SSM : A Frequent Sequential Data Stream Patterns Miner

C.I. Ezeife[*], Mostafa Monwar
School of Computer Science, University of Windsor,
cezeife@uwindsor.ca, woddlab@uwindsor.ca
http://www.cs.uwindsor.ca/∼cezeife

Abstract

Data stream applications like sensor network data, click stream data, have data arriving continuously at high speed rates and require online mining process capable of delivering current and near accurate results on demand without full access to all historical stored data. Frequent sequential mining is the process of discovering frequent sequential patterns in data sequences as found in applications like web log access sequences. Mining frequent sequential patterns on data stream applications contend with many challenges such as limited memory for unlimited data, inability of algorithms to scan infinitely flowing original dataset more than once and to deliver current and accurate result on demand. Existing work on mining frequent patterns on data streams are mostly for non-sequential patterns. This paper proposes SSM-Algorithm (Sequential Stream Mining-algorithm), that uses three types of data structures (D-List, PLWAP tree and FSP-tree) to handle the complexities of mining frequent sequential patterns in data streams. It summarizes frequency counts of items with the D-List, continuously builds PLWAP tree and mines frequent sequential patterns of batches of stream records, maintains mined frequent sequential patterns incrementally with FSP tree. The proposed algorithm can be deployed to analyze E-commerce data where the primary source of data is click stream data.

Keywords *Web Sequential Mining, Stream Mining, Customer Access Sequence, Frequent Sequential patterns, Click Steam Data*

1 Introduction

A data stream is a continuous, unbounded, and high-speed flow of data items. Applications generating large amounts of data streams, include 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 sequential 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 transaction and mining frequent sequential patterns from them presents additional complexities. Simply mining a set of items would treat the CAS sequence example above as the set (TV, radio, jean pants, shirt) and the order is not important as is the focus for such non-sequential mining algorithms like FP-stream algorithm [Giannella et al., 2003]. So, the SSM-Algorithm implements a continuous process that extracts frequent CAS from incoming stream in one pass, mines sequential patterns, stores the patterns compactly and quickly. While many sequential mining algorithms (e.g., PLWAP [Ezeife & Lu, 2005]) mine statically pre-processed data like web log data containing hundreds of records, the SSM-Algorithm, dynamically processes one customer sequence after the other to form batches of records to progressively mine. Then, a data structure, called D-LIST, which maintains dynamically, the total frequency counts of all items that have passed through the stream, is used. D-List collects and updates frequency of all items passing through streams and is used for computing the frequent 1-item list for each batch after updating the D-List with the streams of the batch. A busy website generates a huge amount of click stream data everyday. Each click stream data series

[*]This research was supported by the Natural Science and Engineering Research Council (NSERC) of Canada under an Operating grant (OGP-0194134) and a University of Windsor grant.

reflects a customer's buying interest. For an E-commerce company, detecting future customers based on their sequential mouse movements on the content page would help significantly to generate more revenue. There are some recent studies on mining data streams, classification of stream data [Domingo & Hulten, 2000], online classification of data streams [Las, 2002], clustering data streams [Guna et al., 2000], web session clustering [Gunduz & Ozsu, 2003], approximate frequency counts over data streams [Manku & Motwani, 2002], mining frequent patterns in data stream at multiple time granularities [Giannella et al., 2003], and multi-dimensional time series analysis [Chen et al., 2002], temporal pattern mining in data streams
[Teng, Chen & Yu, 2003] but more work is needed on mining frequent sequential patterns in data streams.

1.1 Contributions

Considering the importance of sequential mining in data streams, this paper proposes the SSM-Algorithm, a sequential stream miner that extends the functionality of PLWAP to make it compatible in data stream environment using additional efficient data structures D-List and FSP-tree.

