become an important research issue. Previous research can only handle a single item at a time and hence is incapable of coping with the changing environment of multiple data streams. In this paper, therefore, we propose the IAspam algorithm that not only can handle a set of items at a time but also can incrementally mine across-streams sequential patterns. In the process, stream data are converted into bitmap representation for mining. Experimental results show that the IAspam algorithm is effective in execution time when processing large amounts of stream data.**

***Index Terms*— Multiple data streams, Data stream mining, Sequential pattern mining, Incremental mining**

I. INTRODUCTION

In the era of knowledge economy, the creation, communication and application of knowledge are the driving force behind the growth of industry. As a result, knowledge has become one of indispensable requirements to success. Therefore, how to acquire valuable knowledge is very important. Data mining is to discover useful knowledge from a great amount of data. Sequential pattern mining is to find out sequential patterns that frequently happen with time or specific sequence. For example, in terms of web pages access, the access patterns of web surfers can be explored to predict the next possibly accessed web page for advance access to expedite the web access.

There are many user behaviors in our daily life that can generate stream data such as the signal data of mobile communication, stock transaction data, web pages browsing records, etc. These stream data have the following characteristics [1]: (1) unlimited data input, (2) limited memory capacity, (3) one-time processing of input data, (4) fast data arrival, (5) inability of system to halt data inflow. As the characteristics of data streams are different from those of traditional databases, database mining algorithms are inapplicable in the data stream environment. Data streams can be classified into single data stream and multiple data streams. Sequential pattern mining in multiple data streams is an important research issue.

With the change of environment in multiple data streams, data will not only grow larger and larger but also get more and more complicated. Existing algorithms can only handle a single item at a time and hence are incapable of coping with the changing environment of multiple data streams. Besides, the sequential patterns mined may cross different data streams but are regarded as existing in a single data stream. To solve these problems, in this paper we propose the IAspam (Incremental Across-streams Sequential Pattern Mining) algorithm to cope with more complicated multiple data streams environment. IAspam not only can handle a set of items at a time but also can incrementally mine across-streams sequential patterns.

In the process of mining sequential patterns in multiple data streams, the efficiency in calculating the support and searching for frequent sequential patterns must be taken into account. To improve efficiency, we use a bitmap representation, in which each item is converted into one bit, to reduce execution time.

The remainder of this paper is organized as follows. Section 2 reviews related work. Section 3 presents the ASPAMDAS method, which covers data sampling and the IAspam algorithm. Section 4 evaluates and compares the performance of the IAspam algorithm. Section 5 concludes this paper.

II. RELATED WORK

© 2011 ACADEMY PUBLISHER
doi:10.4304/jcp.6.3.449-457

Due to the advent of data stream environment, traditional data mining algorithms are not applicable. Consequently, many researches were conducted on modifying algorithms to be applicable to data stream environment, e.g. mining frequent itemsets, sequential patterns, maximal sequential patterns, and closed sequential patterns in data stream environment. On the other hand, there are also many researches on the types and characteristics of stream data. The MSDD (Multi-Stream Dependency Detection) algorithm proposed by Oates and Cohen [2] aims to find out the dependency rules among stream data in a multiple data streams framework. Whereas, Golab and Ozsu [3] explore the characteristics, models, and query semantics of data stream as well as explain the application of data stream in real life. After the presentation of data stream framework, there are many researches on mining sequential patterns in data stream environment.

In data stream environment, incremental mining retains previous mining results. However, if not updated for a long time, previous mining results might become incorrect, for which, many studies put forward decay mechanisms for data stream mining to prevent data from becoming obsolete and save memory space. Chang and Lee [4] proposed a decay mechanism for data stream environment, using (1) to decay the mining results not updated for a long time to ensure correct mining results. The "d" in the equation stands for "decay rate", "b" for "decay-base", and "h" for "decay-base-life".

$$d = b^{-(1/h)} \quad (b > 1, h \geqq 1, b \leqq d < 1) \qquad (1)$$

The MILE algorithm proposed by Chen et al. [5] is based on the prefix-projection concept of PrefixSpan. It is used for handling multiple data streams in a time period and mining sequential patterns in time sequence data stream environment. It identifies the prefix subsequence and suffix subsequence in stream data to reduce the generation of surplus candidate sequences and combine the sequential patterns with the same prefix. The main purpose is to avoid repeated data searching and speed up the mining of new sequential patterns. However, as MILE is a one-time fashioned algorithm that cannot retain previous mining results, it will take more time in re-mining. IncSPAM mentioned in the previous subsection is a sequential pattern mining algorithm for single data stream environment. It adopts a sliding window to sample stream data and incrementally mines sequential patterns. As sequential patterns will be outdated if not updated for a long time, it uses the weight to decay the support. This concept is based on the decay mechanism proposed in [6], which uses (1) to decay the support of sequential patterns.

