# An Incremental Algorithm for Maintaining the Built FUSP Trees Based on the Pre-large Concepts

Chun-Wei Lin[1,2], Wensheng Gan[1], Tzung-Pei Hong[3,4], and Raylin Tso[5]

[1] Innovative Information Industry Research Center (IIIRC),
[2] Shenzhen Key Laboratory of Internet Information Collaboration
School of Computer Science and Technology
Harbin Institute of Technology Shenzhen Graduate School
HIT Campus Shenzhen University Town, Xili, Shenzhen 518055 P.R. China
[3] Department of Computer Science and Information Engineering
National University of Kaohsiung, Kaohsiung, Taiwan, R.O.C.
[4] Department of Computer Science and Engineering
National Sun Yat-sen University, Kaohsiung, Taiwan, R.O.C.
[5] Department of Computer Science
National Chengchi University, Taipei, 11605, Taiwan, R.O.C.
jerrylin@ieee.org, wsgan001@gmail.com, tphong@nuk.edu.tw,
raylin@cs.nccu.edu.tw

**Abstract.** Mining useful information or knowledge from a very large database to aid managers or decision makers to make appropriate decisions is a critical issue in recent years. In this paper, we adopted the pre-large concepts to the FUSP-tree structure for sequence insertion. A FUSP tree is built in advance to keep the large 1-sequences for later maintenance. The pre-large sequences are also kept to reduce the movement from large to small and vice versa. When the number of inserted sequences is smaller than the safety bound of the pre-large concepts, better results can be obtained by the proposed incremental algorithm for sequence insertion in dynamic databases.

**Keywords:** Pre-large concept, dynamic databases, sequential pattern mining, sequence insertion, FUSP-tree structure.

## 1 Introduction

Mining desired knowledge or information to aid mangers or decision makers for making the efficient decisions from a very large database is a critical issue in recent years. [1-4, 6, 8, 13]. Among them, sequential patterns mining considers the order sequence data such as Web-click logs, network flow logs or DNA sequences, which is the major issue in real-world applications. For basket analysis, sequential patterns mining can also be used to mine the purchased behaviors of customers to predict whether there is a high probability that when customers buy some products, they will buy some other products in later transactions.

J.-S. Pan et al. (eds.), *Intelligent Data Analysis and Its Applications, Volume 1*,
Advances in Intelligent Systems and Computing 297,
DOI: 10.1007/978-3-319-07776-5_15, © Springer International Publishing Switzerland 2014

Agrawal et al. first proposed AprioriAll algorithm [3] to level-wisely mine sequential patterns in a batch way. Various algorithms applied in different applications of sequential patterns mining have been proposed to handle the static database [10, 15-17]. Discovered sequential patterns may, however, become invalid since sequences are changed in dynamic databases. Developing an efficient approach to maintain and update the discovered sequential patterns is a critical issue in real-world applications. Lin et al. proposed an incremental FASTUP algorithm [14] to maintain the discovered sequential patterns. Lin et al. designed a fast updated sequential pattern (FUSP)-tree structure and algorithms to handle the sequential patterns in dynamic databases [11-12].

The FASTUP or FUSP-tree algorithms are, however, required to re-scan the original database if the small itemsets or sequences are necessary to be maintained and updated. Hong et al. then extended the pre-large concepts [7] of association-rule mining to level-wisely maintain the sequential patterns in dynamic databases [9]. In this paper, the pre-large concepts are adopted in the FUSP tree to efficiently maintain the discovered sequential patterns for sequence insertion. A FUSP-tree structure is first built to keep only large sequences in the tree, and the pre-large sequences are mined out and kept in a set for later maintenance. The proposed incremental algorithm divides the 1-sequences in the newly inserted sequences into three parts with nine cases. Each case is then performed by the designed algorithm to maintain and update the built FUSP tree. Experimental results also then show that the proposed algorithm has a good performance for incrementally handling new inserted sequences.

## 2 Review of Related Works

In this section, sequential patterns mining and the pre-large concepts are briefly reviewed.