1.2 Related Work

Han et al. in [Han et al., 2004] proposed the FP-Tree algorithm to generate frequent pattern itemsets. Frequent sequential pattern mining algorithms include the GSP [Srikanth & Aggrawal, 1996], which is Apriori-like, PrefixSpan [Pei et al., 2001], a pattern-growth method, the WAP [Pei et al., 2000], which is based on the FP-Tree but used for mining sequences, the PLWAP [Ezeife & Lu, 2005], [Ezeife et al., 2005], which is the position-coded, pre-order linked version of WAP that eliminates the need for repetitive re-construction of intermediate trees during mining. PLWAP tree algorithm first builds each frequent sequential pattern from database transactions from "Root" to leaf nodes assigning unique binary position code to each tree node and performing the header node linkages pre-order fashion (root, left, right). Both the pre-order linkage and binary position codes enable the PLWAP to directly mine the sequential patterns from the one initial WAP tree starting with prefix sequence, without re-constructing the intermediate WAP trees. To assign position codes to a PLWAP node, the root has null code, and the leftmost child of any parent node has a code that appends '1' to the position code of its parent, while the position code of any other node has '0' appended to the position code of its nearest left sibling. The PLWAP technique presents a much better performance than that achieved by the WAP-tree technique, making it a good candidate for stream sequential mining. Data stream algorithms include the Lossy counting algorithm [Manku & Motwani, 2002], which is used for frequency counting in data streams. FP-Stream algorithm [Giannella et al., 2003] is used to mine frequent patterns in data stream. Previously, landmark model [Manku & Motwani, 2002] was introduced that mines frequent patterns in data stream by assuming that patterns are measured from the start of the stream up to current moment. The authors of FP-Stream also extended their framework to answer time-sensitive queries over data stream. Teng et al. proposed FTP-Algorithm [Teng, Chen & Yu, 2003] to mine frequent temporal patterns of data streams. FTP-DS scans online transaction flows and generate frequent patterns in real time. Sliding window model is used in this paper. Data expires after N time units since its arrival.

2 The Proposed SSM Sequential Stream Mining Algorithm

On continuous arrival of data streams, the SSM-Algorithm (sequential stream mining-algorithm) forms sized batches dynamically by updating and storing on the D-List structure, frequency counts f of each item meeting the defined support boundaries of $e \leq f$, given the tolerance error support threshold e. D-List uses hash chain indexing to maintain incoming elements and their frequencies and is very efficient in such stream applications like E-commerce, where thousands of items are used, and brand new items get posted to the E-commerce site while unpopular items get discontinued from the site on a regular basis. Proposed algorithm handles dynamically sized batches and provides added flexibility to suit the rate of stream flow. FP-Stream [Giannella et al., 2003] or Lossy Counting algorithm [Manku & Motwani, 2002] use fixed size batches. The SSM-Algorithm next performs batch mining with the PLWAP sequential mining technique and constantly stores frequent sequential pattern result on a compact tree (the FSP-tree), that is able to deliver results on demand. SSM-Algorithm uses previously introduced efficient PLWAP-tree algorithm to find frequent sequential patterns. Thus, SSM takes advantage of the preordered linkage and position coding of PLWAP-tree in order to eliminate the cost of reconstruction and computation time of intermediate trees during mining, unlike the FP-tree [Han et al., 2004]. The SSM-Algorithm uses the FSP-tree for storing mined batch frequent sequential patterns. The FSP tree is a simple form of pattern-tree [Giannella et al., 2003] that is introduced in FP-Stream algorithm. The produced result by SSM-Algorithm does not cross pre-defined error threshold and delivers all frequent sequential patterns that have user defined minimum support. The use of buffer mechanism in SSM restricts memory usage to a specific size. In other words, we use a small fixed size buffer in memory that handles continuous incoming data streams.

