Mining Web Sequential Patterns Incrementally with Revised PLWAP Tree

C.I. Ezeife*
School of Computer Science, University of Windsor,
Windsor, Ontario, Canada N9B 3P4
cezeife@uwindsor.ca
Min Chen
School of Computer Science, University of Windsor

No Institute Given

Abstract. Since point and click at web pages generate continuous data stream, which flow into web log data, old patterns may be stale and need to be updated. Algorithms for mining web sequential patterns from scratch include WAP, PLWAP and apriori-based GSP. An incremental technique for updating already mined patterns when database changes, which is based on an efficient sequential mining technique like the PLWAP is needed.
This paper proposes an algorithm, Re-PL4UP, which uses the PLWAP tree structure to incrementally update web sequential patterns. Re-PL4UP scans only the new changes to the database, revises the old PLWAP tree to accommodate previous small items that have become large and previous large items that have become small in the updated database without the need to scan the old database. The approach leads to improved performance.

Keywords: Incremental Mining, sequential mining, frequent patterns, PLWAP tree, Scalability

1 Introduction

When data are inserted or deleted from a database (like access patterns in web access log), previous patterns may no longer be interesting and new interesting rules could appear in the updated database. Incremental mining of sequential patterns is the process of generating new patterns in the updated database (old + new data) by using only the updated part (new data) and previously generated old patterns.

Sequential mining, unlike general association rule mining pays attention to the order the items occur as well [2]. For example, given a small web access log

* 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.

that recorded user accesses to 8 sites represented as sites {a, b, c, d, e, f, g, h} as shown in Table 1. Each transaction consists of items (or attributes, e.g., web

Table 1. The Example Database Transaction Table with Frequent Sequences

TID	Web access Seq.	Frequent subseq with s = 50%
100	abdac	abac
200	aebcace	abcac
300	baba	baba
400	afbacfc	abacc
500	abegfh	ab

sites visited), the association rule rule $a \rightarrow b$ asserts that if site a is visited, then, site b is visited. Each of these 8 sites represents an item in the database. An itemset X is called an i-itemset if it contains i items. The support of a rule is defined as the percentage of transactions (records) that contain the sets X and Y, while its confidence is the percentage of all database transactions containing "X", that also contain "Y". All items with support higher than a specified minimum support are called large or frequent itemsets. While minimum support is used to mine frequent patterns, only rules with confidence higher than the minimum confidence are retained. An example of a sequential pattern from the database of Table 1 is "if site a is accessed in a record, it is often followed by an access to site b". While the order is important in sequential pattern, items in a sequence do not necessarily need to be consecutive and an item can be repeated in one sequence. A sequence also has subsequences, e.g., abacc, afb are some subsequences of the sequence afbacfc. Frequent sequences are those with support equal or higher than minimum support (minsupport).

When data are inserted or deleted from a database, previous patterns may no longer be interesting and new interesting rules could appear in the updated database. Incremental mining of sequential patterns is the process of generating new patterns in the updated database (old + new data) using only the updated part (new data) and previously generated patterns.

1.1 Related Work

Few existing algorithms [1, 4, 6] for incremental sequential patterns mining are apriori-based rather than tree-based. They are composed of iteratively: (1) generating all large itemsets in the database and (2) generating sequential patterns in the database according to the large itemsets generated in the first step. ISE algorithm is proposed by [4] for incremental sequential patterns mining and scans the database several times. ISM [6] is also an apriori-like algorithm. It still needs to rescan the entire updated database many times if previous small items

become large after database update. When the 1-sequence becomes large, the updated data becomes explosive. WAP tree algorithm [5] scans the database only twice to build the WAP tree without generating candidate sets. It then, mines the WAP tree to extract sequential patterns. PLWAP algorithm uses a preorder linked version of WAP tree and an algorithm to eliminate the need to recursively re-construct intermediate WAP trees during mining as done by WAP tree technique. Neither WAP nor PLWAP algorithm is used for incremental mining.