### 2.1 Sequential Patterns Mining

In the past, Agrawal et al. designed an AprioriAll algorithm [3] to level-wisely mine sequential patterns in a static database. Lin et al. thus proposed an incremental FASTUP algorithm [14] to maintain sequential patterns in dynamic databases. The FASTUP algorithm is, however, required to re-scan the original database if it is necessary to maintain the discovered sequential pattern, which is large in the added sequences but small in the original database. Hong et al. extended the pre-large concepts of association-rule mining [7] to handle the sequential patterns whether for the sequence insertion [9] or deletion [5]. It is also based on Apriori-like approach [2] to generate-and-test the candidates for deriving the desired sequential patterns. Lin first designed a fast updated sequential pattern (FUSP)-tree and developed the algorithms for efficiently handling sequence insertion in incremental mining [11-12]. Based on the built FUSP tree structure, the discovered sequential patterns can be thus easier maintained.

The FUSP tree [11] is used to store customer sequences with only large 1-sequences in the original database. Based on the FUSP tree, the complete sequential patterns can be derived from it without level-wisely rescanning the original database. An example is given to show the FUSP tree. Assume a database shown in Table 1 is used to build the FUSP tree.

**Table 1.** An example

| CID | Customer Sequence |
|---|---|
| 1 | (AC)(F) |
| 2 | (CE)(D)(H) |
| 3 | (AB)(D) |
| 4 | (C)(D)(EF)(H) |
| 5 | (AC)(GI) |
| 6 | (BC)(DEH) |
| 7 | (A)(D)(H) |
| 8 | (AF)(DG) |
| 9 | (A)(D)(EH) |
| 10 | (C)(F)(BD) |

Also assume that the upper support threshold is set at 60%, and the lower support threshold is set at 30%. The large 1-sequnces are (A), (C), and (D) from which Header_Table can be constructed. The pre-large 1-sequences are then kept in a set of *Pre_Seq*s = {B:3, E:4, F:4, H:5}. The built FUSP tree from the database is shown in Figure 1. In Figure 1, only large 1-sequences are kept in the FUSP tree. The link between two connected nodes is marked by the symbol ***s*** if the sequence is within the sequence relation in a sequence; otherwise, the link is marked by the symbol ***i*** if the sequence is within the itemset relation in a sequence. The built Header_Table is used as an index table to find appropriate items or sequences in the tree. It keeps the large 1-sequences initially in descending order of their counts. Infrequent ones are not used to build the tree. After all customer sequences are processed, the FUSP tree is completely constructed.

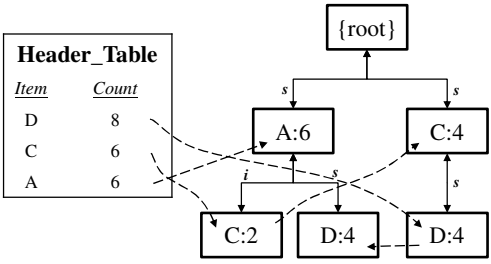


**Fig. 1.** Initial constructed FUSP tree with its Header_Table

### 2.2 Pre-large Concepts

A pre-large sequence [9] is not truly large, but has highly probability to be large when the database is updated. A lower support threshold and an upper support threshold are used to respectively define the pre-large and the large concepts. The pre-large concepts act like buffers to reduce the movement of sequences directly from large to small and vice-versa in the maintenance process. Considering an original database and some customer sequences are inserted into the original database, three parts with nine cases in Figure 2 may arise.