Algorithm 1 *(Sequential Stream Miner:Mines Frequent Sequential Stream Patterns)*

```
Algorithm SSM()
Input:     (1) Minimum support threshold (s) where $0 < s < 1$,
        (2) Maximum support error threshold (e) where $0 < e < s$,
        (3) Size of D-List (Size)
Output: 1) Frequent sequential patterns
Temp variables: exit = true, i=0,
        num-records (total number of database records);
Begin
        While (exit) // exit when user wants
        Begin
        1. i = i + 1 // indicates which batch or cycle
        2. Create-Batch(CAS) //creates a batch of CAS
                2.1 Scan $B_i$ and generate candidate
                1-sequences or $C_{1_{B_i}}$
        3. Update D-List[Size] with $C_{1_{B_i}}$
        // Hash index based structure creation
        4. Generate-Frequent-Pattern($FS_{B_i}$)
        with $PLWAP_{B_1}$ // FP generation
        5. Update-FSP-tree($FP_{B_i}$)
        // Update method for FSP-tree
        6. If user wants result, then from FSP-tree,
        get all FSP with count $\geq$ (s-e)* num-records
        7. Maintain-Structures()
        //i.e prune D-List and FSP, drop PLWAP,
        8 If user wants to exit, exit = false;
        End
End
```

Figure 1. The Main Sequential Stream Mining (SSM-Algorithm)

The proposed sequential data stream algorithm, SSM is given below as algorithm 1.

2.1 Steps in Sequential Stream Mining System

The main components and steps of the SSM-Algorithm is shown in Figure 2. Step 1: Buffer is basically a staging area where preprocessed transaction IDs and stream sequences like customer access sequences (CAS) arrive. We treat a buffer as a long empty string initially with limited size of about 50MB. Once the stream starts coming, they are added into the buffer. For example, (100, a b d a c), (101, a b c a c), ... Here, 100 is the transaction ID and the letters (a b d a c) following transaction ID 100 are item IDs for this transaction. Lossy Counting Algorithm [Manku & Motwani, 2002], uses buffer mechanism to deal with incoming data stream. On the other hand, FP-Stream [Giannella et al., 2003] uses main memory to handle data stream.

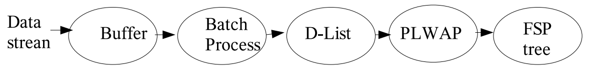


Figure 2. The Main Components of the SSM Stream Miner

While the stream sequences arrive at the Buffer area, each record is processed and placed in the current batch. The system mines a batch once it contains the minimum number of records set for a batch, and it continuously checks the buffer for more records every 1 minute (or a different value can be set to check the buffer based on application needs), if there are not enough records to form a batch. If there are enough records in the buffer, the maximum batch size is used in creating the current batch for mining. Thus, as soon as there is a batch with number of records equal to *b*, where $minimum\_Batch\_size \leq b \leq maximum\_Batch\_size$, the batch is mined. The SSM-Algorithm does not know how many click stream items it will be dealing with before forming each batch. For extremely busy stream environment where streams of data are arriving at faster rate than can be accommodated by the buffer, the buffer could either be increased, shut off for some time when too full and re-opened when de-congested. Another solution is to re-route non-accommodated arriving streams to a holding area, where they can be processed later during slow period. Our model can be used to find frequent sequential patterns of visited pages for a site as well.
Step 2: Batching Engine forms batches of stream sequences like the CAS (customer access sequences) data from the buffer. For example, a batch is a number (n) of customer access sequences similar to *n* record sequences of *n* distinct transaction IDs. The size of the batch depends on the incoming stream. The batch size of the SSM system is not fixed, unlike Lossy Counting Algorithm [Manku & Motwani, 2002], or FP-Stream [Giannella et al., 2003].
Step 3: Mining Algorithm or SSM-algorithm uses three data structures: D-List, PLWAP-tree and FSP-tree. It scans records of each batch, B_i, uses the counts of the items in this batch to update the item counts of items in the D-List. It then computes the cumulative support counts of D-List items to find overall frequent 1-items of this batch, $L_{1_{B_i}}$ (that is 1-item sequences with support in the database so far received after including counts of items in batch B_i, that are greater than or equal to the minimum support threshold, *s*). Then, all items in the batch records that are not in the $L_{1_{B_i}}$ list are removed from the stream sequences to create the batch frequent sequence, FS_{B_i}, used for batch mining. On arrival, the items are stored in D-List structure (a hash indexed array structure for storing all items with cumulative support count greater than or equal to the tolerance error support count, *e*). D-List keeps each

