Incremental Mining of Web Sequential Patterns Using PLWAP Tree on Tolerance MinSupport

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

Abstract

This paper proposes an algorithm, PL4UP, which uses the PLWAP tree structure to incrementally update web sequential patterns. PL4UP initially builds a bigger PLWAP tree that includes all sequences in the database with a tolerance support, t, that is a fraction of the database minimum support, s. The position code features of the PLWAP tree are used to efficiently mine this tree to extract both current frequent and non-frequent sequences, which are likely to become frequent when the database is updated. This approach more quickly updates old frequent patterns without the need to re-scan the entire updated database.

Keywords: Incremental Mining, sequential mining, frequent patterns, Apriori-like algorithms, PLWAP tree, Scalability

1 Introduction

Sequential pattern mining is based on association rule mining concepts of support, confidence, minimum support, minimum confidence and frequent sequence (pattern). When data are inserted or deleted from a database, previous patterns may no longer be interesting and new interesting patterns could appear in the updated database. The process of generating new patterns in the updated database (old + new data) using only the updated part (new data) and previously generated patterns is called incremental mining of sequential patterns. Although sequential mining is an important data mining task, it has received relatively little attention compared with association rules mining, especially in the area of incremental mining of sequential patterns [4, 6]. Existing incremental sequential mining algorithms that are apriori-based include [1, 4, 6]. Tree-based mining algorithms are [3, 5] and these are much faster than the apriori-based algorithms and incremental sequential techniques may benefit from a tree-based mining approach. This paper proposes a method for incremental sequential data mining based on the PLWAP-tree that can achieve better performance than existing algorithms.

*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.

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], to address the maintenance problem for sequential patterns mining. ISM [6] is also an apriori-like algorithm. 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

In this paper, an algorithm named PL4UP (PLWAP FOR UPdated sequential mining) is proposed. This algorithm applies the PLWAP-tree [3] to the incremental sequential mining problem. The PL4UP algorithm eliminates the need to re-scan the old database when new changes arrive, in order to update old patterns. The approach is to initially build a PLWAP

tree that is based on a lower threshold minimum support, t, which is a fraction of the application's regular minimum support, s. The tolerance support, t, is based on the average computed changing rate of data in the database.

2 The Proposed Incremental PL4UP Algorithm

With F and S representing frequent and small items in original database respectively, while F' and S' represent updated large (frequent) items and updated small items in the updated database U (old + new data), all events (items) in updated database U will fall into six categories of items as: (1) large in old database, DB and still large in updated database U ($F \rightarrow F'$); (2) large in old DB but small in U ($F \rightarrow S'$); (3) small in old DB but large in U ($S \rightarrow F'$); (4) small in old DB and still small in U ($S \rightarrow S'$); (5) new items that are large in U ($\emptyset \rightarrow F'$); and (6) new items that are small in U ($\emptyset \rightarrow S'$).

The main idea of PL4UP is to avoid scanning the whole updated database, U (DB+db) but to scan only the changes to the database (db). Then, to perform incremental mining by building a small $PLWAP^{db}$ tree for only these new changes to the database (db). Once this small $PLWAP^{db}$ is built, it can be used to update the existing old $PLWAP^{DB}$. With this approach, old patterns can quickly be updated without the need to scan the old DB. However, this will work for five of the six cases identified above, but not for the third case, when $S \rightarrow S'$. If there are any items in this category, the existing $PLWAP^{DB}$ will not contain adequate information regarding all sequences which had these previous small items (not shown in the $PLWAP^{DB}$) so that they could be updated with the new data. The PL4UP solution is to use a lower tolerance support in building the initial $PLWAP^{DB}$ tree to accommodate most potentially frequent items.

Thus, from the original $PLWAP^{DB}$ tree, the small items S are broken into two groups to get those that are highly likely to become frequent during next database update (potentially frequent PF), and those that are highly likely to still remain small (SS) using the tolerance support t, which is lower than the regular minimum support such that t = factor * s for $0 \leq factor \leq 1$. The limit on the t value suitable for good performance is determined using the average rate of changes in the database collected over a period of time.

2.1 Steps in Mining Frequent Patterns Incrementally with PL4UP

1. Construct initial PLWAP tree using tolerance minimum support, t (instead of minimum support s). We obtain the frequent 1-items (meeting support s), F_1, the frequent 1-items (meeting support t), F_t, the Small 1-items (not meeting support s), S_1. From the S_1 list, we obtain the potentially frequent 1-items (in S_1 list but meeting support t), PF_1. We also identify the potentially still small 1-items, SS_1 (in S_1 list but not meeting the support t). The candidate 1-items C_1^{DB}, recording the support of all items in the database are also obtained from the database. From each transaction in the database, we obtain two subsequences, the frequent subsequence meeting s requirements (seq_s) and the frequent subsequence meeting the t requirements (seq_t).