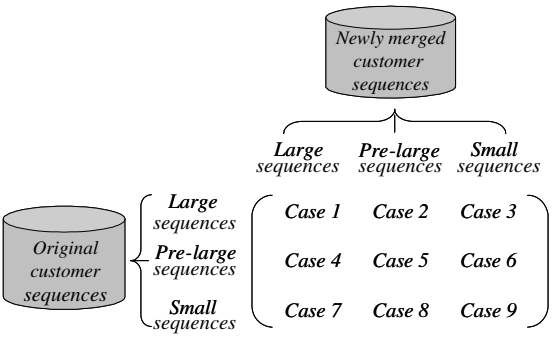


**Fig. 2.** Nine cases arising from the original database and the inserted sequences

Cases 1, 5, 6, 8 and 9 will not affect the final large sequences. Cases 2 and 3 may remove existing large sequences. Cases 4 and 7 may add new large sequences. If we retain all large and pre-large sequences with their counts in the original database, cases 2, 3 and 4 can be easily handled. In the maintenance phase, the ratio of newly added customer sequences to original customer sequences is usually very small. This is more apparent when the database grows larger. It has been formally shown that the sequences in Case 7 cannot possibly be large in the updated database as long as the number of customer sequences is small compared to the number of customer sequences in the original database. The formula [9] is shown below.

$$t \leq \frac{(S_u - S_l) \times d}{1 - S_u} - \frac{q \times S_u}{1 - S_u},$$

where $t$ is the number of the newly added customer sequences, and $q$ be the number of newly added customer sequences belongs to old customers, $S_u$ is the upper threshold, $S_l$ is the lower threshold, and $d$ is the number of customer sequences in the original database.

## 3 Proposed an Incremental Algorithm

A fast updated sequential pattern (FUSP)-tree [11] must built in advance to keep the large sequences from the original database before new transactions or sequences

come. The pre-large 1-sequences are also kept in a set for later maintenance process. When the sequences are inserted into the original database, the proposed incremental algorithm is then performed below in details.

***Proposed algorithm*:**

**INPUT:** An old database consisting of ($d$ + $t$) sequences, its corresponding Header_Table, a set of *Pre_Seqs* to keep the pre-large 1-sequeces, its corresponding FUSP tree, a lower support threshold $S_l$, an upper support threshold $S_u$, and a set of $t$ new inserted sequences.

**OUTPUT:** An updated FUSP tree.

**STEP 1:** Set $b = b + q$, where $q$ number of newly inserted sequences belonging to old customers in the original database;

**STEP 1:** Calculate the safety bound $f$ to determine whether the original database is required to be re-scanned of the new inserted transactions by the formula as [9]:

$$t \le \frac{(S_u - S_l)\times d}{1-S_u} - \frac{b\times S_u}{1-S_u}.$$

**STEP 2:** Scan the new sequences to get all 1-sequences with their frequencies.

**STEP 3:** Divide the1-sequences in STEP 2 into three parts with nine cases according to whether they are large (appears in the Header_Table), pre-large (appears in the set of *Pre_Seqs*) or small (not appears in the Header_Table either in the set of *Pre_Seqs)* in the original database.

**STEP 4:** For each 1-sequence $s$ which is large in the original database, do the following substeps (**Cases 1**, **2** and **3**):

Substep 4-1: Set the count $S^U(s)$ of $s$ in the updated database as:

$$S^U(s) = S^D(s) + S^T(s),$$

where $S^D(s)$ is the frequency of $s$ in the Header_Table (original database) and $S^T(s)$ is the frequency of $s$ in the new transactions.

Substep 4-2: If $S_u \le S^U(s)/(d+c+t-b)$, update the frequency of $s$ in the Header_Table as $S^U(s)$; put $s$ in the set of *Insert_Seqs*, which will be further processed in STEP 8. Otherwise, if $S_l < S^U(s)/(d+c+t-b) \le S_u$, remove $s$ from the Header_Table; connect the parent node of $s$ to its child nodes directly in the FUSP tree; put $s$ in the set of *Pre_Seqs* with its updated frequency $S^U(s)$. Otherwise, 1-sequence $s$ becomes small after the database is updated; remove $s$ from the Header_Table and connect each parent node of $s$ directly to its child nodes in the FUSP tree.

**STEP 5:** For each 1-sequence $s$ which is pre-large in the original database, do the following substeps (**Cases 4**, **5** and **6**):

Substep 5-1: Set the new count $S^U(s)$ of $s$ in the updated database as:

$$S^U(s) = S^D(s) + S^T(s).$$

Substep 5-2: If $S_u \leq S^U(s)/(d+c+t-b)$, 1-sequence $s$ will be large after the database is updated; remove $s$ from the set of *Pre_Seqs*; put $s$ with its updated frequency in the set of *Branch_Seqs*; put $s$ in the set of *Insert_Seqs*. Otherwise, if $S_l < S^U(s)/(d+c+t-b) \leq S_u$, 1-sequence $s$ still remains pre-large after the database is updated; update $s$ with its new frequency $S^U(s)$ in the set of *Pre_Seqs*. Otherwise, remove 1-sequence $s$ from the set of *Pre_Seqs*.