item's ID and their frequency in a hash chain that stores each unique item in a D-List hash bucket corresponding to the item-id modulus number of buckets. Only items whose frequent support count are greater than or equal to the total number of stream sequences that have passed through the database via all already processed batches multiplied by the percentage maximum tolerance error, e, are kept in the D-List. While items with support count less than this value are deleted from the D-List, those with count greater than or equal to the total number of stream records times (minimum support (s) minus tolerance error (e)) are kept in the $L_{1_{B_i}}$ list. The use of tolerance error to reduce the count of items kept in the D-List allows for handling items that may have been small (that is, not frequent) previously, and whose cumulative count would not be available if they suddenly turn frequent. Once the D-List is constructed, the performance of insertion, updating and deletion of nodes are faster through this hash chain structure. Thus, this structure contributes to processing speed and efficiency.

PLWAP-tree is constructed by taking frequent subsequences from the batch, PLWAP-algorithm is used to mine frequent sequential patterns and the sequential patterns are stored into FSP-tree incrementally. This process continues. When the next batch arrives, items are extracted from the batch and used to update frequencies of those items in the D-List if the items are already registered in the D-List. Otherwise, if the arriving items in the new batch are new elements to the D-List, they are inserted. Perform PLWAP-mining by extracting frequent subsequences from the current batch. If the obtained frequent patterns are already in FSP-tree, the frequencies of those patterns are updated in the tree, otherwise, they are inserted into the tree as new patterns. PLWAP-tree or Pre-ordered Linked WAP-tree was introduced in [Ezeife & Lu, 2005], [Ezeife et al., 2005]. The basic idea behind the PLWAP tree algorithm is using position codes and pre-order linkage on the WAP-tree [Pei et al., 2000] to speed up the process of mining web sequential patterns by eliminating the need for repetitive re-construction of intermediate WAP trees during mining. Eliminating the re-construction of intermediate WAP trees also saves on memory storage space and computation time. Given a set of frequent sequences of a stream batch, FS_{B_i}, the PLWAP tree is first constructed by inserting each sequence from root to leaf, incrementing the count of each item node every time it is inserted. Each node also has a position code from root, where the root has null position code and the position code of any other node has '1' appended to the position code of its parent node if this node is the leftmost child node, but it has '0' appended to the position code of its nearest left sibling if not the leftmost child node. After construction, the $L_{1_{B_i}}$ list is used to construct pre-order header linkage nodes for mining. The mining of the PLWAP tree is prefix based in that it starts from the root and following the header linkages, it uses the position codes to quickly identify item nodes of the same type (e.g., item a) on different branches of the tree at that level of the tree and if the sum of the counts of all these items (e.g. *a* node) on different branches of the tree is greater than or equal to the accepted minimum support count (number of records * (s - e)), then, this item is confirmed frequent and appended to the previous prefix frequent stream sequence.

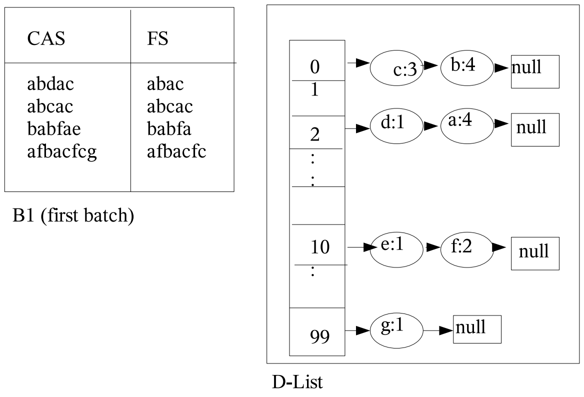
Frequent Sequential Pattern-tree or FSP-tree is a simple form of Pattern-tree [Giannella et al., 2003] for storing result structure. The sequences and their counts are simply inserted from root to leaf where the count of the sequence is assigned to the leaf of the frequent pattern. The FSP tree is maintained with both footer linked lists that has linkage to all leaf nodes for pruning nodes from leaf not meeting required support count, and from the Root for inserting newly found frequent sequential patterns.