Later the SPEED (Sequential Patterns Efficient Extraction in Data Streams) algorithm was proposed by Raissi et al. [7] for mining maximal sequential patterns in data stream. It uses a new data structure to maintain frequent sequential patterns and a fast pruning strategy to allow users to locate at any time the maximal sequential patterns in arbitrary time intervals. The SSM algorithm proposed by Ezeife and Monwar [8] differs from those mentioned above. It uses a D-list structure to effectively store and maintain the support of all items in data stream environment. Meanwhile it continuously builds a PLWAP tree to effectively mine in batches sequential patterns in data stream, after which an FSP-tree is used to incrementally maintain the mining results. Also, there are some algorithms that use graphs for mining sequential patterns. The GraSeq algorithm proposed by Li and Chen [9] adopts a directed weighted graph structure to store the synopsis of sequences and only needs to scan the data stream once. It also makes use of subsequence matching to reduce the handling time of longer sequences and adopts the concept of valid node to improve the correctness of mining results.

The behavior patterns of mobile user in mobile communication environment are also a kind of sequential patterns. Lee et al. [10] proposed the T-Map-Mine algorithm for mining the behavior patterns of mobile user. It mines mobile sequential patterns from mobile user's signal data within each time interval at each location. A mobile sequential pattern includes a sequential pattern and a moving path. T-Map-Mine uses T-Map-Tree to store sequential patterns and adds the moving locations and customer data to T-Map-Tree one by one to increase the support of locations and sequential patterns. The mining results are the sequential patterns with frequent moving paths found in T-Map-Tree. In the mining process, T-Map-Mine must continuously search and insert nodes. Besides, the tail node of each branch of T-Map-Tree is connected to a RST (Requested Services Tree) to store sequential patterns. Therefore, in processing large amount of data, more time is needed for the construction and traversal of the tree.

III. THE ASPAMDAS METHOD

In multiple date streams environment, data will grow larger in size and become more complicated with the change of real-world environment. However, existing algorithms can only handle a single item at a time and are hence incapable of coping with the changing environment of multiple data streams. Based on commercial consideration, mobile operators may need to find out the association among streams produced at different locations to provide their customers with better services. In this paper, therefore, we propose the ASPAMDAS (Across-streams Sequential Pattern Mining in Multiple Data Streams) method to solve these problems. ASPAMDAS mainly consists of two stages: data sampling and incremental mining. In the stage of data sampling, we adopt the sliding window model to sample stream data. The size of the sampled data will be the window size. In the stage of incremental mining, we convert customer transaction data in the sliding window into a bitmap to find across-streams sequential patterns hereby to improve mining efficiency. After that, new and old across-streams sequential patterns are either updated or eliminated to produce precise mining results. Figure 1 shows the overall process of the ASPAMDAS method.

© 2011 ACADEMY PUBLISHER

A. Data Sampling

In data stream environment, due to the increasing need of real-time knowledge and the limitation of data streams, the newer the data is, the more important it becomes. Accordingly, we use a sliding window to sample new stream data and dump the old ones that have been processed. As shown in Figure 2, the sliding window only maintains the newest N pieces of transaction data. N stands for the window size. After the transaction data within the window is processed, the window will move forward one time point to sample the newest stream data to process.