1.2 Contributions

This paper proposes an algorithm named Re-PL4UP (Revised PLWAP FOR UPdated sequential mining). This algorithm applies the PLWAP-tree [3] to the incremental sequential mining problem. The Re-PL4UP algorithm eliminates the need to re-scan the old database when new changes arrive, in order to update old patterns. The Re-PL4UP algorithm uses the information from original PLWAP tree, without the need to re-scan the entire database (old + new data) in order to update old patterns. The approach is to initially build a PLWAP tree that is based on minimum support, s and which remembers the position codes of small items. Revising the PLWAP tree will entail traversing the tree to insert new nodes, delete nodes that are no longer frequent, changing links and re-building the prefix linkages used during mining. A mirror copy of all modified and newly inserted branches of the tree is mined and the patterns from this mirror copy are used to update the previously mined patterns. This yields better performance and allows for application scalability.

Section 2 first presents the proposed incremental sequential mining algorithm, Re-PL4UP. Section 3 discusses an example mining of a database and its update using the proposed Re-PL4UP algorithm, section 4 presents performance analysis of the algorithm, and finally, section 5 presents conclusions and future work.

2 The Proposed Incremental Re-PL4UP Algorithm

The algorithm, Re-PL4UP being proposed for mining frequent sequential patterns incrementally uses PLWAP tree structure. Section 2.1 discusses the algorithm Re-PL4UP. Let F and S represent previous large (frequent) items and previous small items in original database respectively; F' and S' represent updated large (frequent) items and updated small items in the updated database U, each event (item) in updated database U, belongs to one of these six categories of items: (1). Frequent in old database, DB and are still frequent in updated database (old + new data), U, ($F \rightarrow F'$); (2). Frequent in old DB but small in updated database, ($F \rightarrow S'$); (3). Small in old DB but frequent in updated database, ($S \rightarrow F'$); (4). Small in old DB and still small in updated database, ($S \rightarrow S'$); (5). New and frequent in updated database ($\emptyset \rightarrow F'$); (6). New and small in updated database ($\emptyset \rightarrow S'$).

2.1 Mining Incremental Patterns with Re-PL4UP

This algorithm scans only the changes to the database (db). Next, it uses frequent items in the changes to the database to update the old PLWAP tree of the original database before mining. The most important update made to the old PLWAP tree are for two classes of items namely: items in category 2, which are, $F \rightarrow S'$, that now need to be deleted from the old tree; and the items in category 3, which are $S \rightarrow F'$ that need to be inserted into the tree. The Re-PL4UP approach is to take advantage of the position codes property of the PLWAP tree and during initial construction of the PLWAP tree, store the list of position codes of all small items. During update, these unique position codes are used to re-insert the previous small items in proper positions in the tree without re-scanning the old database. The updated patterns are now the union of the old patterns, the patterns from modified branches of old Re-PL4UP tree and the patterns from the new incremental database tree.

2.2 Mining Initial Frequent Patterns with PL4UP

This section presents the sequence of steps for mining initial frequent patterns using the proposed Re-PL4UP algorithm, as well as how to mine it incrementally, while an example application of the algorithm is given in section 3. Given a web access sequence database (WASD) or its equivalent, a minimum support, s, the proposed Re-PL4UP algorithm will take the following steps in mining initial frequent patterns.
1. Construct initial PLWAP tree using minimum support, s obtaining frequent 1-items (with support *geq* s), F_1, the frequent 1-items, the Small 1-items (with support $<$ s), S_1. Then, during construction of the PLWAP tree, we shall define a small item code profile for every small 1-item in the S_1 list by including a list of position codes of the PLWAP tree indicating the position that this small item would have been on, in the tree if it were frequent.
2. We construct a PLWAP tree [3] using the seq_s from first step above and the frequent F_1 items for header link connections and define the small item code profile for small items $S - code^{DB}$. This is called $Re - PL4UP^{DB}$ tree.
3. We now mine the $Re - PL4UP^{DB}$ tree for frequent patterns (called FP^{DB}), by extracting all patterns with support greater than or equal to s.

2.3 Steps in Incremental Mining of Frequent Patterns with Re-PL4UP

When new transactions are inserted into or deleted from the database, the steps for updating old patterns are given below. The input data for this process are: (1) the changed database, db, (2) the old database DB tree, $Re-PL4UP^{DB}$, (3) the candidate 1-items, C_1^{DB}, (4) minimum supports, s, (5) the frequent 1-items, F_1, (6) the small 1-items, S_1, (7) the frequent patterns, FP^{DB}. The steps in incrementally mining the database using mostly changed db and old available patterns are presented in Figure 1 while and example mining with this algorithm is presented in section 3.

