

A PLWAP-BASED ALGORITHM FOR MINING FREQUENT SEQUENTIAL STREAM PATTERNS

Christie I. EZEIFE¹, Monwar MOSTAFA¹

¹*School of Computer Science,
University of Windsor, Windsor, Canada, N9B 3P4*
E-mail: cezeife@uwindsor.ca, <http://www.cs.uwindsor.ca/~cezeife>

Abstract

The challenges in mining frequent sequential patterns on data stream applications include contending with limited memory for unlimited data, inability of mining algorithms to perform multiple scans of infinitely flowing original stream data set, to deliver current and accurate result on demand. Recent work on mining data streams exist for classification of stream data, clustering data streams, mining non-sequential frequent patterns over data streams, time series analysis on data stream. However, only very limited attention has been paid to the problem of stream sequential mining in the literature. This paper proposes SSM-Algorithm (Sequential Stream Mining-algorithm) based on the efficient PLWAP sequential mining algorithm, which 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 continuously summarizes frequency counts of items with the D-List, builds PLWAP tree and mines frequent sequential patterns of batches of stream records, maintains mined frequent sequential patterns incrementally with FSP tree. Capabilities to handle varying time range querying, transaction fading mining, sliding window model are also added to the SSM-algorithm.

Keywords: Web Sequential Mining, Stream Mining, Customer Access Sequence, Frequent Sequential Patterns, Buffer, Click Stream Data.

1 Introduction

When a user visits a web site, the web server usually keeps some important information about the user in the web log file, such as, the client's IP address,

the URL requested by the client, and the date and time for that request [18]. The three types of log files available on a web server are server-logs, error-logs, and cookie-logs. Analyzing data on the web server-logs is called “click stream analysis”. E-commerce or e-business can benefit from this click stream analysis. Click stream analysis provides important information to understand better the marketing and merchandising efforts, for example, how customers find the store, what products they search for, and what products they purchase. Some uses of such stream applications like web log click stream analysis include: improving the web site by better understanding the users’ interests, typical user navigational paths, and the correlation between customer behavior and the products. Yang et al. [24] suggest an interesting method to reduce network latency by applying web mining techniques. The method finds frequent access patterns from web logs by analyzing click stream data. Once frequent access pattern is found, it can be used in web caching and pre-fetching system in order to improve the hit rate and reduce unnecessary network traffic.

A data stream is a continuous, unbounded, and high-speed flow of data items. Applications generating large amounts of data streams, include web logs and web click streams, computer network monitors, traffic monitors, ATM transaction records in banks, sensor network monitors, and transactions in retail chains. Mining data in such applications is 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 data set more than once, as the whole data set 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 for customer buying interest in an E-commerce site, 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.

Data stream can be either offline streams, characterized by regular bulk arrivals or online streams characterized by real-time updated data [13, 7]. The four main stream data processing models are:

1. The landmark model, which mines all frequent itemsets over the entire history of the stream from a specific beginning time point called the landmark to the present.
2. The damped model (also called the time-fading model), which has a

weight attached to each transaction, where the transaction weight decreases as the entry time of the transaction into the stream gets older. For example, while a record that joins the stream at time 00:00 may have a decreasing weight of 0.2, a record that has just joined the stream at the processing time of 16:00 may have a higher weight of 1. This means that older transactions contribute less weight toward itemset frequencies. This is suitable for applications where old data has a fading effect on frequency as time goes on and may be effective for garbage collection of records.

3. The sliding windows model, which finds and maintains frequent itemsets in sliding windows and only the new data streams within the current sliding window are processed. For example, assume a window size of 4 records and a stream of twelve records S_1, \dots, S_{12} , in sliding window SW_1 , contains records S_1, \dots, S_4 , sliding window SW_2 contains records S_2, \dots, S_5 , while window SW_3 has records S_3, \dots, S_6 and so on.
4. Batch Window model, which finds and maintains frequent itemsets in batch window. All records in the batch window are removed once the window is out of range. Batch window is a special case of sliding window model where there is no overlap in transactions in consecutive windows. For example, assume a window size of 4 records and a stream of twelve records S_1, \dots, S_{12} , in batch window BW_1 , there are records S_1, \dots, S_4 , batch window BW_2 contains records S_5, \dots, S_8 , while batch window BW_3 , has records S_9, \dots, S_{12} and so on

Three classifications of web mining are:

1. Web content mining for extracting meaningful information from contents of web pages presented in HTML or XML file formats.
2. Web Structure Mining for discovering interesting patterns from hyperlinks structures from mining in-going and outgoing links from web pages.
3. Web Usage Mining for discovering interesting patterns in large access-log files recording information about web users and their navigation sites.

All users' behaviors on each web server can be extracted from the web log. The web log records are generally pre-processed and cleaned to transform its records in the form of a transactional database with schema (Transaction id, access sequences) before mining to generate frequent patterns. The format

Table 1. Sample Web Access Sequence Database

TID	Web access Sequences
100	abdac
200	abcac
300	babfae
400	afbafcg

and example of a line of data in a web log before pre-processing is: host/ip user [date:time] “request url” status bytes
137.207.76.120 - [30/Aug/2001:12:03:24 -0500] “GET /jdk1.3/docs/relnotes/deprecatedlist.html HTTP/1.0” 200 2781 where the host ip address of the computer accessing the web page (137.207.76.120), the user identification number (-)(this dash stands for anonymous), the time of access is (12:03:24 p.m. on Aug 30, 2001 at a location 5 hours behind Greenwich Mean Time (GMT)), request is (GET /jdk1.3/docs/relnotes/deprecatedlist.html) (other types of request are POST and HEAD for http header information), the unified reference locator (url) of the web page being accessed is (HTTP/1.0), status of the request (which can be either 200 series for success, 300 series for re-direct, 400 series for failure and 500 series for server error), the number of bytes of data being requested (2781). Since some data may not be available in web log, to obtain complete web usage data source, some extra techniques, such as packet sniffer and cookies may be needed. The stream sequential mining operations being proposed in this paper starts with a transaction data set form of the web log obtained after pre-processing web log. A busy website generates a huge amount of click stream data everyday. Each click stream data series 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. An example processed web log data representing only four sequences of access to product sites labeled “a, b, c, d, e, f, g” is given in Table 1.