3 An Example Application of the SSM Stream Miner

Assume a continuous stream with first stream batch B_1 consisting of stream sequences as: [(abdac, abcac, babfae, afbacfcg)]. Assume the minimum support, s is 0.75 (or 75%) and the tolerance error e, is 0.25 (or 25%).

The task is to continuously compute the frequent sequential stream patterns, FSP, of the streams as they arrive.

Step 1: The algorithm first computes the candidate 1-item of the batch as $C_{1_{B_1}}$ = {a:4, b:4, c:3, d:1, e:1, f:2, g:1} from the stream records as they arrive. Then, it updates the D-List with the $C_{1_{B_1}}$, keeping items with support count greater than or equal to (number of records times s) on the $L_{1_{B_1}}$ list, but keeping only items with minimum tolerance support count of (number of records * e) or more on the D-List. Since this is the first batch with only 4 records, the tolerance minimum support cardinality is: 4 * (0.75 - 0.25) = 2. Thus, all items with support greater than or equal to 2 should be large and in $L_{1_{B_1}}$. The D-List minimum error support cardinality is 4 * (0.25) = 1. Thus, all items with support greater or equal to 1 are kept in the D-List while those with support count greater than or equal to 2 are also in the $L_{1_{B_1}}$ list. The $L_{1_{B_1}}$ = {a:4, b:4, c:3, f:2}. The D-List after reading the batch B_1 is shown as Figure 3. Note that since the stream sequence being used in this example are those of E-commerce customer access sequence (CAS), the SKU (a special unique item code) for the items (a,b,c,d,e,f,g) given as (2, 100, 0, 202, 10, 110, 99) are used in the hash function (item-id modulus number of buckets) (for this example 100 buckets assumed) to find the bucket chain for inserting the item in the D-List. Since the $C_{1_{B_1}}$ has been used to update the D-List, which was used for computing the current $L_{1_{B_1}}$



CAS	FS
abdac	abac
abcac	abcac
babfae	babfa
afbacfcg	afbacfc

B1 (first batch)

Figure 3. The D-List After Batch B_1

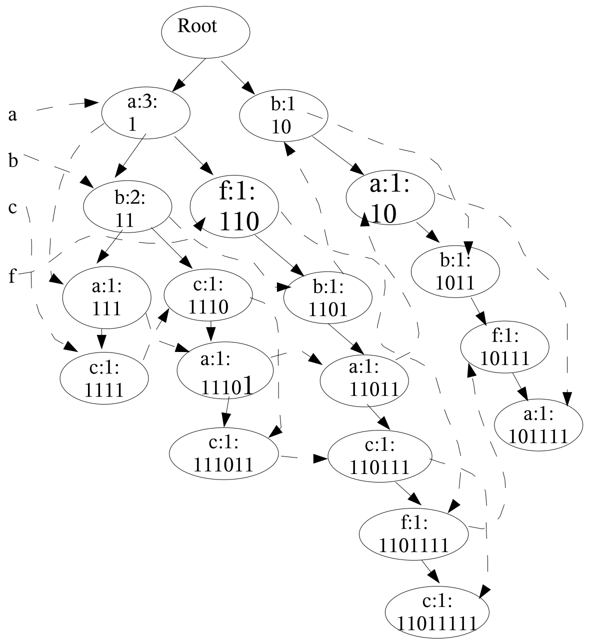


Figure 4. The PLWAP Tree of Batch B_1

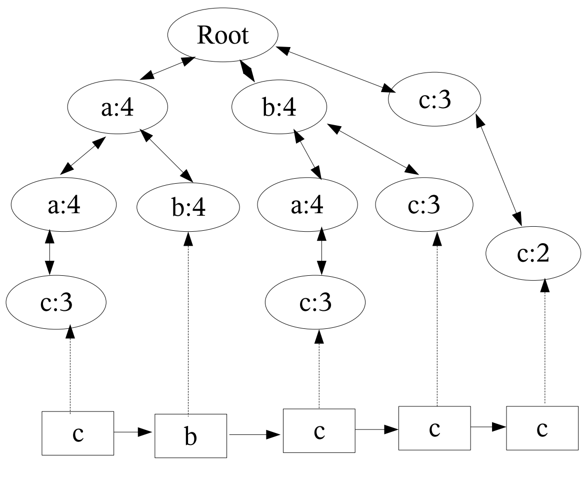


Figure 5. The FSP Tree After Batch B_1

and the frequent sequence FS_{B_1}, $C_{1_{B_1}}$ can now be dropped to save on limited memory.