**STEP 6:** For each 1-sequence $s$ which is neither large nor pre-large in the original database but large or pre-large in the new transactions (**Cases 7** and **8**), put $s$ in the set of *Rescan_Seqs*, which is used when rescanning the database in STEP 7 is necessary.

**STEP 7:** If $t + c \leq f - h$ or the set of *Rescan_Seqs* is *null*, then do nothing; Otherwise, do the following substeps for each1-sequence $s$ in the set of *Rescan_Seqs*:

Substep 7-1: Rescan the original database to decide the original count $S^D(s)$ of $s$.

Substep 7-2: Set the new count $S^U(s)$ of $s$ in the updated database as:

$$S^U(s) = S^D(s) + S^T(s),$$

Substep 7-3: If $S_u \leq S^U(s)/(d+c+t-b)$, 1-sequence $s$ will become large after the database is updated; put $s$ in both the sets of *Insert_Seqs* and *Branch_Seqs*; insert the items in the *Branch_Seqs* to the end of the Header_Table according to the descending order of their updated frequencies. Otherwise, if $S_l < S^U(s)/(d+c+t-b) \leq S_u$, 1-sequence $s$ will become pre-large after the database is update; put $s$ with its updated frequency in the set of *Pre_Seqs*. Otherwise, do nothing.

Substep 7-4: For each original transaction with a 1-sequence $s$ existing in the *Branch_Seqs*, if $s$ has not been at the corresponding branch of the FUSP tree for the transaction, insert $s$ at the end of the branch and set its count as 1; otherwise, add 1 to the count of the node $s$.

**STEP 8:** Insert the sequences in the *Branch_Seqs* to the end of Header_Table according to the descending order of their updated counts. For each original sequence with a sequence $s$ existing in the *Branch_Seqs*, if $s$ has not been at the corresponding branch of the FUSP tree for the processed sequence, insert $s$ to its corresponding position and set its count as 1; otherwise, add 1 to the count of the node $s$.

**STEP 9:** For each new transaction with a 1-sequence $s$ existing in the *Insert_Seqs*, if $s$ has not been at the corresponding branch of the FUSP for the new sequence, insert $s$ to its corresponding position and set its count as 1; otherwise, add 1 to the count of the node $s$.

**STEP 10:** If $t + c > f - h$, then set $d = d + t + c$ and set $c = 0$; otherwise, set $c = t + c$.

In STEP 7, a corresponding branch is the branch generated from the large 1-sequences in a transaction and corresponding to the order of 1-sequences appeared in the Header_Table. After STEP 10, the final updated FUSP tree is maintained by the proposed algorithm. The new transactions can then be integrated into the original database. Desired sequential patterns can then be found by the FUSP-growth mining algorithm [11].

## 4 An Example

In this section, an example is given to illustrate the proposed incremental algorithm for maintaining the discovered sequential patterns based on the built FUSP tree [11]. An original database was shown in Table 1, which consists of 10 customer sequences with nine purchased items. In this example, an upper support $S_u$ and the lower support $S_l$ were respectively set at 30% and 60%. The built FUSP tree was shown in Figure 2. Suppose three new customer sequences shown in Table 2 are inserted into the original database. The proposed incremental algorithm is then performed by the designed steps. The global variables $c$ and $b$ are initially set at 0.