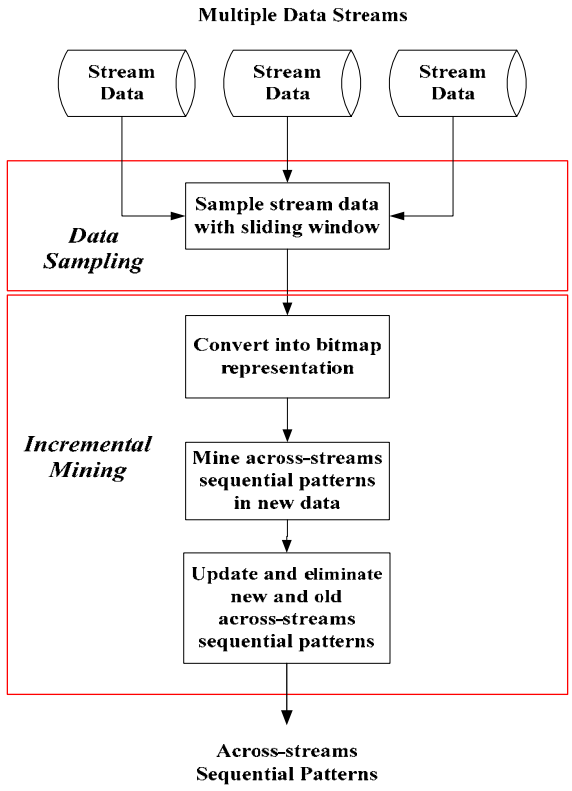


Figure 1. Overall process of ASPAMDAS

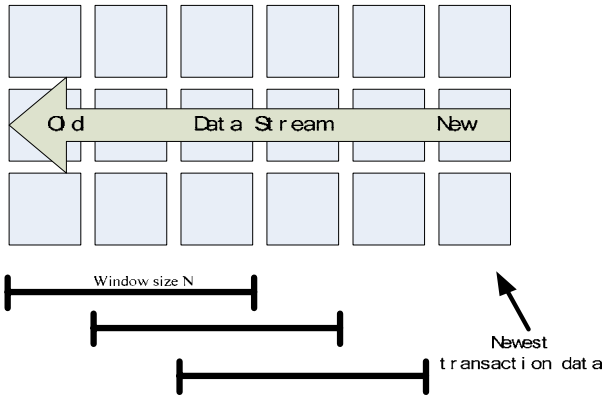


Figure 2. Sliding window in data stream environment

Figure 3 shows an example of multiple data streams. S1, S2, and S3 represent three data streams. Data from three data streams will be received at each time point. Wi stands for the i^{th} window. Each data stream can be regarded as a location of customer transaction; whereas the stream data at each time point can be viewed as a piece of customer transaction data. These customer transaction data ordered by their arrival sequence become the transaction sequence data. The window size is 3, which means each time the sliding window samples transaction data of the latest 3 time points. For example, the customer transaction data sampled at the first time point are C1:(a,c), C3:(b,c,d), and C4:(b,c). The sliding window will move in the direction of time. After the transaction data in W1 are processed, the sliding window will move forward one time point to process the transaction data in W2. In this way, stream data will be mined incrementally.

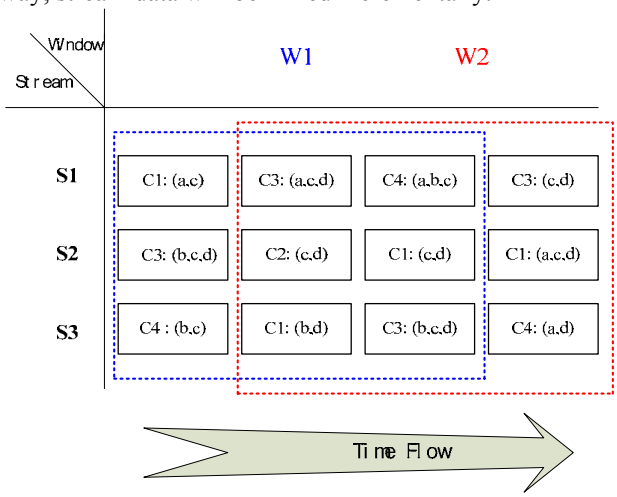


Figure 3. Example of multiple data streams

B. Incremental Mining

As described in previous subsection, a sliding window is used to sample stream data for mining. But due to the characteristics of data stream and the limitation of memory capacity, not all the mining results will be completely stored. Therefore, the concept of incremental mining is introduced to prevent the problem of imprecise mining results. As new stream data are produced continuously, the mining results will be continuously updated with the mining of new stream data. Below are possible situations of sequential pattern updating:

1. New data bring in new sequential patterns, which are frequent sequential patterns.
2. After updated, frequent sequential patterns remain frequent sequential patterns.
3. Frequent sequential patterns may become infrequent sequential patterns.
4. Infrequent sequential pattern may turn into frequent sequential patterns.

Situation 1: A frequent sequential pattern mined in current sliding window does not appear in previous mining results. This sequential pattern is a new frequent sequential pattern.

Situation 2: A frequent sequential pattern mined in current sliding window already exists in previous mining results. After updating its support, this sequential pattern remains a frequent sequential pattern.

Situation 3: A previously mined frequent sequential pattern does not appear in later sliding windows. When

© 2011 ACADEMY PUBLISHER

its support decays to less than the minimum support, this sequential pattern will become an infrequent sequential pattern.

Situation 4: An infrequent sequential pattern may turn into a frequent sequential pattern in current sliding window if in previous mining results each of its subsequences is a frequent sequential pattern and does not have any supersequence.

Definition 1 (**support**): Let α be a sequential pattern. The support of α is the ratio of the number of transactions containing α to the number of all transactions.

Definition 2 (**frequent sequential pattern**): Let α be a sequential pattern. Then α is a frequent sequential pattern if and only if the support of α is greater than or equal to the minimum support.

Definition 3 (**frequent across-streams sequential pattern**): Let α be a sequential pattern, β be a stream sequence, and the sequence of streams containing the subsequences of α be a subsequence of β. Then α is a frequent across-streams sequential pattern if and only if β is a frequent stream sequence.

Figure 4 shows the IAspam algorithm whose parameter definitions are as follows:

- SW' stands for the data in current sliding window and SW stands for the data in previous sliding windows.
- min_sup stands for minimum support.
- S stands for stream number, I for itemset, CID for customer identification, and SD for stream data.
- k-item represents the item named k.
- CP stands for candidate sequence.
- CS stands for customer sequence table.
- $sup_{sw}(p)$ stands for the support of sequential pattern p in SW and $sup_{sw'}(p)$ stands for the support of sequential pattern p in SW.
- L-tree stands for lexicographical sequence tree.
- FASP represents frequent across-streams sequential pattern and StP represents stream sequence.
- $Sup_{SD}(p)$ represents the support of sequential pattern p in SD.
- Sup(FASP) represents the support of sequential pattern FASP in L-tree.