1.1 Problem Definition

Given the set of events or items $E=\{e_1, e_2, \dots, e_n\}$, standing for retail store items like bread, butter, cheese, egg, milk, sugar or web pages, a stream sequential database, SSDB, is an infinitely flowing sequence of batches (B_1, B_2, \dots) of sequential records (each record is a sequence of events $e_1 e_2 \dots e_m$) in

the schema (transaction id, record sequence). The problem of stream sequential mining requires finding all frequent sequential patterns, FP, having support greater than or equal to a user given minimum support ($s\%$) from beginning of a stream batch (B_i) to current batch (B_c) time, given one pass of stream sequential database SSDB as a continuous stream of batched records B_1, B_2, \dots and a tolerance error support (ε) much lower than the minimum support $s\%$ for allowing potentially frequent items to have their counts maintained in the D-List structure.

When the initial batch for mining is the very first batch of the stream database, the mining model is the landmark model. When a number of record overlap is allowed between consecutive batches, the model is sliding window model. When aging of records of older batches is considered in the frequency count, it is the damped model and when the initial or final batch is any batch, the algorithm would compute frequent patterns for sequences between any arbitrary range of batches of the stream. batch. This paper would present a solution for the batched landmark model first and presents extensions to support sliding window, damped model and varying batch frequent pattern requests.

A batch B_i of stream sequential records consists of q sequential records S_1, S_2, \dots, S_q . For example, Table 1 represents a batch with four sequential records. The length of a sequence, $|S|$ is the number of events in the sequence and a sequence with length j is called a j -sequence. For example, while babfae is a 6-sequence record, abdac is a 5-sequence record. In access sequence $S = e_1 e_2 \dots e_k e_{k+1} \dots e_m$, if subsequence $S_{suffix} = e_{k+1} \dots e_m$ is a super sequence of pattern $P = e'_1 e'_2 \dots e'_l$, where $e_{k+1} = e'_1$, then, $S_{prefix} = e_1 e_2 \dots e_k$, is called the prefix of S with respect to pattern P , while S_{suffix} is the suffix sequence of S_{prefix} . For example, in sequence babfae, ba is a prefix subsequence of bfae, while bfae is a suffix of ba. A frequent pattern in a database or a set of q sequential records is a sequence that has occurrence count (called the pattern's support) in the number of records, which is greater than or equal to the given minimum support, s . Thus, the support of a sequence is the number of records it appears in as a subsequence divided by the number of records in the stream up to the time of measure. For example, if the sequential database of Table 1 is batch B_1 and we want to compute frequent patterns when the minimum support threshold is 75% or 3 out of 4 records, then, it can be seen that frequent 1-sequences with their support counts include $\{a:4, b:4, c:3\}$ and some frequent 2-sequences and 3-sequences patterns are $\{ab:4, aa:4, ac:3, ba:4, aba:4, bac:3, \dots\}$.

1.2 Related Work

Han et al. in [12] proposed the FP-tree algorithm to generate frequent pattern itemsets. Frequent sequential pattern mining algorithms include the GSP [22], which is Apriori-like, PrefixSpan [20], a pattern-growth method, the WAP [19], which is based on the FP-tree but used for mining sequences, the PLWAP [4, 5], 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. An example mining of a batch with the PLWAP tree is presented as EXAMPLE 2. This structure is chosen for stream sequential mining, in order to take advantage of its compact structure and speed, in stream environment. Data stream algorithms include the Lossy counting algorithm [17], which is used for frequency counting in data streams. Previously, landmark model [17] 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 Lossy counting algorithm finds frequent items in non-sequential data streams with a maximum support s and maximum error $\epsilon \ll s$. The algorithm suggests an $\epsilon = 10\%$ of s . The Lossy counting algorithm works with a full fixed sized buffer of n stream records, which are placed in a number m of buckets. Each bucket contains $1/\epsilon$ records. Then, the Lossy counting algorithm runs on buckets one after the other to find frequent items. It stores the ϵ -frequent items (having frequency more than ϵ) in a D data structure in the form of (element, frequency, maximum error count of element). Whenever a new element arrives, it increments its frequency in the D structure by 1 if the element is already in the D structure, but inserts a new record (element, 1, current bucket id - 1) if element is not found. At the boundary of the bucket or window, it performs two types of pruning (1) it decrements all elements in D by ϵ 's value, and (2) it deletes items from the D structure if their frequency +

maximum error count \leq current bucket-id. If a trimmed item comes back later, the algorithm compensates an approximate loss frequency. Whenever a user requests a list of items that have support s , the algorithm generates output for item with frequency $f > (s - \epsilon) * \text{current length of stream}$. While the Lossy counting algorithm uses decrement mechanism to efficiently eliminate small items with counts less than or equal to ϵ , this method prevents all items including frequent items from maintaining their true counts. Also, this algorithm is not suitable for mining sequential stream pattern mining. Our technique is thus different from this system in the sense that we aim at producing more complete patterns and to work on sequential and not non-sequential patterns. Thus, while we use a modified form of the D structure for counting item frequencies, our D-List structure increments rather than decrements counts and is hash-based for speed improvement. The mining efficiency is further improved on with the PLWAP and FSP sequential mining and pattern storing structures.