2. Construct a PLWAP tree using the seq_t from first step above and the frequent F_t items for header link connections. This is called $PL4UP_{DB}$ tree.

3. Now mine the $PL4UP_{DB}$ tree for tolerance frequent patterns (called TFP), by extracting all patterns with support greater than or equal to t. Since TFP includes all SFP (for support s requirements) patterns (that is, $TFP \supseteq SFP$), we obtain the needed SFP from the TFP as those patterns with support greater than or equal to minimum support, s.

4. Now, when database changes and we want to update the patterns incrementally, Update all intermediate candidate lists and patterns as follows: Obtain the updated candidate 1-items, C_1' $C_1' = C_1 \cup C_1^{db}$. F_1' and F_t' are obtained as items in C_1' with supports greater or equal to regular support s, and tolerance support, t respectively. $F_1^{db} = C_1^{db} \cap F_1'$, and $F_t^{db} = C_1^{db} \cap F_t'$, Next, the updated small, potentially frequent and still small items in the changes to database (db) are obtained as follows. $S_1^{db} = C_1^{db} \cap S_1'$, and $PF^{db} = C_1^{db} \cap PF'$,

5. Construct the small PL4UP tree using only the changes to the database, db and with frequent items based on tolerance support, F_t^{db}. A significant difference in this step is that the frequent 1-items used to construct the small db tree is not that directly computed from tolerance support of db, but computed from intersecting ($C_1^{db} \cap F_t'$).

6. Combine the frequent patterns from the changes to database, db and the old database, DB, keep-

Table 1: The Example Database Transaction Table with Frequent Sequences

TID	Web access Seq. Seq.	Frequent subseq with s = 50%	Frequent subseq with t = 0.6s
100	abdac	abac	abac
200	aebcace	abcac	aebcace
300	baba	baba	baba
400	afbacfc	abacc	afbacfc
500	abegfh	ab	abef

ing only those patterns that meet the regular support s' in updated SFP' and those that meet tolerance support, t' in updated TFP'. Thus, update SFP' and TFP' as follows: SFP' are patterns in $TFP \cup SFP^{db}$ with support greater or equal to s'. Similarly, TFP' are patterns in $TFP \cup TFP^{db}$ with support greater or equal to t.

3 Mining an Example Database with PL4UP Algorithm

PL4UP algorithm being proposed in this paper is a more generalized form of the PLWAP tree algorithm [3], which uses a lower tolerance minimum support to accommodate future incremental mining when updates occur. This section shows an example mining with PL4UP algorithm presented in Section 2.

3.1 PL4UP tree Algorithm on a Sample Database

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, t is given as 0.6s%. 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 $PL4UP_{DB}$ tree using sequences in Table 1 with support greater than or equal to the tolerance support t of 0.6s or 30% (equivalent to 2 transactions in Table 1). The PL4UP tree is built the same way the PLWAP tree is built but with lower tolerance support, t. To, build, first scan the database sequence (column 2 of the Table) once to obtain the candidate 1-items with their supports as: C_1 = {a:5, b:5, c:3, d:1, e:2, f:2:, g:1, h:1}.
Next, with regular support of 50% or 3 transactions, define the regular and tolerance frequent 1-items as:

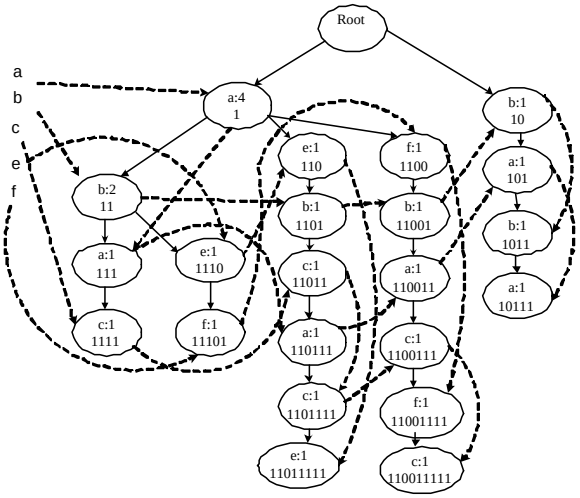

Figure 1: The PL4UP tree with tolerance Support t for the Example Database

$F_1 = \{a, b, c\}$ and $F_t = \{a, b, c, e, f\}$. Also, obtain the regular small 1-items, the potentially frequent and potentially still small items as: $S_1 = \{e, f, d, g, h\}$, $PF = \{e, f\}$ and $SS = \{d, g, h\}$ respectively. Then, scan the web access database (column 2 of Table), a second time to create two frequent sequences from each transaction, the regular frequent sequence (seq_s) that includes all items with support greater or equal to regular support, s (seq_s is shown on column 3 of Table 1), and the tolerance frequent sequence (seq_t) which includes all items with support greater or equal to tolerance support, t (seq_t is shown on column 4 of Table 1). Figure 1 shows the $PL4UP^{DB}$ tree for the example database. Each seq_t sequence is inserted into the tree and each node is labeled as (node name: node count: node position code). Position code of a node is '1' appended to the position code of its parent if the node is the leftmost child, otherwise it is '0' appended to the position code of its nearest left sibling. 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 tolerance frequent 1-items, $F_t = \{a, b, c, e, f\}$. The broken lines on Figure 1 starting with each frequent F_t item is used to show the pre-order linkage between nodes of this type. Next, this tree is mined for frequent patterns the PLWAP approach. To mine a PLWAP tree for frequent patterns, we easily identify the suffix root set of the node (e.g, 'a') under consideration by getting all such nodes on different branches of the tree at the tree level under consideration. This node label is considered frequent if the sum of the counts of these labels on different branches is greater than the tolerance support t; and if found

Table 2: The Changes to Database Transaction Table

TID	Web access Seq.	Frequent subseq with s using% $C_1^{db} \cap F_1'$	Frequent subseq with t using $C_1^{db} \cap F_t'$
700	bahefg	baef	bahefg
800	aegfh	aef	aegfh

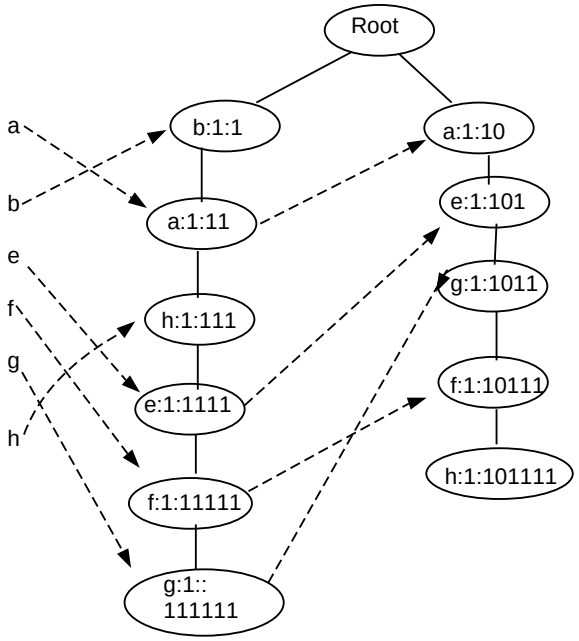

Figure 2: The PL4UP tree with tolerance Support t for Changes to Database

frequent, it is appended to the previously discovered prefix sequence and the process continues until there is no more frequent suffix to append. Progressively, sequences "a", "aa", "aaa", "aac", etc. are discovered frequent. After checking all patterns, the list of TFP mined is: {a:5, aa:4, aac: 3, ab:5, aba:4, abac:3, abc:3, abcc:2, abe:2, abf:2, ac:3, acc:2, ae:2, af:2, b:5. ba:4, bac:3, bab:1, bc:3, bcc:2, be:2, bf:2, c:3, cc:2, e:2, f:2}. The actual needed frequent pattern, SFP, now are the ones based on the regular support of 3. These are all patterns in TFP with support 3 or more. The SFP = {a:5, aa:4, aac: 3, ab:5, ac:3, aba:4, abac:3, abc:3, b:5. ba:4, bac:3, bc:3, c:3}.