Algorithm: IAspam

Input: SW', SW, min_sup, CID, S, I, k-item, CS, L-tree
Output: FASP, StP

1. **For each** transaction data of SW' **do**
   store into CS as <S, I>;
2. **For each** transaction sequence of CS **do**
   convert into bit-vector and store into bitmap matrix;
3. **For each** 1-item **do**
   add 1-item and sup_{SD}(1-item) into L-tree;
   If sup_{SD}(1-item) ≧ min_sup **then**
   generate CP with I-step and S-step and map to bitmap matrix;
4. **For each** CP **do**
   If $sup_{sw'}$(CP) ≧ min_sup **then**
   If CP exists in L-tree **then** update sup_{sw}(CP);
   else add FASP and StP into L-tree;
5. **For each** FASP in L-tree **do**
   If sup(FASP) not updated **then**
   decay sup(FASP);
   If sup(FASP) < min_sup **then** delete FASP;
6. **return**;

Figure 4. IAspam algorithm

The IAspam algorithm consists of three parts. The first part is the conversion into bitmap representation (Step 1-2). Customer transaction data will be converted into a bitmap matrix. The second part is the mining of across-streams sequential patterns in new data (Step 3). I-step and S-step are conducted on the 1-item to generate candidate sequences, which will be mapped to the bitmap matrix. The third part is the integration of new and old sequential patterns (Step 4-5). The newly mined patterns will be compared with previous patterns to update sequential patterns and decay the support of the sequential patterns not updated. Below is the detailed explanation of the steps of the IAspam algorithm.

Step 1: Store customer transaction data in current sliding window into the customer sequence table, in which data are ordered by customer identification. Data format is <S, I>.

Step 2: Convert each transaction sequence of the customer sequence table into a bitmap vector of the customer bitmap matrix. Judge if the itemset at each time point contains k-item. If yes, store bit 1 into the column of that time point of k-item; otherwise, store bit 0.

Step 3: Count the support of each 1-item in each stream and store each 1-item and its supports into L-tree. Judge if every support of a 1-item is greater than or equal to the minimum support. If yes, conduct I-step and S-step to generate candidate sequences and map them to a bitmap matrix. In the mapping process, the candidate itemsets generated by I-step will be used to search for the '1' bits that are at the same time point and in the same stream but not in the same item, and count the support. Whereas, the candidate sequences generated by S-step will be used to search, in different items and at different time points, for the '1' bits whose prefix subsequences are at the same time point and in the same stream and the '1' bits whose suffix subsequences are at the same time point and in the same stream, and count the support.

Step 4: Judge if the support of each candidate sequence in SW' is greater than or equal to the minimum support. If yes, check to see if the frequent sequential pattern exists in the lexicographical tree. If yes, update the support of this sequential pattern. Otherwise, insert a frequent across-streams sequential pattern (FASP) and a stream sequence (StP) into the lexicographical tree.

Step 5: Judge if the support of each FASP in the lexicographical tree has been updated. If not, decay the support of FASP and judge if the decayed support is less than the minimum support. If yes, delete FASP to get rid of the FASP that has not been updated for a long time.

Step 6: Output FASP and StP.

The steps of sequence extension are to conduct I-step before S-step. I-step is to insert a new item into an itemset. For example, after going through I-step, <(a,b)> and <(c)> will become <(a,b,c)>, which can be expressed as <(a,b)> $\diamondsuit_I$ <(c)> = <(a,b,c)>. S-step is to append a new itemset to an itemset. For instance, after undergoing S-step, <(a,b)> and <(c)> will turn into <(a,b)(c)>, which can be expressed as <(a,b)> $\diamondsuit_S$ <(c)> = <(a,b)(c)>. According to the Apriori property, the supersequences of an infrequent sequence cannot be frequent. Consequently, it

© 2011 ACADEMY PUBLISHER

is not necessary to conduct I-step and S-step on infrequent sequences and hence no surplus candidate sequences will be generated.

Each leaf node of a lexicographical tree records a frequent across-streams sequential pattern plus its support and stream sequence. The purpose of constructing a lexicographical tree is to store frequent across-streams sequential patterns in order to insert new across-streams sequential patterns or to combine or update new and old across-streams sequential patterns. The nodes of a lexicographical tree have a lexicographical order. The order of constructing a lexicographical tree is from top to bottom and from left to right. According to the characteristics of lexicographical order, the possible lexicographical orders of sequences are as below: [8]:

- If s' = s $\diamond$ p, then s $<$ s'; e.g. <(a,b)> $<$ <(a,b)(b)>.
- If s = α $\diamond_I$ p, s' = α $\diamond_I$ p', and p $<$ p', then s $<$ s'; e.g. <(a,b,c)> $<$ <(a,b,d)>.
- If s = α $\diamond_S$ p, s' = α $\square_S$ p', and p $<$ p', then s $<$ s'; e.g. <(a,b)(c)> $<$ <(a,b)(d)>.
- If s = α $\square_I$ p and s' = α $\square_S$ p', then s $<$ s' regardless the order of p and p'; e.g. <(a,b,c)> $<$ <(a,b)(a)>.