FP-Stream algorithm [8] is used to mine frequent patterns in data stream. The authors of FP-Stream also extended their framework to answer time-sensitive queries over data stream. FP-stream algorithm uses a pattern-tree, FP-tree and a fixed tilted time window frame to handle time-sensitive stream queries. The idea of the natural tilted time window used for holding up to one month's data or 31 days data is that one month's data is partitioned into 59 units, where the first four units (1 hour data) are in 15 minute units each, next 24 units (for 1 day data) are in one hour units each, and the last 31 units (for one month data) are in 1 day unit each. Thus, if for example, five itemsets with their frequency counts as a:48, b:50, c:45, ab:27, ac:29 are stored in tilted time window t_0 (standing for the most recent 15 minutes data), the itemsets stored in the next tilted time window t_1 (standing for the most recent 30 minutes data) will again include all the t_0 patterns and so on as most recent patterns are shifted down from the right more specific tilted time window to the left more general tilted time window. With this model, it is possible to compute frequent patterns in the last hour with the precision of quarter of an hour, for last day with the precision of its hours, for the whole month with the precision of its days. However, to keep a whole year's data will require keeping $366 (\text{number of days}) * 24 (\text{hours}) * 4 (15 \text{ minutes}) = 35,136$ units, which is huge and they proposed the Logarithmic tilted time window frame to cut down on the number of needed time units to 17 as $\log_2(366 * 24 * 4) + 1$. The FP-stream algorithm forms a batch B_i every 15 minutes, computes all item frequencies and stores them in main memory in descending order of support in f-list. Then, it constructs an FP-tree, pruning all items with frequency less than $\epsilon * |B_i|$. Then, it finds ϵ -frequent itemsets from the FP-tree, which it stores in the pattern tree. While the FP-stream is able to

retrieve and answer queries over a wide range of time, it requires big storage for keeping short term period data in order to maintain several time slots. The FP-stream is only for non-sequential data. Our approach unlike the FP-stream, uses PLWAP, hash based D-List for frequency counting and FSP for storing results and avoids repetitive storage of patterns like the FP-stream.

Teng et al. proposed FTP-Algorithm [23] 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. While most existing data stream systems are for mining non-sequential records, two recent sequential data stream systems proposed are SPEED [21] and SSM [6]. The SPEED algorithm mines maximal sequential patterns of an arbitrary time interval of the stream by inserting each sequence of an arriving batch in a region of a tree-like data structure called Lattice and later it will extract maximal subsequences from the lattice structure. In addition to the lattice storing maximal sequences, it stores a table of information about the set of items, sequences, their lengths as their sizes and the tilted-time window they belong to, as well as the region of the lattice tree they belong to and their roots. SPEED prunes unfrequent sequences by dropping the tail sequences of the tilted-time windows supports that are less than desired batch supports. The SSM algorithm [6] presents a draft stream mining algorithm, which is being extended by this paper. The main extensions provided by this paper are mining for more than the basic landmark model, more extensive literature review, capabilities to handle sliding window, damped model and arbitrary batch range querying, as well as more experimentation of the system.

Other studies on mining data streams exist for classification of stream data [3], online classification of data streams [14], clustering data streams [10], web session clustering [11], approximate frequency counts over data streams [17], mining frequent patterns in data stream at multiple time granularities [8], and multi-dimensional time series analysis [2], temporal pattern mining in data streams [23] but more work is needed on mining frequent sequential patterns in data streams.

1.3 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 flexible and efficient data structures D-List and FSP-tree. The structures allow the flexibility of extending the basic solution model provided

for landmark model to handle damped, sliding window and computation of frequent patterns between any adhoc ranges of stream batches.

1.4 Outline of the Paper

The balance of the paper has section 2 presenting details of the proposed sequential stream mining (SSM) algorithm that is based on the PLWAP sequential mining algorithm with some examples. Section 3 presents Experimental and Performance Analysis and section 4 presents conclusions and future work.

2 The Proposed SSM Sequential Stream Mining System

By considering stream environment requirements, we have developed a method that uses SSM-Algorithm (sequential stream mining algorithm) to collect data stream into a buffer from web applications, forms dynamic sized batches by taking data stream from the buffer and mines each batch to deliver results. A batch is a group or a number of customer access sequences. SSM-Algorithm maintains three data structures (D-List, PLWAP-tree, FSP-tree) in order to handle and generate result for click stream data. Click stream data can be generated from the clicks of users on the web. Each data structure has its own algorithm for updating and retrieving data from the structures. D-List is a hash chain based data structure that stores all incoming items' ID and their frequency if they are above the tolerance error threshold ϵ . D-List is very efficient when there are huge numbers of items that are used at the E-commerce site. Brand new items get posted to the E-commerce site very often. It is very realistic that each E-commerce site introduces new items as soon as they get items from the vendors. We assume that we do not have any information on number of items and their IDs for our algorithm. In other words, our D-List is totally dynamic, it grows with incoming data stream. PLWAP-tree [4, 16] gets constructed by selecting frequent sequences from batches. PLWAP-mining algorithm uses preordered linkage and position coding method to avoid costly reconstruction of intermediate trees to generate frequent sequential patterns unlike WAP-tree [Pei et al.2000]. The process of mining frequent sequences continues batch by batch. Once we find frequent patterns from first batch, we keep incrementing frequent patterns onto frequent sequential pattern-tree (FSP-tree) batch by batch. The construction process of FSP-tree is similar to Pattern-tree that was introduced in [8] but the structure of FSP-tree is simpler than pattern-tree. The FSP-tree is made simpler for efficiency and speed. FSP-

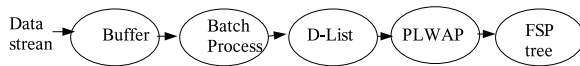


Figure 1. The Main Components of the SSM Stream Miner

tree stores frequent patterns instead of customer access sequences. Whenever we need to view the results, we are able to generate updated frequent patterns from the FSP-tree. The proposed method is very memory efficient and ideal for click stream environment. As the mining process moves only forward and there is no way to go backward and rescan previous data stream, therefore we do not keep any items that have support count less than maximum support error threshold in D-List structure where updated candidate 1-sequences and their counts are stored. The items less than maximum support error threshold have very small count and chances are very slim for them to become large items later. Therefore, we get rid of those small items from the beginning. However, those small items that are potentially large items (very close to large items) because they have support greater than or equal to maximum support error, are all kept in the D-List. Maximum support error is a predefined threshold that gives a tolerance of error in the result to allow for approximate, near accurate mining that produces all correct frequent patterns with possibility of a few potentially frequent patterns due to the use of tolerance error support. It is obvious that if maximum support error is very small compared to user defined minimum support, then the error in the result will be very nominal. By keeping all items that have support more than or equal to maximum support error in the D-List, we assure that our result does not cross error boundary.