Now, 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 PL4UP algorithm is to use old rules, SFP and TFP 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, to compute the new frequent patterns in the entire updated database, using the same regular support, s, of 50% and tolerance support, t, of 30%. Thus, the PL4UP algorithm mines the updated database incrementally 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' = $\{a : 7, b : 6, c : 3, d : 1, e : 4, f : 4, g : 3, h : 3\}$. Updated regular support, s', of updated database, U is 50% of 7 or 4 transactions, while the tolerance support, t', of U is 30% of 7 or 3 transactions(ceiling(2.1)). $F_1' = \{a : 7, b : 6, e : 4, f : 4\}$ and $F_t' = \{a : 7, b : 6, c : 3, e : 4, f : 4, g : 3, h : 3\}$. $F_1^{db} = C_1^{db} \cap F_1'$, $= \{a : 2, b : 1, e : 2, f : 2, g : 2, h : 2\} \cap \{a : 7, b : 6, e : 4, f : 4\}$ = {a:2, b:1, e:2, f:2}. $F_t^{db} = C_1^{db} \cap F_t' = \{a : 2, b : 1, e : 2, f : 2, g : 2, h : 2\} \cap \{a : 7, b : 6, c : 3, e : 4, f : 4, g : 3, h : 3\}$. $= \{a : 2, b : 1, e : 2, f : 2, g : 2, h : 2\}$. $S_1^{db} = C_1^{db} \cap S_1'$ = {g:2}. PF' = {c:3, g:3, h:3 }. $PF^{db} = C_1^{db} \cap PF'$ = {g:2, h:2}. SS' = {d:1}. $SS^{db} = C_1^{db} \cap SS' = \emptyset$ Next, we construct the small $PL4UP^{db}$ tree using only the changes to the database, db and mine this $PL4UP^{db}$ to obtain the regular, SFP^{db}, and the tolerance frequent patterns, TFP^{db}. Figure 2 shows the $PL4UP^{db}$ tree for the small db based on the tolerance support t of 30% or 1 transaction. The mined TFP^{db} = {a:2, ae:2, aef:2, af:2, b:1, ba:1, bae:1, baef:1, be:1, bf:1, ..., e:2, ef:2, f:2} and the SFP^{db} as usual is obtained from its TFP^{db} counterpart as sequences that have support greater or equal to the regular support of changed db, which is also 1 transaction in this case. Thus, $SFP^{db} = TFP^{db}$. Finally, we combine frequent patterns from changes to database, db and the old database, DB, keeping only those patterns that meet the regular support s' (which is 4) in updated database, SFP' and those that meet tolerance support, t' (which is 3) in updated TFP. SFP' are patterns in $TFP \cup SFP^{db}$ = {a:7, aa:4, aac:3, ab:5, aba:4, abac:3, abc:3, ac:3, ae:4, af:4, b:6, ba:5, bac:3, bc:3, be:3, bf:3, c:3, e:4, f:4} with support greater or equal to s' (of 4 transactions). Thus, SFP' = {a:7, aa:4, ab:5, aba:4, ae:4, af:4, b:6, ba:5, e:4, f:4} Similarly, TFP' are patterns in $TFP \cup TFP^{db}$ with support greater or equal to t (or 3 transactions). Thus, TFP' = {a:7, aa:4, aac:3, ab:4, aba:4, abac:3, abc:3, ac:3, ae:4, af:4, b:6, ba:5, bac:3, bc:3, be:3, bf:3, c:3, e:4, f:4}.

4 Experimental and Performance Analysis

A performance comparison of PL4UP, PLWAP and ISE algorithms was conducted and the results of the experiments are presented in this section. 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 correctness of the implementations were confirmed by checking that the frequent sequences generated for the same dataset by the three algorithms are the same. 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. From observation of experimental result, we can see that (i) as the size of the support increases, the execution times of all the algorithms decrease. (ii) for the same support, the execution time of PL4UP algorithm is less than that of PLWAP and ISE algorithms.

Experiment 2: Execution 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 5% is used for PL4UP, ISE and PLWAP and the result of the experiment show that when the inserted transactions become large, all the execution times become larger. Also, the larger size or updates to database affect the execution time of ISE and PLWAP with lower minimum support much more than with higher minimum support. The PL4UP always achieves better performance than PLWAP and ISE.

5 Conclusions and Future Work

ISE [4] or similar apriori-like algorithms generate numerous candidate itemsets that need to be computed at each level and scans the updated database U many times. For all types of database updates, ISE updates the old generated frequent patterns by mining data level-wise the same way. If the PLWAP algorithm is directly used to mine the updated database, it has to scan the updated database U two times to construct PLWAP tree, and mine the tree step by step without using existing patterns and tree of the original database DB, which wastes a lot of reusable resources. The proposed PL4UP inherits the advantages of PLWAP, and no huge candidate itemsets need to be generated. They also fully utilize old existing information like the old patterns and tree for mining the updated database U. 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.

[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 2003.

[4] Masseglia, F., Poncelet, P., Teisseire, M. Incremental Mining of Sequential Patterns in Large Databases, Actes des 16imes Journes Bases de Donnes Avances (BDA'00), Blois, France, October 2000.

[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

[7] Srikant, Ramakrishnan., Agrawal, Rakesh.: Mining generalized association rules. In proceedings of the 21st Int'l Conf. on Very Large Databases (VLDB), Zurich, Switzerland, Sept. 1995.