# A Modified Maintenance Algorithm for Updating FUSP Tree in Dynamic Database

Ci-Rong Li[1], Chun-Wei Lin[2,3,*], Wensheng Gan[2], and Tzung-Pei Hong[4,5]

[1] Faculty of Management,
Fuqing Branch of Fujian Normal University, Fujian, China
[2] Innovative Information Industry Research Center (IIIRC)
[3] 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, China
[4] Department of Computer Science and Information Engineering
National University of Kaohsiung, Kaohsiung, Taiwan, R.O.C.
[5] Department of Computer Science and Engineering
National Sun Yat-sen University, Kaohsiung, Taiwan, R.O.C.
{cirongli,wsgan001}@gmail.com, jerrylin@ieee.org,
tphong@nuk.edu.tw

**Abstract.** In the past, we proposed a pre-large FUSP tree to preserve and maintain both large and pre-large sequences in the built tree structure. In this paper, the pre-large concept is also adopted for maintaining and updating the FUSP tree. Only large sequences are kept in the built tree structure for reducing computations. The PreFUSP-TREE-MOD maintenance algorithm is proposed to reduce the rescans of the original database due to the pruning properties of pre-large concept. When the number of modified sequences is smaller than the safety bound of the pre-large concept, better results can be obtained by the proposed PreFUSP-TREE-MOD maintenance algorithm for sequence modification in the dynamic database.

**Keywords:** Data mining, pre-large concept, dynamic database, FUSP tree, sequence modification.

## 1 Introduction

It is a critical issue to efficiently mine the desired knowledge or information to aid managers in decision-making from a very large database. The mostly common knowledge can be classified as association-rule mining [1, 2, 6], classification [12], clustering [5], and sequential pattern mining [4, 11, 21], among others [16-18]. Finding sequential patterns in temporal transaction database has become an important issue since it allows the modeling of customer behaviors.

[*] Corresponding author.

Agrawal et al. then proposed AprioriAll algorithm [4] for mining sequential patters in a level-wise way. Although customer behaviors can be efficiently extracted by several sequential-pattern-mining algorithms [4, 11, 21] to assist managers in making decisions, the discovered sequential patterns may become invalid since sequences are inserted [13, 20], deleted [15] or modified [14] in real-world applications. Developing an efficient approach to maintain and update sequential patterns is thus a critical issue in real-world applications.

Few studies [13-15, 20] are, however, designed to handle the sequential patterns in the dynamic database compared to those on maintaining association rules. In the past, a pre-large concept [9] was adopted to maintain the build pre-large FUSP tree. Since the pre-large sequences are kept in the tree structure, more computations are required to maintain both the large and pre-large sequences for finding the corresponding branches [19]. In this paper, the pre-large concept is also adopted in the FUSP tree but only large sequences are kept in the built tree structure. A pre-large fast updated sequential pattern tree for sequence modification (PreFUSP-TREE-MOD) maintenance algorithm is designed to easier facilitate the updating process of the built FUSP tree. A FUSP tree [13] is initially built to completely preserve customer sequences with only large items in the given databases. When some sequences are modified from the original database, the proposed PreFUSP-TREE-MOD maintenance algorithm is then processed to maintain the built FUSP tree and the Header_Table. Experimental results show that the proposed PreFUSP-TREE-MOD maintenance algorithm balances the trade-off between execution time and tree complexity and has the better performance than the batch methods.

## 2 Review of Related Works

In this section, works related to mining sequential patterns, FUSP-tree structure, and maintenance approach of pre-large concept are briefly reviewed.

### 2.1 Mining Sequential Patterns

Agrawal et al. first proposed the AprioriAll algorithm [4] for level-wisely mining sequential patterns in a static database. Conventional approaches may re-mine the entire database to update the sequential patterns in the dynamic database. Lin et al. thus proposed the FASTUP algorithm [20] to maintain sequential patterns. Hong and Wang et al. then extended the pre-large concept of association-rule mining [9] to handle the sequential patterns [10, 22]. Cheng et al. proposed the IncSpan (incremental mining of sequential patterns) algorithm for efficiently maintaining sequential patterns in a tree structure [7]. Lin et al. designed a fast updated sequential pattern (FUSP)-tree and developed the algorithms for efficiently handling sequence insertion [13], sequence deletion [15], and sequence modification [14] to maintain and update the built FUSP tree for efficiently discovering sequential patterns in the dynamic database. Other algorithms for mining various sequential patterns are still developed in progress [11, 20].