2.1 Architecture and Components of the Proposed SSM System

The main components of the SSM (Sequential Stream Mining) system is depicted in Figure 1.

Step 1: Buffering of Arriving Data Streams: 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 [17], uses buffer mechanism to deal with incoming data stream. On the other hand, FP-Stream [8] uses main memory to handle data stream. 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 storage, 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 incoming stream. The batch size of the SSM system is not fixed, unlike Lossy Counting Algorithm [17], or FP-Stream [8]. For extension to accommodate sliding window model with o number of overlapping records between two consecutive windows, SW_i and SW_{i+1} , the SSM system will maintain a global record counter *record-id*. The batching engine unlike the basic batching model will retain the last o records of earlier sliding window SW_i deleting this window's first $b - o$ records, where b is the total number of records in new sliding window being formed, SW_{i+1} and the system inserts as the last records of the new window being formed $b - o$ records. For example, if records 1 to 4 have been in the first sliding window, SW_1 and the number of overlapping records is 2, then, in forming the sliding window 2, SW_2 , the SSM system should delete the first $(4 - 2) = 2$ records of SW_1 and accept new $(4 - 2) = 2$ records for new window SW_2 such that the records to be processed in the new SW_2 window are records 3, 4, 5, 6. To handle the damped model, the batching model would create either regular batches or sliding window batches as described above and the D-List can be modified to remember counts of items of last two or three batches such that fading weights of older item counts can be used to affect the overall frequency counts when updating the D-List.

Step 3: D-List data structure for maintaining counts of frequent items: SSM system uses three data structures: D-List, PLWAP-tree and FSP-tree. D-List

structure is used for efficiently managing the frequency counts of 1-items and for obtaining the updated frequent 1-items list L_{1B_i} at the end of each batch, B_i . The 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. The SSM system 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_{1B_i} . Only items with frequent support count 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, ε , are kept in the D-List. While items with support counts less than this value are deleted from the D-List, those with counts greater than or equal to the total number of stream records times (minimum support (s) minus tolerance error (ε)) are kept in the L_{1B_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.

EXAMPLE 1: 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 ε , is 0.25 (or 25%). Construct and maintain the support counts of the items using the D-List structure and compute the set of frequent 1-item lists L_{1B_1} and frequent sequence records FS_{B_1} of batch 1 stream records for mining.

SOLUTION 1: Since this is the first batch with only 4 records, the tolerance maximum 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_{1B_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_{1B_1} list. The $L_{1B_1} = \{a:4, b:4, c:3, f:2\}$. The D-List after reading the batch B_1 is shown as Figure 2. 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_{1B_1} has been used to update the D-List, which was used for computing

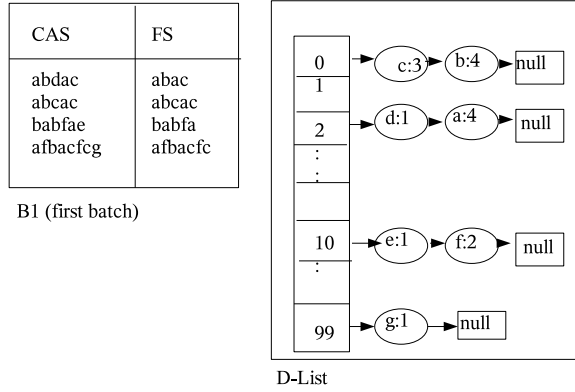


Figure 2. The D-List After Batch B_1

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

Extension to be made to the D-List processing structure to accommodate the sliding window model, which for each new window, SW_{i+1} of size b records and with o overlapping records is to insert only the window's last $(b - o)$ records into the D-List. This is because the first o records had already been inserted when processing records of the previous window, SW_i . To handle the damped model, the D-List can be modified to remember counts of items of last two or three batches so that fading weights of older counts can be reflected on the overall frequency counts of the items when updating the D-List. For example, while the weight of all current batch items will be rated at 100% of their counts, the weight of last batch items can be rated at 50%, while the weight of last two batches items is rated at 25% and any older batch items are rated at 0%. The D-List can be updated such that three counts are listed for each item and each item node appears with the item label: item current count: item last batch count: item last two batch count. An example D-List item node will then look like a:12:5:1 meaning that item a now has total count of 12 after the current batch, but had a total count of 5 after the last batch and a total count of 1 after the last two batches. With an a node labeled as above, the actual weighted frequency count is computed by discarding all counts of this item that occurred before the last two batches. Since the current batch has only a count of 3 for item a , it can be seen that previous counts of this item prior to its counts form the last two batches is $12 - (5 + 4) = 4$. Thus, this count of 4 is taken out of the total count of 12 to have 8, which are weighted

as: $(3 * 100\%) + (4 * 50\%) + (1 * 25\%) = 3 + 2 + .25 = 5$. Thus, for item a the used frequency for $3 + 2 + .25 = 5$. Thus, for item a the used frequency for $3 + 2 + .25 = 5$. Thus, for item a the used frequency for deciding frequent 1-items will be the computed weighted frequency.