Algorithm 21 *(Re-PL4UP-Mines Web Log Sequences Incrementally)*

Algorithm Re-PL4UP()
Input: original database, DB, Incremental database, db, minimum support λ ($0 < \lambda \leq 1$) original DB tree $Re-PL4UP^{DB}$, old frequent pattern, FP, old candidate lists (C_1, F_1, S_1), small code profile $S-code^{DB}$.
Output: updated frequent patterns for database, U (FP'), $Re-PL4UP^{DB}$ tree. updated candidate lists (C_1', F_1', S_1'). updated small code profile $S-code^{U}$.
Intermediate data:incremental db candidate lists (C_1^{db}, F_1^{db}, S_1^{db}))
begin
(1)Update all candidate lists as follows:
$C_1' = C_1 \cup C_1^{db}$; s' = λ of $|DB| + |db|$.
F_1' = elements in C_1' with support geq s'
S_1' = elements in C_1' with support $<$ s'.
$F_1^{db} = C_1^{db} \cap F_1'$; $S_1^{db} = C_1^{db} \cap S_1'$.
(2)Classify items in the updated data, U into one of 6 classes as:
$F_1 \cap F_1'$ are in class $F \rightarrow F'$; $F_1 \cap S_1'$ are in class $F \rightarrow S'$.
$S_1 \cap F_1'$ are in class $S \rightarrow F'$; $S_1 \cap S_1'$ are in class $S \rightarrow S'$.
$F_1' - F_1$ are in class $\emptyset \rightarrow F'$; $S_1' - S_1$ are in class $\emptyset \rightarrow S'$.
(3) Modify the old $Re-Pl4UP^{DB}$ tree such that all $F \rightarrow S'$ items are deleted from the tree, and all $S \rightarrow F'$ are inserted into tree using the $S-code^{DB}$.
(4) Mine modified branches of $Re-PL4UP^{DB}$ to get frequent patterns $Re-FP^{DB}$.
(5) Construct and mine small $Re-PL4UP^{db}$ to get frequent patterns $Re-PL4UP^{db}$
(6) Combine the three frequent patterns to obtain FP' as:
$FP' = FP^{DB} \cup Re-FP^{DB} \cup FP^{db}$
(7) Insert the frequent sequence transactions from the incremental database into the original $Re-PL4UP^{DB}$ tree to update it; update the links and small code profiles.
end // of Re-PL4UP //

Fig. 1. The Re-PL4UP Algorithm

3 Mining an Example Database with Re-PL4UP Algorithm

Suppose we have a database DB with set of items, I= {a, b, c, d, e, f, g, h} and minimum support = 50% of DB transactions. A simple database transaction table for illustrating the idea is given in the first two columns of Table 1. The first process is to build the initial $Re-PL4UP^{DB}$ tree [3] using sequences in Table 1 with support greater than or equal to the tolerance support s of 50% (equivalent to 3 transactions in Table 1). The $Re-Pl4UP^{DB}$ tree is built the same way the PLWAP tree is built. To build, first scan the database sequence (column 2 of the Table) once to obtain the candidate 1-items, 1-frequent and 1-small lists with their supports as: C_1 = {a:5, b:5, c:3, d:1, e:2, f:2:, g:1, h:1}. $F_1 = \{a, b, c\}$. $S_1 = \{e, f, d, g, h\}$. Then, scan the web access database (column 2 of Table), a second time to create a frequent sequence from each transaction, the frequent sequence (seq_s) that includes all items with support greater or equal

to support, s (seq_s is shown on column 3 of Table 1). insert each of the seq_s transactions in the tree with their count and position code to obtain the PLWAP tree in (Figure 2). Each node is described as (a:1:1) standing for (label of the node:count of the node:position code of the node). In defining the position code of a node, the PLWAP algorithm applies the rule that the root of the tree has a null position code, but every other node has a position code that is equivalent to appending '1' to the code for this node's parent if this node is the leftmost child of its parent, otherwise, its position code is obtained by appending '0' to the position code of its nearest left sibling. While inserting the frequent items in the sequence, the algorithm checks the original transaction to mark location of small items in the transaction. For example, small item d in the original first transaction *abdac* would have had the position code (d:1:111) in the created branch if this d were frequent. It will write this position code in the small item code profile for item d as $S - code^d = \{111\}$. The position code of a node does not change unless the node moves. The complete small item code profiles for all