After the construction of the lexicographical tree, the across-streams sequential patterns in the tree will be updated with the production of new mining results. The across-streams sequential patterns that have not been updated for a long time may become obsolete. In maintaining the lexicographical tree, it is therefore necessary to decay their supports to delete those sequential patterns that have not been updated for a long time. The decay equations are as follows:

$$weight = d^{p},\ 0 \le d \le 1 \tag{2}$$

$$p = w - u \tag{3}$$

$$w\_\sup = weight \times \sup \tag{4}$$

In (2), d stands for the decay rate that is defined by the user to control the speed of decaying sequential patterns. In (3), p is the decay period of a sequential pattern, w is the number of times of sliding window movement, and u is the number of times of updating a sequential pattern in the lexicographical tree. The decay period of a sequential pattern is obtained by deducting the number of times of updating the sequential pattern from the number of times of sliding window movement. The longer a sequential pattern has not been updated, the bigger its p value becomes, which means its weight will become lower accordingly. In (4), w_sup stands for the weighted support a sequential pattern, which is obtained by multiplying the support of the sequential pattern by its weight. An across-streams sequential pattern will be deleted when its weighted support is less than the minimum support.

We will use an example to illustrate the IAspam algorithm. Table I is the CID-ordered customer sequence table of the data sampled for the first execution of IAspam (W1). In a customer sequence table, the transaction data of each customer transaction sequence are sequenced according to the order of their arrival time. Because the size of the sliding window is set to 3, customer transaction data at 3 time points in each data stream are sampled. Each customer transaction data is represented as <S_i, α>, in which S_i is a stream number and α is an itemset. <S1, (a,c)> means that itemset (a,c) comes from stream S1. The empty sets in the first and third columns whose CID is 2 mean that there is no transaction at these 2 time points for customer #2.

TABLE I.
CUSTOMER SEQUENCE TABLE OF W1

CID	Customer transaction sequence
1	<S1, (a,c)> <S3, (b,d)> <S2, (c,d)>
2	<∅> <S2, (c,d)> <∅>
3	<S2, (b,c,d)> <S1, (a,c,d)> <S3, (b,c,d)>
4	<S3, (b,c)> <∅> <S1, (a,b,c)>

After the data of W1 is mined, the sliding window will move forward one time point to sample data for the second execution of IAspam (W2). As shown in Table II, three pieces of new customer transaction data, <S2, (a,c,d)>, <S1, (c,d)>, and <S3, (a,d)>, are appended respectively to the customer transaction sequences whose CIDs are 1, 3 and 4. These three pieces of customer transaction data are the new stream data in W2. The customer transaction sequence whose CID is 2 does not have any new transaction data, which means this customer did not engage in any transaction at that time point.

TABLE II.
CUSTOMER SEQUENCE TABLE OF W2

CID	Customer transaction sequence
1	<S3, (b,d)> <S2, (c,d)> <S2, (a,c,d)>
2	<S2, (c,d)> <∅> <∅>
3	<S1, (a,c,d)> <S3, (b,c,d)> <S1, (c,d)>
4	<∅> <S1, (a,b,c)> <S3, (a,d)>

Figure 5 is the bitmap matrix converted from the customer sequence table of W1, in which a, b, c, and d represent items and CID=1 represents customer #1. For each customer, there is a bit list for each item and these bit lists are converted from the transaction sequence of this customer. The bit list (1,0,0) for item a means that item a appears in the transaction data at the first time

© 2011 ACADEMY PUBLISHER

point and does not appear in the transaction data at the second and third time points. The stream sequence <S1,S3,S2> means that the transaction data at the first, second, and third time points come respectively from streams S1, S3, and S2.

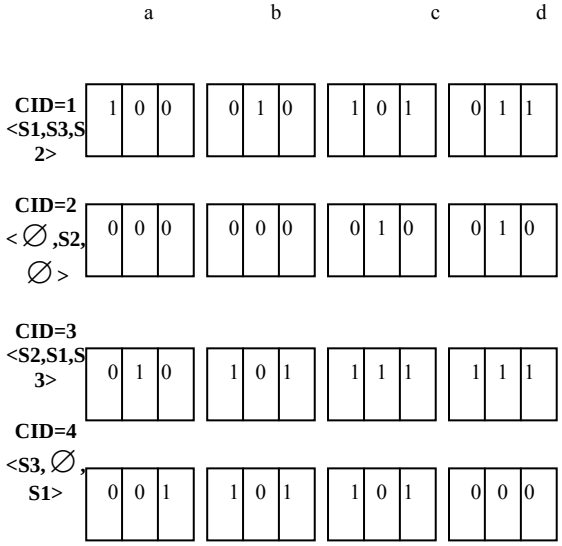


Figure 5. Bitmap matrix of W1

After the customer sequence table is converted into a bitmap matrix, the support of each 1-item will be counted to generate candidate sequences. Figure 6 shows the support of each 1-item in each stream. Suppose the minimum support is set to 2. The supports of c-item in S1, S2, and S3 are 3, 3, and 2 respectively, all of which are greater than or equal to the minimum support. Therefore, c-item is a frequent item in each of these three streams.