Step 4: PLWAP Sequential Mining Data Structure for mining current batch frequent sequences: PLWAP-tree or Pre-ordered Linked WAP-tree was introduced in [4], [5]. The basic idea behind the PLWAP tree algorithm is using position codes and pre-order linkage on the WAP-tree [19] 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. Generally, the PLWAP miner starts with a frequent 1-item list, L_{1B_i} , which had already been formed in the D-List stage. Next, for each sequence in the current batch, it creates a set of frequent sequences FS_{B_i} of the batch by deleting all items in the sequences that are not frequent or not in the L_{1B_i} list. It now builds the PLWAP tree by inserting each frequent sequence S_i from FS_{B_i} top-down from Root to leaf node. PLWAP tree is 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_{1B_i} list is used to construct pre-order header linkage nodes for mining. Each node has the item node label: its count: its position code. The tree maintains frequent 1-item header linkage for quick mining by traversing the tree preorder fashion (visit root, visit left node, visit right node) and linking the order of node of the frequent 1-item type on the tree. For example, all nodes of label a will be linked with a dashed a -frequent header linkage starting from frequent header linkage item a . 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 tolerance support count (number of records $* (s - \epsilon)$), then, this item is confirmed frequent and appended to the previous prefix frequent stream sequence.

EXAMPLE 2: Given the set of frequent sequences for batch 1, FS_{B_1} con-

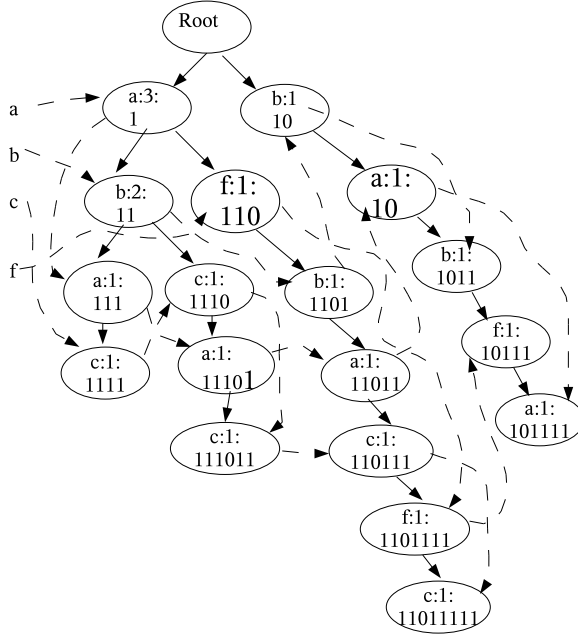


Figure 3. The PLWAP Tree of Batch B_1

constructed from EXAMPLE 1 above as $\{abac, abcac, babfa, afbacfc\}$, as well as the frequent 1-items $L_{1_{B_1}} = \{a:4, b:4, c:3, f:2\}$, construct the $PLWAP_{B_1}$ tree and mine the tree to generate frequent stream sequential patterns for this batch FP_{B_1} .

SOLUTION 2: The constructed PLWAP tree is as given in Figure 3.

The leftmost branch of this tree shows the insertion of the first sequence $abac$ when the first a and b nodes initially had a count of ‘1’ each. Then, when the second sequence $abcac$ is inserted, the leftmost branch top nodes counts and position codes became $a:2:1$ and $b:2:11$. Since the next child of this b node is not c , then, a new child node of b is created for the suffix sequence cac as shown in the figure. The remaining sequence 3 and 4 of the batch are inserted into the PLWAP tree in a similar fashion. Then the tree is traversed pre-order way to link the frequent 1-item header nodes $L_{1_{B_1}} = \{a:4, b:4, c:3, f:2\}$ to the PLWAP tree nodes using the dashed arrow headed links. For example, following the left subtree of “Root”, we find an $a:3:1$ node that is linked to the frequent header of its kind ‘a’, then the next $a:1:111$ node found during pre-order traversal has a link from the first linked ‘a’ node to itself. The idea is to trace all a nodes on the tree by following the dashed links from the ‘a’ frequent header

link and this is used with the position code to quickly mine this tree without needing to construct intermediate conditional pattern bases and trees. Next operation is to mine this constructed PLWAP tree for frequent patterns meeting a support threshold greater or equal to minimum tolerance support of $s - \varepsilon$ or 0.50. 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 $((s - \varepsilon) * |B_1| = 0.50 * 4 = 2)$. The PLWAP mining method starts from PLWAP Root to examine subtree suffix root sets and extract all patterns with total count on all different branches in the suffix root sets at that height of the tree, greater or equal to minimum tolerance support. For example, the left branch rooted at a:3:1 and right branch rooted at b:1:10 form the first suffix root sets. It can be seen that a:3:1, a:1:10 mean that pattern a has a total count of 4 and is frequent. Then, ab:2:11, ab:1:1101 and ab:1:1011 mean ab has a count of 4 and is frequent. Also, aba:1:111, aba:1:11101, aba:1:11011 and aba:1:101111 mean a count of 4 for pattern aba . The algorithm uses position codes of nodes to quickly determine if they are on different branches of the tree and their counts can be added. More details regarding the PLWAP algorithm can be found in [4] and [5] and the source codes of the algorithms are generally available through the author's web sites. The found $FP_{B_1} = \{a:4, aa:4, aac:3, ab:4, aba:4, abac: 3, abc: 3, ac: 3, acc:2, af: 2, afa: 2, b: 4, ba: 4, bac: 3, bc: 3, c: 3, cc: 2, f: 2, fa: 2\}$.

To accommodate both the sliding window model and the damped model, no changes need to be made to the PLWAP construction and mining as the frequent 1-item list and the current batch sequences are all that are needed to build and mine the current batch frequent patterns and to save memory, the current batch tree is discarded once the patterns are mined and stored in the FSP result structure. An improvement at this stage of the system is using the incremental version of the PLWAP algorithm [Ezeife & Chen, 2004a, Ezeife & Chen, 2004b], rather than discarding the tree, but the benefits of such an extension can be investigated for future work and would work only if there is need to perform cumulative mining on the same tree without using a separate structure to store mined patterns cumulatively.