**Table 2.** Three added customer sequences

| CID | Customer sequence |
|---|---|
| 5 | (CH)(I) |
| 11 | (A)(I)(H) |
| 12 | (A)(G)(H) |

The value of the first term in Formula 1 [9] is calculated as:

$$f = \frac{(S_u - S_l) \times d}{1 - S_u} = \frac{(0.6 - 0.3) \times 10}{1 - 0.6} = 7.5 \cdot$$

Since only one customer sequence with $CID$ = 5 in Table 2 belongs to old customers in Table 1, $q$ is thus set at 1, and $b = (b + q)$ (= 0 + 1) (= 1). The value of the second term in Formula 1 [9] is calculated as:

$$h = \frac{b \times S_u}{1 - S_u} = \frac{1 \times 0.6}{1 - 0.6} = 1.5 \cdot$$

The customer sequences in Table 2 are firstly scanned to get the 1-sequences and their counts. After that, 1-sequences in the added customer sequences are then divided into three parts with nine cases. The designed algorithm is then performed to maintain and update the built FUSP tree. The final updated FUSP tree is then shown in Figure 3.

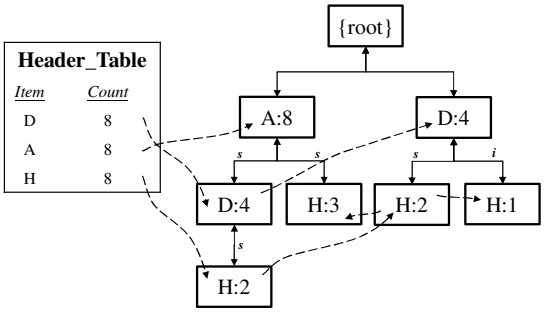


**Fig. 3.** Initial constructed FUSP tree with its Header_Table

Based on the FUSP tree in Figure 3, the desired large sequences then be found by the FUSP-growth approach [11].

## 5 Experimental Results

Experiments were made to compare the performance of FUSP-TREE-BATCH algorithm [11], FUSP-TREE-INS algorithm [11], and the proposed incremental algorithm. A real database called BMSWebView-1 [18] is used to evaluate the performance of the proposed incremental algorithm. In the experiments, the execution time and the number of tree nodes are then compared to show the performance of the proposed incremental algorithm at different number of minimum support thresholds. To evaluate the performance of the proposed algorithm at different minimum support thresholds, $S_l$ values are respectively set at $S_u$ values minus 0.21% for BMSWebview-1 database. The results are respectively shown from Figures 4 to 5.

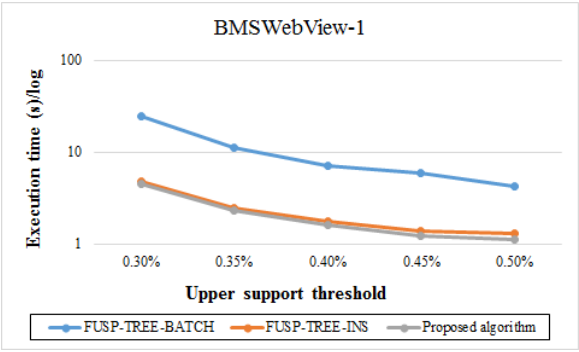


**Fig. 4.** Comparisons of execution times

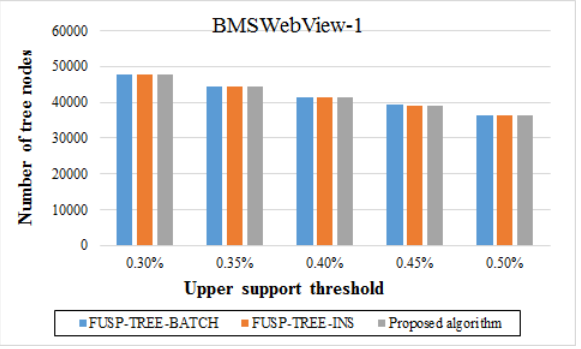


**Fig. 5.** Comparisons of tree nodes

From Figures 4 and 5, it can obvious to see that the proposed algorithm runs faster than the FUSP-TREE-BATCH and FUSP-TREE-INS algorithms and generates nearly the same number of tree nodes compared to the other two algorithms. The proposed algorithm can thus acceptable in terms of execution time and number of tree nodes.