<a>	<b>	<c>	<d>
[S1:3,S2:0,S3:0]	[S1:1,S2:1,S3:3]	[S1:3,S2:3,S3:2]	[S1:1,S2:3,S3:2]

Figure 6. Support of 1-item

Based on the 1-item in Figure 6, sequence extension will go through I-step and S-step to generate candidate sequences, which will be mapped to the bitmap matrix. As shown in Figure 7, different extension steps use different ways of mapping, which are described below.

- I-step: The candidate itemset <(a,c)> extended using I-step will be mapped to the bitmap matrix and its support will be counted. As shown in Figure 8, a-item and c-item have bit 1 (the 1 in bold type) at the same time point for customers #1, #3, and #4 and all of them appear in S1. Therefore the support of <(a,c)> is 3. In Figure 6, as a-item is infrequent in S2 and S3, it will remain infrequent after I-step. Accordingly, it needs not to be processed.
- S-step: After going through S-step, <(a,c)> and <(b,d)> will become <(a,c)(b,d)>, which will be mapped to the bitmap matrix. Due to the time sequence of a sequential pattern, <(b,d)> must be mapped to a time point after that of <(a,c)>. As shown in Figure 7, bit 1 (the underlined 1) appears at the first time point of a-item and c-item as well as the second time point of b-item and d-item for customer #1. Also, bit 1 (the underlined 1) appears at the second time point of a-item and c-item as well as the third time point of b-item and d-item for customer #3. In the meantime, <(a,c)> and <(b,d)> show up respectively in S1 and S3. Therefore, the support of <(a,c)(b,d)> is 2 and the stream sequence is <(S1)(S3)>.

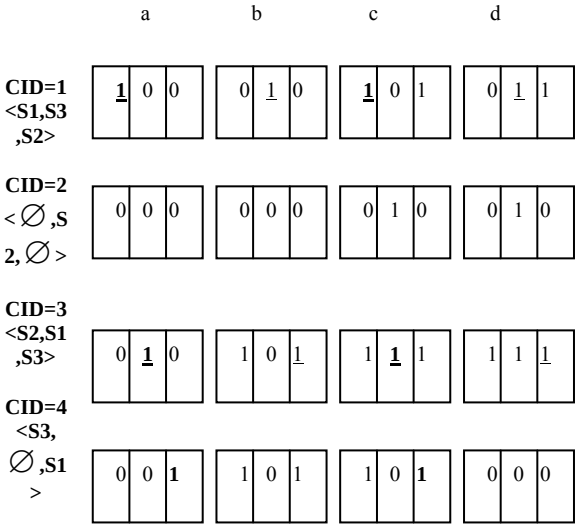


Figure 7. Mapping candidate sequences to bitmap matrix

Suppose the minimum support is set to 2. Figure 8 and Table III show separately the lexicographical tree and result of the first execution of IAspam. The across-streams sequential patterns in the lexicographical tree include <(a)(b)>, <(a)(d)>, <(c)(b)>, <(c)(d)>, and <(a,c)(b,d)>. The stream sequence of all of these across-streams sequential patterns is <(S1)(S3)>, which means that their moving path goes from stream S1 to stream S3. As <(a,c)(b,d)> is the supersequence of all the other across-streams sequential patterns, it will be the only frequent across-streams sequential pattern.

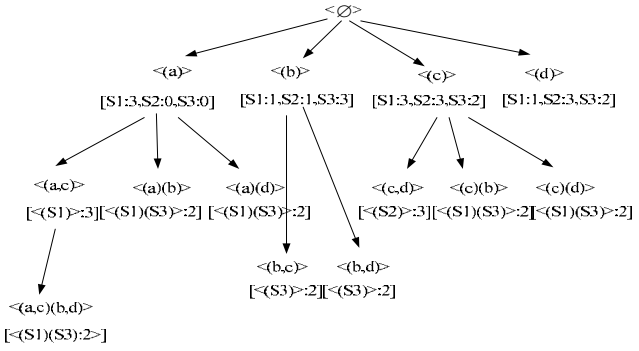


Figure 8. Lexicographical tree of the first execution

TABLE III.
RESULT OF THE FIRST EXECUTION

© 2011 ACADEMY PUBLISHER

Frequent across-streams sequential pattern	Stream sequence
<(a,c)(b,d)>	<(S1)(S3)>

Figure 9 and Table IV show separately the lexicographical tree and result of the second execution of IAspam. In Figure 9, the support within a red square is an updated support, which indicates the sequential pattern remains frequent in W2 and its support is updated from 3 to 2. The support within a red circle is a weighted support, which is decayed from 2 to 1.8 under the condition that the decay rate is set to 0.9. Accordingly, the sequential pattern will be deleted because its support is less than the minimum support, and the link will be represented as a dotted line.