Step 5: The FSP Result Structure for Storing mined frequent patterns: Frequent Sequential Pattern-tree or FSP-tree is a simple form of Pattern-tree [8] for storing result structure. The sequences and their counts are simply inserted or updated 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

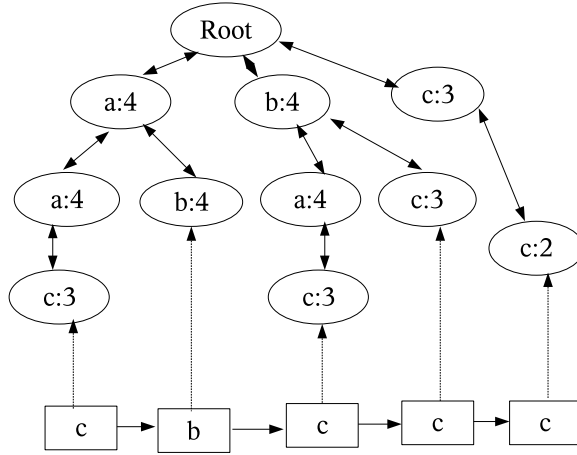


Figure 4. The FSP Tree After Batch B_1

sequential patterns.

EXAMPLE 3: Given the found frequent patterns from batch 1 from EXAMPLE 2 above, $FP_{B_1} = \{a:4, aa:4, aac:3, ab:4, aba:4, abac: 3, abc: 3, ac: 3, acc:2, af: 2, afa: 2, b: 4, ba: 4, bac: 3, bc: 3, c: 3, cc: 2, f: 2, fa: 2\}$, save this in the compact frequent sequential pattern tree FSP tree for cumulative storage and retrieval of mined patterns.

SOLUTION 3: The constructed self-explanatory FSP-tree which inserts all FP_{B_1} with their counts into FSP-tree without pruning any items for the first batch B_1 is given in Figure 4. The patterns are inserted in a similar way the sequences are inserted in the PLWAP tree from Root to leaf node, sharing common nodes and incrementing node counts by the count of the pattern when appropriate. For example, patterns $a:4, aa:4, aac:3, ab:4$ are all inserted and can be retrieved from the leftmost branch of Root. If after the next batch mining, also a pattern $a:4$ is mined, then, the $a:8$ nodes is now the updated node of a :node.

Step 6: Extracting Frequent Patterns from the FSP tree: The FSP is good for quickly extracting both maximal frequent patterns and all frequent patterns.

EXAMPLE 4: Given the FSP_{B_1} from EXAMPLE 3 above, the total number of records up to the end of this batch, the minimum support s and tolerance error support ε , extract all all frequent sequential patterns from the beginning of stream to end of this batch, FP_{B_1} with support counts $\geq \text{number of records} * s - \varepsilon$.

SOLUTION 4: From the FSP tree of Figure 4, the maximal patterns (longest

patterns on each sequence track) $MaxFP_{B_1}$, with support counts $\geq (s - \epsilon)$ or $4 * (.75 - .25) = 2$ transaction counts are: {aac:3, ab:4, bac:3, bc:3, cc:3} and these can be found through suffix pruning I of the tree using footer list of the leaf nodes described below. Note that because this is the first batch, all patterns on the tree met the extraction condition. Also, since this method allows an error tolerance, it is an approximate mining that finds all frequent patterns but may include some false positive patterns or patterns that are not exactly frequent. The amount of false positive patterns depends on the precision of choice of error tolerance support. It is usually much less than the minimum support count of s but a relatively high value is used in this example to make it understandable.

FSP-tree maintains a footer list, which has linkages to each leaf node of the tree. Footer list is a linked list that grows with the leaves of the tree. The FSP-tree maintains this list in order to read the tree from the leaf instead of root for maintenance purposes of the tree. Like the PLWAP tree, the FSP tree has the property that all the parent nodes have frequency counts higher than or equal to those of their children in FSP-tree. Therefore, if parent node does not have minimum support, its children are ignored during the frequent pattern extraction process. While the mining algorithm searches for frequent patterns or FPs in FSP-tree from root down to leaf node for a particular branch during the journey, if it finds any node with a count less than minimum support, it does not go further down on that branch. It cuts the suffix sequence of the branch from that point in this process we call Suffix Pruning I. Suffix Pruning I is quickly used to trim those branches of the FSP tree from the leaf that have support counts less than $number\ of\ records * \epsilon$ after each batch. On the contrary, Suffix Pruning II allows for milestone pruning of the FSP tree after a number of batches and transactions have passed through the stream for reducing on the size of the tree. One other important role of the FSP tree Suffix Pruning II is that it is used to perform a sequential pruning on the D-List once we are done with suffix pruning II at the boundary. Sequential pruning is the process that checks the frequency of each element in D-List sequentially. If the frequency of elements $f < number\ of\ records * \epsilon$, they are deleted from the D-List. It can be seen that the FSP tree is quick for extracting both maximal frequent patterns (the longest frequent patterns) and all frequent subsequent patterns.

This FSP tree structure does not need to be changed to support sliding window and damped models. However, to enable extracting patterns between arbitrary ranges of batches or times, one solution is to employ the same annotation scheme for node employed in PLWAP and used in our extension to

the D-List, where total counts of the most current batch is listed first, then followed by the last batch etc. For example, if after processing batches B_1 , B_2 , B_3 , the found patterns include a:4 in B_1 , a:8 in B_2 and a:12 on B_3 , the a node is modified in the FSP tree as a:12:8:4. This way we can answer queries like “List patterns from batch B_1 to B_2 ” as those that are frequent in B_1 and B_2 .

Step 7: Maintain FSP, D-List, PLWAP Data Structures for next round mining: For next batch mining, the current PLWAP tree is dropped to free space, the FSP tree is pruned as discussed in step 6 above through either Suffix Pruning I or II. The D-List is pruned through sequential pruning of all items that have support counts less than the number of records multiplied by the error support count of ϵ .

Step 8: Continuous Infinite Processing of Arriving Stream Data: User may exit the system at this point or continue to process incoming streams by going back to Step 1, where stream data are buffered before batching and mining.