Step 2: Now that the FS_{B_1} had been found, the next step entails using these FS_{B_1} to construct the $PLWAP_{B_1}$ tree and mining the tree to generate frequent stream sequential patterns for this batch. During mining, the minimum tolerance support of s-e or 0.50 for this example is used. The mining method recursively mines frequent sequential patterns from the tree to generate frequent sequential patterns for batch B_1 (or FP_{B_1}) with frequency of $(s-e) * |B1|$ = 0.50 * 4 = 2). The found FP_{B_1} = {a:4, aa:4, aac:3, ab:4, aba:4, abac: 3, abc: 3, ac: 3, acc:2, af: 2, afa: 2, b: 4, ba: 4, bac: 3, bc: 3, c: 3, cc: 2, f: 2, fa: 2}. The constructed PLWAP tree is as given in Figure 4.

Step 3: Construct FSP-tree and insert all FP_{B_1} into FSP-tree without pruning any items for the first batch B_1 to obtain Figure 5. When the next batch, B_2 of stream sequences, (abacg, abag, babag, abagh) arrives, the number of stream records in the batch is used to update the total number of stream records in the database to 8. The SSM algorithm then proceeds with updating the D-List using the stream sequences. After updating the D-List, it deletes all elements not meeting the minimum tolerance support of $|D|$ * e (or 8 * 0.25 = 2) from the D-List, and keeps all items with $|D| * (s-e) = 8 * 0.5 = 4$ frequency counts in the $L_{1_{B_2}}$. The $L_{1_{B_2}}$ is then used to compute the frequent sequence FS_{B_2}, which is used to construct $PLWAP_{B_2}$, mined to generate the FSP_{B_2}. This FSP_{B_2} is used for obtaining frequent sequential patterns on demand. This process continues with incoming batches like B_3 with stream sequences (abcg, aegd, abfag, afeg) to give the final frequent sequential patterns of FP = {a:12,aa:8, ab:10, aba:8, abg:7, ag:9, b:10, ba:8, bg:7, g:9}.

Table 1. Execution Times (Sec) of SSM-Algorithm and FP-Stream at s=0.0045 and e= 0.0004

	SSM-Algorithm		*FP-Stream Alg*	
Dataset	Average	Total	Average	Total
Dataset	CPU time	CPU time	CPU time	CPU time
	per batch	per batch	per batch	
T10K	4.85	9.7	7.25	14.5
T20K	4.4	18.03	6.5	26
T40K	4.37	35.01	5.75	46
T60K	6.25	75	11.66	139.92
T80K	5.69	91.04	10.56	168.99

Table 2. Execution Times (Sec) of SSM-Algorithm and FP-Stream at s=0.0035 and e=0.0003

	SSM-Algorithm		*FP-Stream Alg*	
Dataset	Average	Total	Average	Total
Dataset	CPU time	CPU time	CPU time	CPU time
	per batch	per batch	per batch	
T10K	6.06	12.12	10.56	21.12
T20K	6.81	27.27	11.77	47.08
T40K	6.9	55.27	12.55	100.4
T60K	7.0	84.02	12.44	149.28
T80K	6.93	111.01	12.23	195.68

4 Experimental and Performance Analysis

This section reports the performance of proposed SSM algorithm. SSM is implemented in Java. The experiments are performed on a 2.8 GHz (Celeron D processor) machine with 512 MB main memory. The operating system is Windows XP professional. The datasets are generated using the publicly available synthetic data generation program of the IBM Quest data mining project at: http://www.almaden.ibm.com/software/quest/. A data loader program is incorporated with SSM to load streams of datasets into the Buffer from the source. The loader loads a transaction, waits 1 ms and then loads the next transaction. The parameters for describing the datasets are: [T] = Number of transactions; [S] = Average Sequence length for all transactions; [I] = Number of unique items. For example, T20K;S5;I1K represents 20000 transactions with average sequence length of 5 for all transactions and 1000 unique items.