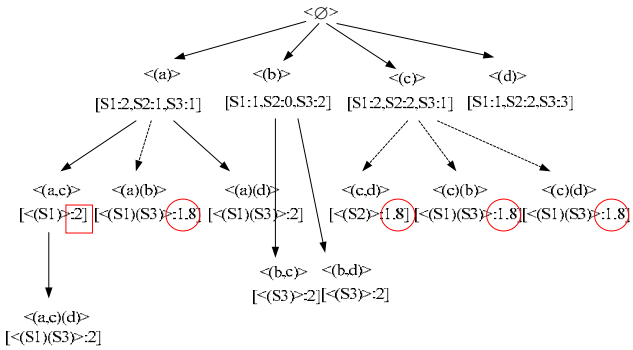


Figure 9. Lexicographical tree of the second execution

TABLE IV.
RESULT OF THE SECOND EXECUTION

Frequent across-streams sequential pattern	Stream sequence
<(a,c)(d)>	<(S1)(S3)>

IV. PERFORMANCE EVALUATION

A. Experimental Environment and Data

Table V shows the experimental environment. In NetBeans IDE 6.1 compilation environment, Java JDK 6.0 programming language is used to write programs. A Pentium D 2.8 GHz processor and 1 GB memory are used. Transaction data generated by the synthetic data generator developed by IBM Almaden Research Center are used as experimental data. Table VI shows the parameters of experimental data. For instance, the expression S6T100I3 stands for a multiple data streams environment in which there are 6 data streams, 100 time points, and averagely 3 items per transaction.

TABLE V.
EXPERIMENTAL ENVIRONMENT

Equipment	Specification
Processor	Pentium D 2.8 GHz
Memory	1 GB
Hard Disk	160 GB
Operating System	WINDOWS XP
Programming Language	Java JDK 6.0

TABLE VI.
PARAMETERS OF EXPERIMENTAL DATA

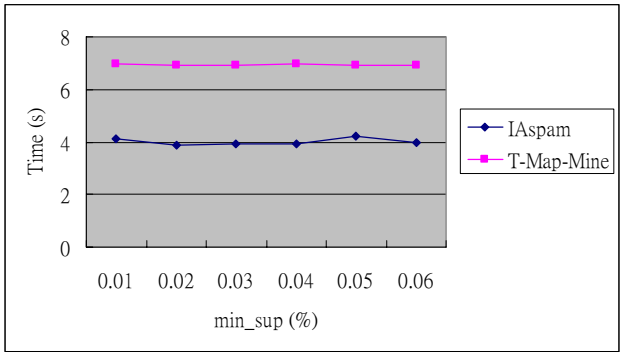
Parameter	Parameter Description
S	Number of data streams
T	Number of time points
I	Average number of items per transaction
C	Number of customers
D	Number of item varieties

B. Performance Comparison

Both IAspam and T-Map-Mine are used for mining sequential patterns of moving paths in data stream environment. IAspam uses a bitmap representation for mining frequent across-streams sequential patterns; whereas T-Map-Mine uses a T-Map-Tree structure for mining behavior sequence patterns. In the mining process, T-Map-Mine needs to repeatedly search T-Map-Tree to insert new nodes and increase the support, and hence may take a great amount of time in constructing and maintaining T-Map-Tree. On the contrary, IAspam maps candidate sequences to a bitmap matrix based on the stream, which can save a lot of execution time while searching for across-streams sequential patterns. We compare the execution and memory usage of these two algorithms.

(1) Execution Time

Figure 10 shows the comparison of execution time between IAspam and T-Map-Mine in the S5T100C100I1 multiple data streams environment. The vertical axis is execution time in second and the horizontal axis is the minimum support in percentage. To be able to compare with T-Map-Mine, the average number of items per transaction is set to 1. The execution time of IAspam and T-Map-Mine are compared under various minimum supports. As seen in Figure 10, the execution time of IAspam is less than that of T-Map-Mine under various minimum supports. This is because T-Map-Mine must repeatedly search T-Map-Tree and RST for each customer data as well as insert new nodes and increase the support, which will take more time in the mining process.



© 2011 ACADEMY PUBLISHER

Figure 10. Comparison of execution time

(2) Memory Usage

Figure 11 shows the comparison of memory usage between IAspam and T-Map-Mine in the same environment as that for the comparison of execution time. The vertical axis is the maximum usage of memory in kilobyte (KB). As seen in Figure 11, the memory usage of IAspam is about the same as that of T-Map-Mine under various minimum supports. The memory usage of IAspam will be affected by the number of item varieties and the number of customers. The branching of T-Map-Tree of T-Map-Mine will depend on the complexity of customer transaction data. The more chaotic the data is, the bigger the T-Map-Tree becomes.