2.2 The SSM Sequential Stream Mining Algorithm

The proposed sequential data stream algorithm, SSM is given as Algorithm 1 of Figure 5, which calls sub algorithms 2 and 3 of Figures 6 and 7. Details on all other steps of the main algorithm are discussed in section 2.1.

3 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 data sets 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 data sets 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 data sets 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 data sets (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 data sets increased from 10K, 20K, 40K, 60K and 80K for two thousand unique items and aver-

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) //Fig. 6

 2.1 Scan B_i and generate candidate

 1-sequences or C_{1B_i}

3. Update D-List[Size] with C_{1B_i}

// Fig. 7

4. Generate-Frequent-Pattern(FS_{B_i})

with $PLWAP_{B_i}$ // as in [4]

5. Update-FSP-tree(FP_{B_i})

// Update as explained in step 5 of section 2.1

6. If user wants result, then from FSP-tree,
get all FSP with count $\geq (s-e) * \text{num-records}$

7. Maintain-Structures()

//i.e prune D-List and FSP, drop PLWAP,

// as explained in step 7 of section 2.1

8 If user wants to exit, exit = false;

End

End

Figure 5. The Main Sequential Stream Mining (SSM-Algorithm)

Algorithm 2 (*Batch Creation Program*)

Algorithm Create-Batch()

Input: (1) Stream of m Buffer Customer Access Sequences

Output: 1) a batch of m customer access sequence records

Temp variables: min-CAS, max-CAS

Begin

1. While (min-CAS \leq max-CAS) do
 - 1.1 for $i = 1$ to m sequences do
 - 1.1.1 batch record[i] = buffer record[i]

End

Figure 6. The Batch Creation Program

Algorithm 3 (*D-List Update*)

Algorithm Update-DList()

Input: (1) Hash Array D-List[size] with buckets,
initialized to Null at creation,
set of Candidate 1-items of i th batch $C_{1_{B_i}}$
number of candidate elements q in $C_{1_{B_i}}$

Output: 1) Updated Hash array D-List[size]

Temp variables: m

Begin

1. For each item m in candidate set $C_{1_{B_i}}$ do
 - Begin**
 - 1.1 Find hash bucket in D-List [size] using hash function
(fn) $r = m$ 1.2 If element m of bucket D-List[r] has
count $\geq num_{CAS} * (s - \epsilon)$, then
 $L_{1_{B_i}} = L_{1_{B_i}} \cup m$

Begin

End

Figure 7. The D-List Update Algorithm

Table 2. Execution Times (Sec) of SSM-Algorithm and FP-Stream at $s=0.0045$ and $e= 0.0004$

Dataset	<i>SSM-Algorithm</i>		<i>FP-Stream Alg</i>	
	Average CPU time per batch	Total CPU time per batch	Average CPU time per batch	Total CPU time
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

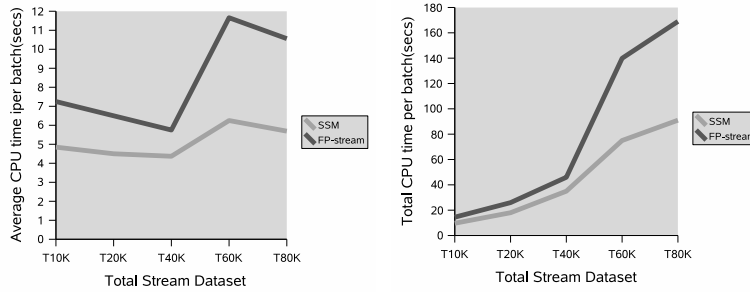


Figure 8. The Average CPU times for $s = .45\%$ and $e = .04\%$

age 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 batch size is set to contain 5000 transactions. This means that runs are on 2 batches to 16 batches. The performance analysis showing the execution times of the proposed SSM Algorithm in comparison with the FP-Stream algorithm on the above data sets is summarized in Table 2 while the chart showing comparisons of average CPU times and the total CPU times used by the two algorithms for this data set is shown as Figure 8.

For testing, the support was lowered to 1% because there are no items in the data sets that have support of over 1%. From the experimental results in Table 2, 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

Table 3. Execution Times (Sec) of SSM-Algorithm and FP-Stream at $s=0.0035$ and $e=0.0003$

Dataset	<i>SSM-Algorithm</i>		<i>FP-Stream Alg</i>	
	Average CPU time per batch	Total CPU time per batch	Average CPU time per batch	Total CPU time
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

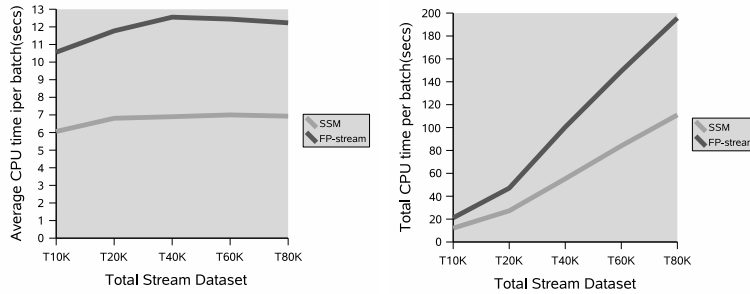


Figure 9. The Average CPU times for $s = .35\%$ and $e = .035\%$

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 data sets. It is not related to the size of the data sets. In this experiment a batch is holding approximately 5000 transactions. A number of experiments similar to the one in Table 2 at a minimum support of less than 1% were run on the data sets and the result of a second experiment on the same data sets but at a minimum support s of 0.0035 (0.35%) and error ϵ of 0.0003 (0.03%) is shown in Table 3 while the chart showing comparisons of average CPU times per batch and the total CPU times after batches used by the two algorithms for this data set is shown as Figure 9.