The test was performed by running a series of experiments using five different datasets (T10K;S3;I2K, T20K;S3;I2K, T40K;S3;I2K, T60K;S3;I2K, T80K;S3;I2K). It can be seen that the sizes of the 5 test datasets increased from 10K, 20K, 40K, 60K and 80K for two thousand unique items and average sequence length of 3. User defined support is set at 0.0045 (.45%) for a minimum support error e, of 0.0004(0.04%). The performance analysis showing the execution times of the proposed SSM Algorithm in comparison with the FP-Stream algorithm on the above datasets is summarized in Table 1.

For testing, the support was lowerered to 1% because there are no items in the datasets that have support of over 1%. From the experimental results in Table 1, it can be seen that SSM requires less time than FP-Stream because SSM-Algorithm uses PLWAP-tree structure and PLWAP-Algorithm to generate patterns, and thus, does not require to construct intermediate trees to mine frequent sequential patterns. For this reason, FP-Growth requires more storage, more computation time than PLWAP. For both algorithms, the average time of batches varies from batch to batch. It does not go higher constantly. We can say that average time of a batch is dependent on the data of the datasets. It is not related to the size of the datasets. In this experiment a batch is holding approximately 5000 transactions. A number of experiments similar to the one in Table 1 at a minimum support of less than 1% were run on the datasets and the result of a second experiment on the same datasets but at a minimum support s of 0.0035 (0.35%) and error e of 0.0003 (0.03%) is shown in Table 2.

From the tables and for both algorithms, it can be seen that the computation times increase with decreasing minimum support because more items will be frequent, making the trees to be mined, bigger and finding more frequent sequential patterns. An experiment was also run on the three algorithms, PLWAP, FP-Stream and the newly proposed algorithm, SSM-Algorithm on a sample data with the purpose of confirming the correctness of the implementation of the SSM-Algorithm. The dataset had 4 transactions (Tid, Sequence) as (100, 10 20 30 40), (200, 10 20 40 30), (300, 20 30 50 60), (400, 20 10 70 30). Note that although PLWAP algorithm is for sequential mining, it does not mine stream sequences but SSM does and although, FP-Stream algorithm mines frequent streams patterns, it does not mine frequent stream sequential patterns For this reason, our implementation of the FP-Stream found more patterns than are found by both the SSM and the PLWAP because of the different natures of frequent sequential stream miner and frequent sequential miner. PLWAP and the SSM algorithm found the same frequent sequential patterns of ((10), (20), (20,30), (30)). As PLWAP is already an established algorithm and the result of SSM matches with that of PLWAP, this confirms that although the SSM-Algorithm was processing streams of data, it processed them correctly and computed the frequent sequential patterns.

5 Conclusions and Future Work

SSM-Algorithm is proposed to support continuous stream mining tasks suitable for such new applications as click stream data. It is a complete system that fulfills all of the requirements for mining frequent sequential patterns in data streams. SSM-Algorithm can be deployed for mining E-commerce's click stream data. Features of the SSM-Algorithm include use of (1) the D-List structure for efficiently storing and maintaining support counts of all items passing through the streams, (2) the PLWAP tree for efficiently mining stream batch frequent patterns, and (3) the FSP tree for maintaining batch frequent sequential patterns. The use of the support error *e* serves to reduce on irrelevant use of memory for short-memoried stream applications. Experiments show the SSM algorithm produces faster execution times than running the FP-Stream on similar datasets. Future work should consider using sliding window techniques on the SSM to create batches and candidate 1-sequences. Sliding window method may save batch creation and candidate generation time. SSM-Algorithm generates frequent sequential patterns based on frequency of items. It is possible to add multiple dimensions (e.g. time dimension) or constraints along with frequency to discover interesting patterns in data streams. It is also possible to incrementally update the PLWAP tree during the mining of each batch rather than dropping and re-creating.