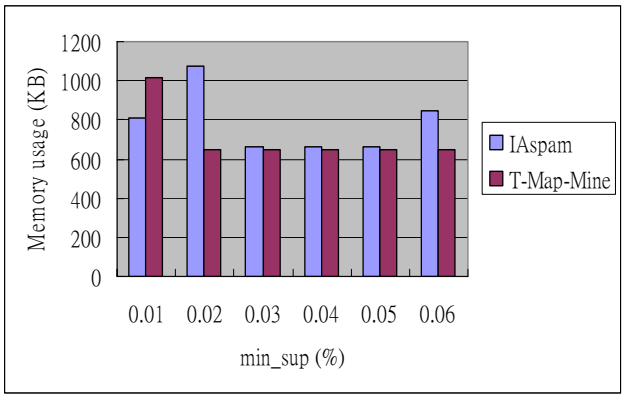


Figure 11. Comparison of memory usage

V. Conclusion

The primary difference between the IAspam algorithm proposed in this paper and existing algorithms for mining sequential patterns in multiple data streams is that IAspam is for mining across-streams sequential patterns, which are a new type of sequential patterns. To avoid inaccurate mining results caused by the re-mining of data, IAspam uses a sliding window to sample stream data and incrementally mines across-streams sequential patterns to maintain the newest mining results. To solve the inefficiency problem caused by mining a massive amount of data, IAspam uses a bitmap representation to search for across-streams sequential patterns and count their supports. As sequential patterns may cross different streams, IAspam searches for sequential patterns based on the stream, which enables the identification of not only across-streams sequential patterns but also stream sequences. Experimental results show that the IAspam algorithm is effective in execution time when processing large amounts of stream data.

Acknowledgment

The authors would like to express their appreciation for the financial support from the National Science Council of Republic of China under Project No. NSC 98-2221-E-031-003.

References

[1] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and Issues in Data Stream Systems. The 21st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems. 2002 June 1-16; Madison, Wisconsin, USA.

[2] T. Oates and P. R. Cohen. Searching for Structure in Multiple Streams of Data. The 13th International Conference on Machine Learning. 1996 July 346-354; Bari, Italy.

[3] L. Golab and M. T. Ozsu. Issues in Data Stream Management. ACM SIGMOD Record. 2003 June Volume 32 Issue 2: 5-14.

[4] J. Chang and W. Lee. Decaying Obsolete Information in Finding Recent Frequent Itemsets over Data Stream. IEICE Transaction on Information and Systems, June, 2004; No. 6, Vol. E87-D.

[5] G. Chen, X. Wu, and X. Zhu. Sequential Pattern Mining in Multiple Streams. The 5th IEEE International Conference on Data Mining, Washington. 2005 Nobember 27-30; USA.

[6] J. H. Chang and W. S. Lee. Finding Recent Frequent Itemsets Adaptively over Online Data Streams. The 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining; 2003 August 487-492; Washington, USA.

[7] C. Raissi, P. Poncelet, and M. Teisseire. SPEED: Mining Maximal Sequential Patterns over Data Streams. The 3rd IEEE International Conference on Intelligent Systems. 2006 September 546-552; London, UK.

[8] C. I. Ezeife and M. Monwar. SSM：A Frequent Sequential Data Stream Patterns Miner. The IEEE Symposium on Computational Intelligence and Data Mining, Honolulu. 2007 March 120-126; USA.

[9] H. Li and H. Chen. GraSeq: A Novel Approximate Mining Approach of Sequential Patterns over Data Stream. The 3rd International conference on Advanced Data Mining and Applications. 2007 May 401-411; Harbin, China.

[10] S.-C. Lee, E. Lee, W. Choi, and U. M. Kim. Extracting Temporal Behavior Patterns of Mobile User. The 4th International Conference on Networked Computing and Advanced Information Management. 2008 September 455-462; Gyeongju, Korea.

Shih-Yang Yang received his Ph.D. degree in Computer Science and Information Engineering from Tamkang University, Taiwan, in January 2008. Since January 2008, he is an Associate Professor with the Department of Media Art at Kang-Ning Junior College of Medical Care and Management (Taipei, Taiwan). His research interests include parallel & distributed systems, web technology, and multimedia.

Ching-Ming Chao received his B.S. degree in Computer Science from Soochow University, Taipei, Taiwan in 1982 and his Ph.D. degree in Computer Science from The University of Iowa, Iowa City, Iowa, U.S.A. in 1990. From 1990 to 1992, he was an assistant professor in the Department of Computing Sciences at the University of Scranton, Scranton, Pennsylvania, U.S.A. He joined the faculty of the Department of Computer and Information Science at Soochow University in 1992 as an

© 2011 ACADEMY PUBLISHER

associate professor. From 1992 to 1996, he served as the department chair. Since 2003, he has been a professor in Department of Computer Science and Information Management at Soochow University. During the past few years, he has published more than 90 refereed journal and conference papers. His research interests include data mining, data warehousing, database, and web technology.

Po-Zung Chen received his Ph.D. degree in Computer Science from the University of Iowa in December 1989. From November 1989 to May 1990, he was a visiting Assistant Professor at Michigan Technological University (Houghton, Michigan). Since August 1990, he is an Associate Professor with the Department of Computer Science and Information Engineering at Tamkang University (Taipei, Taiwan). His research interests include object-oriented distributed programming, parallel & distributed systems, and simulation & modeling.

Chu-Hao Sun is currently a candidate of Ph.D. student in department of Computer Science and Information Engineering in Tamkang University (Taipei, Taiwan). He received his B.E. and M.E. degree from the same university in 1995 and 1998. His research interests include database management system, parallel processing, web technology, and data mining.

© 2011 ACADEMY PUBLISHER