## 6 Conclusion

In this paper, a pre-large concepts are adopted for efficiently maintaining and updating the built FUSP tree for sequence insertion in dynamic databases. A FUSP-tree structure is used to make the updating process become easier. From the experiments, the proposed incremental algorithm can thus achieve a good trade-off between execution time and tree complexity.

**Acknowledgement.** This research was partially supported by the Shenzhen Peacock Project, China, under grant KQC201109020055A, by the Natural Scientific Research Innovation Foundation in Harbin Institute of Technology under grant HIT.NSRIF.2014100, by the National Science Council of the Republic of China under Contract no. NSC 101-2628-E-004-001-MY2, and by the Shenzhen Strategic Emerging Industries Program under grant ZDSY20120613125016389.

## References

1. Agrawal, R., Imielinski, T., Swami, A.: Database mining: A performance perspective. IEEE Transactions on Knowledge and Data Engineering 5, 914–925 (2006)
2. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules in large databases. In: The International Conference on Very Large Data Bases, pp. 487–499 (1994)
3. Agrawal, R., Srikant, R.: Mining sequential patterns. In: The International Conference on Data Engineering, pp. 3–14 (1995)
4. Chen, M.S., Han, J., Philips Yu, S.: Data mining: An overview from a database perspective. IEEE Transactions on Knowledge and Data Engineering 8, 866–883 (1996)

5. Wang, C.Y., Hong, T.P., Tseng, S.S.: Maintenance of sequential patterns for record deletion. In: IEEE International Conference on Data Mining, pp. 536–541 (2001)
6. Han, J., Pei, J., Yin, Y., Mao, R.: Mining frequent patterns without candidate generation: A frequent-pattern tree approach. Data Mining and Knowledge Discovery 8, 53–87 (2004)
7. Hong, T.P., Wang, C.Y., Tao, Y.H.: A new incremental data mining algorithm using pre-large itemsets. Intelligent Data Analysis 5, 111–129 (2001)
8. Hong, T.P., Lin, C.W., Wu, Y.L.: Incrementally fast updated frequent pattern trees. Expert Systems with Applications 34, 2424–2435 (2008)
9. Hong, T.P., Wang, C.Y., Tseng, S.S.: An incremental mining algorithm for maintaining sequential patterns using pre-large sequences. Expert Systems with Applications 38, 7051–7058 (2011)
10. Kim, C., Lim, J.H., Ng, R.T., Shim, K.: Squire: Sequential pattern mining with quantities. Journal of Systems and Software 80, 1726–1745 (2007)
11. Lin, C.W., Hong, T.P., Lu, W.H., Lin, W.Y.: An incremental fusp-tree maintenance algorithm. In: The International Conference on Intelligent Systems Design and Applications, pp. 445–449 (2008)
12. Lin, C.W., Hong, T.P., Lu, W.H.: An efficient fusp-tree update algorithm for deleted data in customer sequences. In: International Conference on Innovative Computing, Information and Control, pp. 1491–1494 (2009)
13. Lin, C.W., Hong, T.P.: A survey of fuzzy web mining. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery 3, 190–199 (2013)
14. Lin, M.Y., Lee, S.Y.: Incremental update on sequential patterns in large databases. In: IEEE International Conference on Tools with Artificial Intelligence, pp. 24–31 (1998)
15. Nakagaito, F., Ozaki, T., Ohkawa, T.: Discovery of quantitative sequential patterns from event sequences. In: IEEE International Conference on Data Mining Workshops, pp. 31–36 (2009)
16. Pei, J., Han, J., Mortazavi-Asl, B., Wang, J., Pinto, H., Chen, Q., Dayal, U., Hsu, M.C.: Mining sequential patterns by pattern-growth: The prefixspan approach. IEEE Transactions on Knowledge and Data Engineering 16, 1424–1440 (2004)
17. Ren, J.M., Jang, J.R.: Discovering time-constrained sequential patterns for music genre classification. IEEE Transactions on Audio, Speech, and Language Processing 20, 1134–1144 (2012)
18. Zheng, Z., Kohavi, R., Mason, L.: Real world performance of association rule algorithms. In: ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 401–406 (2001)