From the experimental results presented above, it can be seen that the av-

verage time for the SSM sequential algorithm to process stream batches is between 4.85 and 5.69 seconds for sequences with average lengths of 3 at minimum support of .45%, and it is between 6.06 and 7.0 for the same sequences with a lower support of .35%. This indicates that the lower the support, the higher the number of frequent patterns found and higher the execution times. Thus, comparing with the experimental results of the only other known sequential stream mining algorithm SPEED as reported in [21], with data set of average sequence length 3, but at a much higher minimum support threshold of 10%, it can be seen that the average execution time of the SPEED batch stream processing is steady at between 5 and 17 seconds for about the same number of batches in our experiment. This indicates that at this higher support, generally the SPEED algorithm execution times are higher than our SSM algorithm time at an even lower support.

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 data set 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.

4 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 ε serves to reduce on irrelevant use of memory for short-memored stream applications. Experiments show the SSM algorithm produces faster execution times than running the FP-Stream on similar data sets as well as shows the SSM produces comparable performance to the only other recently defined sequential stream mining algorithm SPEED based on the result reported in [21]. Discussion on how to extend the SSM algorithm to accommodate the sliding window, damped and any batch query range mining models have been provided in addition to the main landmark model mining presented in detail. Future work should consider the possibility of adding multiple dimensions (e.g. time dimension) or constraints along with frequency to discover interesting patterns in data streams. Methods to reduce or eliminate false positive results from mined results for applications needing highly precise results can be explored. It is still possible to increase the degree of interactivenss provided by the scheme to allow mining for various user-chosen parameters like minimum and error support thresholds. It is also possible to incrementally update the PLWAP tree during the mining of each batch rather than dropping and re-creating.

Acknowledgments

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., School of Computer Science, University of Windsor,

References

- [1] Cai, Y. Dora., Clutter, David., Pape, Greg., Han, Jiawei., Welge, Michael., Auvil, Loretta., 2007, *MAIDS: Mining Alarming Incidents from Data Streams*, Demonstration Proposal Paper, University of Illinois at Urbana-Champaign.
- [2] Chen, Y. and Dong, G. and Han, J. and Wah, W.B. and Wang, J., 2002, *Multidimensional regression analysis of time-series data streams*, pro-

ceedings of the 28th VLDB conference, pages 323-334, Hong Kong, China.

- [3] 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 & Chen, 2004a] Ezeife, C.I., Chen, Min. 2004. Mining Web Sequential Patterns Incrementally with Revised PLWAP Tree, proceedings of the fifth International Conference on Web-Age Information Management (WAIM 2004) Dalian, sponsored by National Natural Science Foundation of China, published in LNCS by Springer Verlag, pp. 539-548.
- [Ezeife & Chen, 2004b] Ezeife, C.I., Chen, Min. 2004. Incremental Mining of Web Sequential Patterns Using PLWAP Tree on Tolerance MinSupport, proceedings of the IEEE 8th International Database Engineering and Applications Symposium (IDEAS04), Coimbra, Portugal, July 7th to 9th, pp. 465-479.
- [4] 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
- [5] Ezeife, C.I. and Lu, Yi and Liu, Yi., 2005, *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.
- [6] Ezeife, C.I., Mostafa, Monwar., 2007, *SSM: A Frequent Sequential Data Stream Patterns Miner*, Proceedings of the IEEE Symposium on Computational Intelligence and Data Mining (CIDM 2007), Honolulu, Hawaii, USA, April 1-5, 2007, IEEE press.
- [7] Gaber, Mohamed M., Zaslavsky, Arkady., Krishnaswamy, Shonali., 2005, *Mining Data Streams: A Review*, ACM Sigmod Record, Vol. 34, Issue 2, June, pages 18-26.
- [8] 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.

- [9] Guna, S. and Mishra, N. and Motwani, R. and O'Callaghan, L., 2000, *Clustering data streams*, Proceedings of IEEE Symposium on Foundations of Computer Science (FOCS'00), pages 359-366.
- [10] Guna, S. and Meyerson, A., and Mishra, N. and Motwani, R., 2000, *Clustering data streams: Theory and Practice*, TKDE special issue on clustering, vol 15.
- [11] Gunduz, S. and Ozsu, M.T., 2003, *A web page prediction model based on click-stream tree representation of user behavior*, SIGKDD, Page 535-540.
- [12] 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.
- [13] Jiang, Nan., Gruewald, Le., 2006, *Research Issues in Data Stream Association Rule Mining*, ACM Sigmod Record, Vol. 35, No. 1, Mar., pages 14-19.
- [14] Last, M., 2002, *Online classification of nonstationary data streams*, Intelligent Data Analysis, Vol. 6, No. 2, Page 129-147.
- [15] Lin, J and Keogh, E. and Lonardi, S. and Chiu, B., 2003, *A symbolic representation of time series, with implication for streaming algorithms*, In Proc. of the 8th ACM SIGMOD workshop on research issues in data mining and knowledge discovery, San Diego, USA.
- [16] Lu, Yi., Ezeife, C.I., 2003, *Position Coded Pre-Order Linked WAP-Tree for Web Log Sequential Pattern Mining*, Proceedings of the 7th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD 2003), pages 337-349, Springer LNCS.
- [17] Manku, Gurmeet Singh. and Motwani, Rajeev., 2002, *Approximate frequency counts over data streams*, proceedings of the 28th VLDB conference, pages 346-357, Hong Kong, China.
- [18] Massegli, Florent., Teisseire, Maguelonne., Poncelet, Pascal., 2002, *Real Time Web usage Mining: a Heuristic based Distributed Miner*, proceedings of the second International conference on Web Information Systems Engineering, Volume: 1, Pages 288-297.

- [19] 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.
- [20] 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.
- [21] Raissi, Chedy., Poncelet, Pascal., Maguelonne, Teisseire., 2006, *SPEED: Mining Maximal Sequential Patterns Over Data Streams*, proceedings of the 3rd International Conference on Intelligent Systems, 2006.
- [22] 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.
- [23] 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.
- [24] Yang, Qiang., Haining, Henry., Li, Tianyi., 2001, *Mining web logs for prediction models in WWW caching and prefetching*, Proceedings of the ACM SIGKDD International conference, ACM Press.