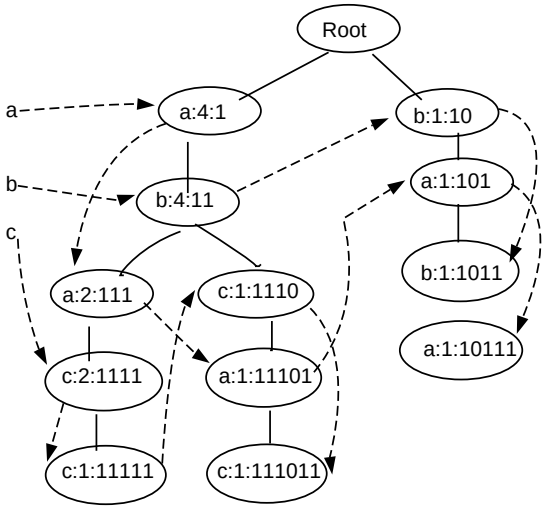

Fig. 2. The Re-PL4UP tree with Support s for the Example Database

small items after inserting all frequent sequences in the tree are:
$S - code^d = \{11\}$ or $\{3\}$, $S - code^e = \{110, 11011111, 1110\}$ or $\{6, 223, 14\}$, $S - code^f = \{1100, 11001111, 111011\}$ or $\{12, 207, 59\}$, $S - code^g = \{11101\}$ or $\{29\}$, $S - code^h = \{1110111\}$ or $\{119\}$. Note that these position codes can also be recorded in their decimal number equivalents as shown above. After building the tree, a pre-order traversal mechanism (visit root, visit left subtree, visit right subtree) is used to add a pre-order linkage on the tree for all frequent 1-items, $F_1 = \{a, b, c\}$. The broken lines on Figure 2 starting with each frequent F_1 item is used to show the pre-order linkage between nodes of this type. In step 3, the

$Re-PL4UP^{DB}$ begins the mining process. It starts following the header linkage of the first frequent item, "a", and obtains the support of this prefix subsequence "a" as the sum of the counts of all first "a" nodes in the current a:suffix root set, which are the tree branches rooted at (a:4:1 and a:1:101). If the sum of the counts of these first a nodes on different branches of the tree at the level of the tree under consideration is greater or equal to support of 3, we include this sequence in the FP^{DB} frequent pattern list. The "a" is frequent because this suffix root set has a total count of 5. Next, we shall continue to check down the suffix trees under use to see if the sequences (aa, ab, ac) are also frequent. The a-header link serves the purpose of tracking and constructing these root sets during mining. Process continues in a similar fashion recursively checking for all patterns beginning with "a", "b" and "c". After checking all patterns, the list of FP^{DB} mined based on the support of 3 is: FP^{DB} = {a:5, aa:4, aac: 3, ab:4, ac:3, aba:4, abac:3, abc:3, b:5. ba:4, bac:3, bc:3, c:3}.

3.1 Mining Updated Database with Re-PL4UP Tree Algorithm

Assume that the original database, DB of Table 1 is updated with the records in the changes to database table shown as Table 2, the objective of the $Re-PL4UP^{DB}$ algorithm is to use old rules, FP^{DB} from the previous section, with the new changes to database and other intermediate information from the previous section like candidate 1-items, frequent 1-items, and small 1-items, small item code profiles $S-code^{DB}$, to compute the new frequent patterns in the entire updated database, using the same support, s, of 50%. Thus, the $Re-PL4UP^{DB}$