## 2.2 FUSP-Tree Structure

The FUSP tree [13] is used to store customer sequences with only large 1-sequences in the original database. An example is given to briefly show the FUSP tree. Assume a database shown in Table 1 is used to build the FUSP tree.

**Table 1.** Original customer sequences

| Customer ID | Customer sequence |
|---|---|
| 1 | (AC)(E)(I) |
| 2 | (A)(I)(B) |
| 3 | (BE)(CD) |
| 4 | (AC)(DF) |
| 5 | (A)(B)(F) |
| 6 | (A)(B)(D)(EF) |
| 7 | (AC)(B)(E) |
| 8 | (AC)(E)(F)(G) |
| 9 | (BG)(D) |
| 10 | (DE)(GH) |

Also assume that the minimum support threshold is set at 60%. For the given database, the large 1-sequnces are (A), (B), and (E), from which Header_Table can be constructed. The results are shown in Figure 1.

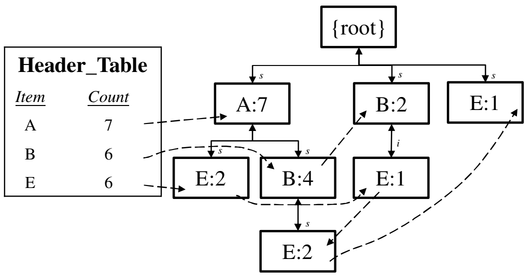

**Fig. 1.** Initial constructed FUSP tree with its Header_Table.

In Figure 1, only the customer sequences with large items (1-sequences) are stored in the FUSP tree. The link between two connected nodes is marked by the symbol ***s*** (representing the sequence relation) if the sequence is within the sequence relation in a sequence; otherwise, the link is marked by the symbol ***i***, which indicates the sequence is within the itemset relation in a sequence [7]. A FP-growth-like algorithm can be used to mine the sequential patterns [8].

## 2.3 Maintenance Approach of Pre-large Concept

A pre-large sequence is not truly large, but has highly probability to be large when the database is updated [9-10]. A lower support threshold and an upper support threshold are used to define pre-large concept. Pre-large sequences act like buffers and are used to reduce the movement of sequences directly from large to small and vice-versa in the maintenance process. Therefore, when few sequences are modified, the originally small sequences will at most become pre-large and cannot become large, thus reducing the amount of rescanning necessary. Considering an original database and some customer sequences to be modified, the following nine cases in Figure 2 may arise.

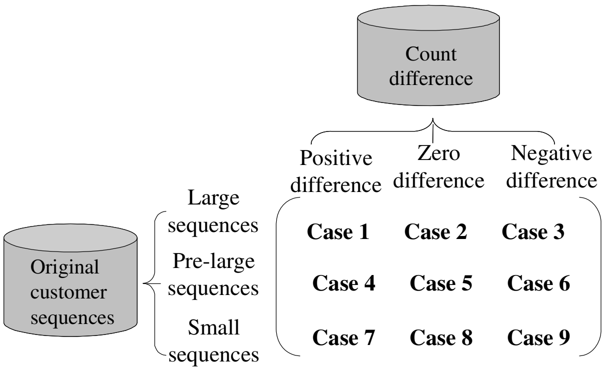

**Fig. 2.** Nine cases arising from the original database and the modified sequences

Cases 1, 2, 5, 6, 8 and 9 above will not affect the final sequential patterns. Case 3 may remove some existing sequential patterns, and cases 4 and 7 may add some new sequential patterns. It has been formally shown that a sequence in case 7 cannot possibly be large for the entire updated database when the number of modified sequences is smaller than the number $f$ shown below [22]:

$$f = \lfloor (S_u - S_l)d \rfloor,$$

where $f$ is the safety number of modified sequences, $S_u$ is the upper threshold, $S_l$ is the lower threshold, and $d$ is the number of customer sequences in the original database.

## 3 Proposed PreFUSP-TREE-MOD Algorithm

In this paper, the pre-large concept is adopted to maintain and update the built FUSP tree [13] for sequence modification [14] in dynamic database. A FUSP tree must be built in advance from the initial original database. When sequences are modified from the original database, the FUSP tree and its corresponding Header_Table are required to be modified and updated by the proposed PreFUSP-TREE-MOD maintenance algorithm. The details of the proposed algorithm are stated below.

***Proposed PreFUSP-TREE-MOD maintenance algorithm*:**

**INPUT:** An old database consisting of $d$ sequences, its corresponding Header_Table, and the built FUSP tree, a lower support threshold $S_l$, an upper support threshold $S_u$, a set of pre-large 1-sequences *Prelarge_Seqs* from the original database, and a set of $t$ modified sequences.