References

[Chen et al., 2002] Chen, Y. and Dong, G. and Han, J. and Wah, W.B. and Wang, J. Multidimensional regression analysis of time-series data streams. proceedings of the 28th VLDB conference, pages 323-334, Hong Kong, China.

[Domingo & Hulten, 2000] Domingos, P. and Hulten, G. 2000. Mining high-speed data streams. proceedings of the 2000 ACM SIGKDD Int. Conf. knowledge Discovery in Database (KDD'00), pages 71-80.

[Ezeife & Lu, 2005] Ezeife, C.I. and Lu, Yi. 2005. Mining Web Log sequential Patterns with Position Coded Pre-Order Linked WAP-tree. the International Journal of Data Mining and Knowledge Discovery (DMKD), Vol. 10, pp. 5-38, Kluwer Academic Publishers, June

[Ezeife et al., 2005] Ezeife, C.I. and Lu, Yi and Liu, Yi. PLWAP Sequential Mining: Open Source Code paper. proceedings of the Open Source Data Mining Workshop on Frequent Pattern Mining Implementations in conjunction with ACM SIGKDD, Chicago, IL, U.S.A., pp. 26-29.

[Giannella et al., 2003] Giannella, C. and Han, J. and Pei, J. and Yan, X. and Yu, P.S. 2003. Mining Frequent Patterns in Data Streams at Multiple Time Granularities, in H. Kargupta, A. Joshi, K. Sivakumar, and Y. Yesha (eds.), Next Generation Data Mining, 2003.

[Guna et al., 2000] Guna, S. and Meyerson, A., and Mishra, N. and Motwani, R. Clustering data streams:Theory and Practice. TKDE special issue on clustering, vol 15.

[Gunduz & Ozsu, 2003] Gunduz, S. and Ozsu, M.T. A web page prediction model based on click-stream tree representation of user behavior. SIGKDD, Page 535-540.

[Han et al., 2004] Han, J. and Pei, J. and Yin, Y. and Mao, R. 2004. Mining frequent patterns without candidate generation: a frequent pattern tree approach. Data Mining and Knowledge Discovery, 8, 1, Page 53-87.

[Las, 2002] Last, M. 2002. Online classification of nonstationary data streams. Intelligent Data Analysis, Vol. 6, No. 2, Page 129-147.

[Manku & Motwani, 2002] Manku, Gurmeet Singh. and Motwani, Rajeev. 2002. Approximate frequency counts over data streams. proceedings of the 28th VLDB conference, Hong Kong, China.

[Pei et al., 2000] Pei, Jian and Han, Jiawei and Mortazavi-asi, Behzad and Zhu, Hua. 2000. Mining Access Patterns Efficiently from web logs. Proceedings 2000 Pacific-Asia conference on Knowledge Discovery and data Mining,Pages 396-407, Kyoto, Japan.

[Pei et al., 2001] Pei, J. and Han, J. and Mortazavi-Asl, B. and Pinto, H. and Chen, Q. and Dayal, U. and Hsu, M.C. 2001. PrefixSpan: Mining Sequential Patterns Efficiently by Prefix-Projected Pattern Growth. In Proceedings of the 2001 International Conference on Data Engineering (ICDE'01), Heidelberg, Germany, pages 215-224.

[Srikanth & Aggrawal, 1996] Srikanth, Ramakrishnan and Aggrawal, Rakesh. 1996. Mining Sequential Patterns: generalizations and performance improvements. Research Report, IBM Almaden Research Center 650 Harry Road, San Jose, CA 95120, Pages 1-15.

[Teng, Chen & Yu, 2003] Teng, W. and Chen, M. and Yu, P. 2003. A regression-based temporal pattern mining scheme for data streams. In proceedings of the 29th VLDB conference, Berlin, Germany.