Table 2. The Changes to Database Transaction Table

TID	Web access Seq.	Frequent subseq with s = 50%	Frequent subseq with t = 0.6s
700	bahefg	bahefg	bahefg
800	aegfh	aegfh	aegfh

algorithm mines the updated database incrementally as:
1. Update all intermediate candidate lists and patterns as follows: $C_1' = C_1 \cup C_1^{db}$. C_1 = {a:5, b:5, c:3, d:1, e:2, f:2:, g:1, h:1}. C_1^{db} = {a:2, b:1, e:2, f:2, g:2, h:2}, thus, C_1' = $\{a:7, b:6, c:3, d:1, e:4, f:4, g:3, h:3\}$. F_1' = $\{a:7, b:6, e:4, f:4\}$ and $F_1^{db} = C_1^{db} \cap F_1'$, = {a:2, b:1, e:2, f:2}. S_1' = $\{c:3, d:1, g:3, h:3\}$. $S_1^{db} = C_1^{db} \cap S_1'$ = {g:2}.
2. Classify items in the updated database U, into one of the defined six categories as:(i) $F_1 \cap F_1'$ = {a, b}, (ii) $F_1 \cap S_1'$ = {c} (iii) $S_1 \cap F_1'$ = {e, f}, (iv) $S_1 \cap S_1'$ = {d, g, h}, (v) $F_1' - F_1 = \emptyset$, and (vi) $S_1' - S_1 = \emptyset$.
3. Modify the old $Re-PL4UP^{DB}$ tree to delete from items in category $F \rightarrow S'$, constituting {c}, to insert all items in the category $S \rightarrow F'$ = {e, f} using their

small code profile. Every small item code profile has a unique position in the current tree, determined by any matching prefix of its binary position code. Thus, if the position profile code is 111011 and we can find a node position in the current tree for the prefix 1110, then, we insert the small-to-large item there. The new code profile of the item is the physical code in the tree. The small code profile list is updated such that the updated codes for the small-to-large items {e, f} after tree modification are: $S - code^{e}$'= {110, 1101, 1110} and $S - code^{f}$' = {1110, 11001, 11101}. Finally, the frequent header linkages of the frequent items in the modified tree are re-constructed. The revised tree is shown as Figure 3.

4. Mine only the modified branches of the revised tree, $Re - PL4UP^{DB}$ (which

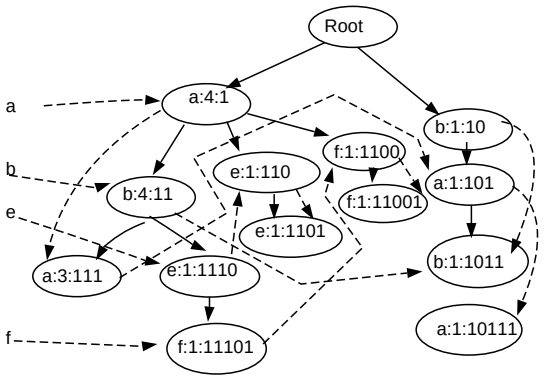

Fig. 3. The Revised PL4UP tree After Deletion and Insertion of Items with Changed Status

is the left branch of Root in Figure 3) to obtain the frequent sequences called $Re - FP^{DB}$ as {aee:1, abef:1, aff:1, ae:2, af:2, be:1, bf:1, bef:1, ef:1, ee:1, ff:1, e:2, f:2}.

5. Construct the small $Re - PL4UP^{db}$ tree using only the changes to the database, db. Mine this $Re - PL4UP^{db}$ to obtain the FP^{db} patterns. The tree for small db is based on $F_1^{db} = C_1^{db} \cap F_1'$={a, b, e, f}. Following this process, the useful mined FP^{db} = {b:1, a:2, h:2, e:2, f:2, g:2, ba:1, be:1, bf:1,aef:2, bef:1, ae:2, af:2, ef:2, bf:1}.

6. Combine the frequent patterns from (1) old database, DB, and (2) modified branches of old DB and (3) changes to the database, db, keeping only those patterns that have support $\geq s'$ (which is 4) in updated database. FP' are patterns in $FP^{DB} \cup Re - FB^{DB} \cup FP^{db}$ with support greater or equal to s' (of 4 transactions). Thus, FP' = {a:7, aa:4, ab:5, aba:4, b:6, ba:5, ae:4, af:4, e:4, f:4}.

7. Finally, the frequent sequences in the incremental database, db, which are (baef, aef) are inserted into the main revised $Re - PL4UP^{DB}$ to keep it up-to-date, while the small code profile of the new small items in the new changes as

well as the small code profiles of all deleted nodes from the tree are recorded in the small code profiles.