**OUTPUT:** An updated FUSP tree.

**STEP 1:** Calculate the safety number *f* for modified sequences as [22]:

$$f = \lfloor (S_u - S_l)d \rfloor.$$

**STEP 2:** Find all 1-sequences in the *t* modified sequences before and after modification. Denote them as a set of modified 1-sequences, *M*.

**STEP 3:** Find the count difference (including zero) of each 1-sequence in *M* for the modified sequences.

**STEP 4:** Divide the 1-sequences in *M* into three parts according to whether they are large, pre-large or small in the original database.

**STEP 5:** For each 1-sequence *s* in *M* which is large in the original database (appearing in the Header_Table), do the following substeps (for **cases 1**, **2** and **3**):

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

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

where $S^D(s)$ is the count of *s* in the Header_Table (from the original database) and $S^M(s)$ is the count difference of *s* from sequence modification.

Substep 5-2: If $S^U(s) \geq (S_u \times d)$, update the count of *s* in the Header_Table as $S^U(s)$, and put *s* in both the sets of *Increase_Seqs* and *Decrease_Seqs*, which will be further processed to update the FUSP tree in STEP 9;

Otherwise, If $(S_l \times d) \leq S^U(s) \leq (S_u \times d)$, connect each parent node of *s* directly to its corresponding child nodes; remove *s* from the FUSP tree and the Header_Table; put *s* in the set of *Prelarge_Seqs* with its updated count $S^U(s)$;

Otherwise, *s* is small after the database is updated; connect each parent node of *s* directly to its corresponding child nodes and remove *s* from the FUSP tree and the Header_Table;

**STEP 6:** For each 1-sequence *s* in *M* which is pre-large in the original database (in the set of pre-large sequences), do the following substeps (for cases 4, 5 and 6):

Substep 6-1: Set the new count $S^U(s)$ of *s* in the entire updated database as:

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

Substep 6-2: If $S^U(s) \geq (S_u \times d)$, 1-sequence *s* will become large after the database is updated; remove *s* from the set of *Prelarge_Seqs*, put *s* in the set of *Branch_Seqs* with its new count $S^U(s)$, and put *s* in the set of *Increase_Seqs*;

Otherwise, if $(S_l \times d) \leq S^U(s) \leq (S_u \times d)$, 1-sequence *s* is still pre-large after the database is updated; update *s* with its new count $S^U(s)$ in the set of *Prelarge_Seqs*;

Otherwise, 1-sequence *s* is small after the database is updated; remove *s* from the set of *Prelarge_Seqs*.

**STEP 7:** For each 1-sequence *s* which is neither large nor pre-large in the original database but has positive count difference in *M* (for **case 7**), put *s* in the set of *Rescan_Seqs*, which is used when rescanning the database in STEP 8 is necessary.

**STEP 8:** If $(t + c) \leq f$ or the set of *Rescan_Seqs* is ***null***, do nothing; Otherwise, do the following substeps for each 1-sequence $s$ in the set of *Rescan_Seqs*:

Substep 8-1: Rescan the original database to determine the original count $S^D(s)$ of $s$ (before modification).

Substep 8-2: Set the new count $S^U(s)$ of $s$ in the entire updated database as:

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

Substep 8-3: If $S^U(s) \geq (S_u \times d)$, 1-sequence $s$ will become large after the database is updated; put $s$ in both the sets of *Increase_Seqs* and *Branch_Seq*s;
Otherwise, if $(S_l \times d) \leq S^U(s) \leq (S_u \times d)$, 1-sequence $s$ will become pre-large after the database is updated; put $s$ in the set of *Prelarge_Seqs* with its updated count $S^U(s)$;
Otherwise, neglect $s$.

**STEP 9:** For each updated sequence before modification ($T$) and with a 1-sequence $J$ existing in the *Decrease_Seqs*, find the corresponding branch of $J$ in the FUSP tree and subtract 1 from the count of the $J$ node in the branch; if the count of the $J$ node becomes zero after subtraction, remove node $J$ from its corresponding branch and connect the parent node of $J$ directly to the child node of $J$.

**STEP 10:** Insert the 1-sequences in the *Branch_Seqs* to the end of the Header_Table according to the descending order of their counts.

**STEP 11:** If the set of *Branch_Seqs* is ***null***, nothing is done in this step;
Otherwise, for each unmodified sequence ($D^{-}$) with a 1-sequence $J$ in *Branch_Seqs*, if $J$ has not been at the corresponding branch of the FUSP tree, then insert $J$ at the end of the branch and set its count as 1; Otherwise, add 1 to the count of the node $J$.