4 Experimental and Performance Analysis

A performance comparison of Re-PL4UP, PLWAP and ISE algorithms was conducted. All these three algorithms were implemented and run on the same datasets generated using the resource code for generating synthetic datasets downloaded from http://www.almaden.ibm.com/cs/quest/syndata.html. The experiments were conducted on a Pentium 4 PC machine with 256 megabytes of main memory running Windows operating system. The programs were written in C++ under visual C++ environment. The number of transactions (D) in this dataset is sixty thousand records, that is $|D| = 60{,}000$ records, the average size of transactions (length of sequences) (T) is 10, $|T| = 10$, average length of maximal pattern (that is, average number of items in the longest frequent sequence) (S) is 6, or $|S| = 6$, number of items or events (N) (the total number of attributes) is one thousand, N=2000. Assume the size of updated (inserted) dataset is 20,000 records, and the support thresholds are varied between 1% and 20%. An experimental result is shown in Table 3. **Experiment 2: Execution**

Table 3. Execution Times for Dataset at Different Supports

Algorithms	CPU Time (in secs) at Supports of				
	1	2	3	4	5
ISE	4000	1405	515	188	101
PLWAP	212	112	80	66	60
Re-PL4UP	164	69	40	27	14

Time for Databases with Different Sizes
We use different database sizes that vary from 20k to 100k to compare the three algorithms. The minimum support 20% is used for Re-PL4UP, ISE and PLWAP and the result of the experiment is shown in Table 4. The five datasets used for the experiment are T10.S5.N2000.D20K, T10.S5.N2000.D40K, T10.S5.N2000.D60K, T10.S5.N2000.D80K, and T10.S5.N2000.D100K.

5 Conclusions and Future Work

Re-PL4UP algorithm proposed in this paper, modifies an existing PLWAP tree by utilizing the metadata of old database transactions as well as old mined frequent patterns in order to incrementally update web log sequential patterns. One major contribution of work is the technique for efficiently using position

Table 4. Execution Times at Different Transaction Sizes on Support 16%

Algorithms (times in secs)	Different Changed Transaction Size				
	20K	40K	60K	80K	100K
ISE	100	189	285	378	470
PLWAP	29	51	81	98	128
Re-PL4UP	19	33	54	83	105

codes of small items in database sequences to restore information about previous small items that were not stored in the tree, when the database is updated and these items become frequent, without re-scanning old database. Experiments show that Re-PL4UP performs better than existing incremental sequential mining algorithms and no huge candidate itemsets need to be generated. like the old patterns and tree for mining the updated database. Future work should investigate applying technique to distributed and parallel mining that may involve continuous time series data, and to web content and text mining.

References

1. Agrawal, R. and Srikant, R.: Mining Sequential Patterns, Proceedings of the 11th Int'l Conference on Data Engineering, Taipei, Taiwan, March 1995, pp. 3-14.
2. Han, J., Kamber, M.: Data Mining: Concepts and Techniques Morgan Kaufmann, 2001.
3. Lu, Yi., Ezeife, C.I.: 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), Seoul, Korea, Apr. 30-May 2 2003, , pp. 337-349.
4. Masseglia, F., Poncelet, P., Teisseire, M.: Web Usage Mining: How to Efficiently Manage New transactions and New Customers. Rapport de Recherche LIRMM, 18 pages, Fevrier 2000. Version courte dans Proceedings of the 4th European Conference on Principles of Data Mining and Knowledge Discovery (PKDD'00), Lyon, France, September 2000, pp.530-535.
5. Pei, Jian., Han,Jiawei., Mortazavi-asl, Behzad., Zhu, Hua.: Mining Access Patterns Efficiently from Web Logs. Proceedings 2000 Pacific-Asia Conf. On Knowledge Discovery and Data Mining (PAKDD'00), Kyoto, Japan, April 2000.
6. Parthasarathy, S., Zaki, M.J., Ogihara, M., Dwarkadas, S. Incremental and Interactive Sequence Mining, In Proc.(1999) of the 8th International Conference on Information and Knowledge Management (CIKM99), 251- 258, Kansas City, MO, November 1999, pp.530-535.