**STEP 12:** For each updated sequence after modification ($T'$) with a 1-sequence $J$ existing in *Increase_Seqs*, if $J$ has not been at the corresponding branch of the FUSP tree, insert $J$ at the end of the branch and set its count as 1;
Otherwise, add 1 to the count of the $J$ node.

**STEP 13:** If $(t + c) > f$, set $c = 0$; otherwise, set $c = t + c$.

After STEP 13, the final updated FUSP tree is thus maintained by the proposed PreFUSP-TREE-MOD maintenance algorithm for sequence modification. Based on the FUSP tree, the desired large sequences can then be found by the FP-growth-like mining approach [8].

## 4 An Illustrated Example

A FUSP tree [13] was firstly built from the original database shown in Figure 2. An upper support threshold was set at 60% and the lower support threshold is set at 30%.

The pre-large 1-sequences with their counts are then found and then put in the set of *Prelarge_Seqs*. The results are shown in Table 2, and the modified customer sequences are shown in Table 3.

**Table 2.** Pre-large 1-sequences

| 1-sequence | Count |
|---|---|
| (C) | 5 |
| (D) | 5 |
| (F) | 4 |
| (G) | 3 |

**Table 3.** Modified customer sequences

| Cust_ID | Before modification | After modification |
|---|---|---|
| 2 | (A)(I)(B) | (A)(BC)(FH) |
| 10 | (DE)(GH) | (BD)(H) |

The safety bound for the modified sequences is calculated as $f = \lfloor (0.6-0.3)\times 10 \rfloor$ (= 3). The proposed PreFUSP-TREE-MOD maintenance algorithm is the performed to maintain and update the FUSP tree by the following steps. The count differences of the sequences before and after modification are then calculated. After that, the results of count difference are then divided into three parts according to whether they are large (appearing in Header_Table), pre-large (appearing in the set of *Prelarge_Seqs*) or small in the original customer sequences.

For each 1-sequence from the divided part, which is large in the original database (appearing in Header_Table shown in Figure 1), is then processed. For each 1-sequence from the divided part, which is pre-large in the original database (appearing in *Prelarge_Seqs* shown in Table 2), is then processed. For each 1-sequence from the divided part, which is small in the original database (not appearing either in the Header_Table or the set of *Prelarge_Seqs*) but has positive count difference is then processed. Since only two sequences are modified from the original database, which is smaller than the safety bound (2 < 3); nothing has to be processed for the set of *Rescan_Seqs*.

The 1-sequences in the *Decrease_Seqs* are then processed to subtract their counts before sequence modification in the built FUSP tree. The 1-sequenecs in the *Branch_Seqs* are then sorted in descending order of their counts. The 1-sequences from *Branch_Seqs* are then inserted into the end of the Header_Table. The corresponding branches of 1-sequences in *Branch_Seqs* are then found from the unmodified sequences in the original database. The 1-sequences in the *Increase_Seqs* are then processed to find the corresponding branches after sequence modification. After that, the final result of the FUSP tree is shown in Figure 3. Since the number of the modified sequences is 2 in this example, variable *c* is then accumulated as (0 + 2) (= 2), which indicates that one more sequence can be modified without rescanning the original database for case 7.

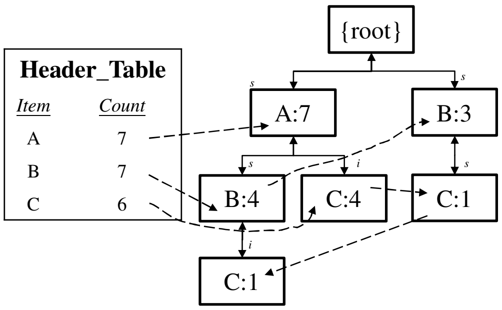

**Fig. 3.** Final updated FUSP tree

## 5 Experimental Results

Experiments were made to compare the performance of the GSP algorithm [21], the FUSP-TREE-BATCH [13] algorithm, the pre-large maintenance algorithm for sequence modification (defined as PRE-APRIORI-MOD) [22], and the proposed PreFUSP-TREE-MOD maintenance algorithm. The S10I4N1KD10K is a simulated database generated from IBM Quest Dataset Generator [3]. The percentage of the modified sequences is set at 1%. The $S_l$ values are respectively set as 50% of $S_u$ values for the PRE-APRIORI-MOD algorithm and the proposed PreFUSP-TREE-MOD maintenance algorithm. The performance of execution time is shown in Figure 4.

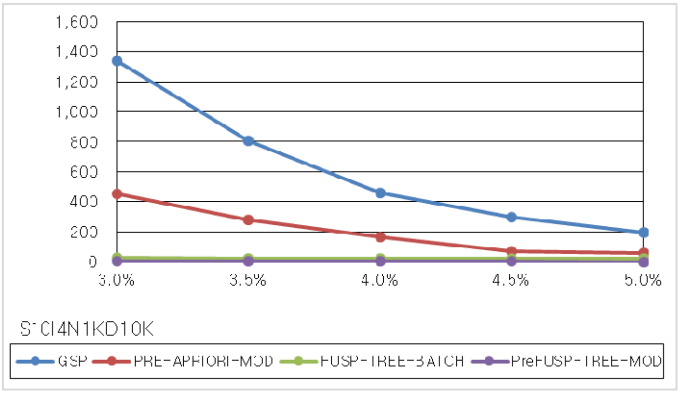

**Fig. 4.** The comparisons of the execution time

From Figure 4, the proposed PreFUSP-TREE-MOD maintenance algorithm has the better performance than the other algorithms. The number of tree nodes are also compared to show the performance. Since only the tree-based approaches involved tree nodes for the comparisons, the FUSP-TREE-BATCH algorithm and the proposed PreFUSP-TREE-MOD maintenance algorithm are then compared and shown in Figure 5.

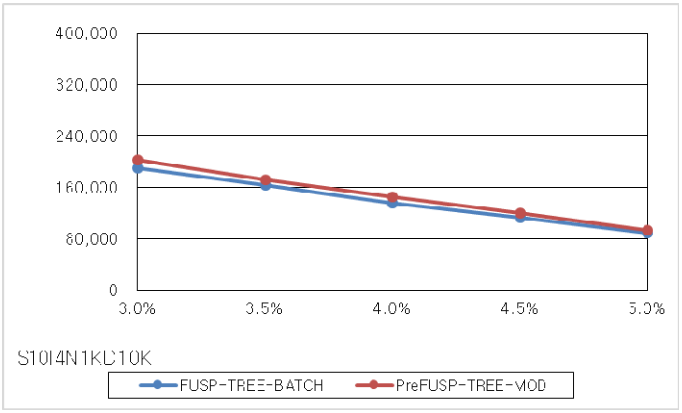

**Fig. 5.** The comparisons of the tree nodes

From Figure 5, it can be seen that the proposed PreFUSP-TREE-MOD maintenance algorithm generates nearly the same number of tree nodes.

## 6 Conclusion

In this paper, the pre-large fast updated sequential pattern tree for sequence modification (PreFUSP-TREE-MOD) maintenance algorithm is proposed to efficiently and effectively maintain the FUSP tree based on pre-large concept for deriving sequential patterns. From the experiments, the proposed PreFUSP-TREE-MOD maintenance 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 BD29100003, 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 (1993)
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.: Quest synthetic data generator (1994), `http://www.Almaden.ibm.com/cs/quest/syndata.html`
4. Agrawal, R., Srikant, R.: Mining sequential patterns. In: The International Conference on Data Engineering, pp. 3–14 (1995)
5. Berkhin, P.: A survey of clustering data mining techniques. In: Grouping Multidimensional Data, pp. 25–71 (2006)

6. Chen, M.S., Han, J., Yu, P.S.: Data mining: An overview from a database perspective. IEEE Transactions on Knowledge and Data Engineering 8, 866–883 (1996)
7. Cheng, H., Yan, X., Han, J.: Incspan: Incremental mining of sequential patterns in large database. In: ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 527–532 (2004)
8. 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)
9. 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)
10. 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)
11. Huang, Z., Shyu, M.L., Tien, J.M., Vigoda, M.M., Birnbach, D.J.: Prediction of uterine contractions using knowledge-assisted sequential pattern analysis. IEEE Transactions on Biomedical Engineering 60, 1290–1297 (2013)
12. Kotsiantis, S.B.: Supervised machine learning: A review of classification techniques. In: The Conference on Emerging Artificial Intelligence Applications in Computer Engineering: Real Word AI Systems with Applications in eHealth, HCI, Information Retrieval and Pervasive Technologies, pp. 3–24 (2007)
13. 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)
14. Lin, C.W., Hong, T.P., Lu, W.H., Chen, H.Y.: An fusp-tree maintenance algorithm for record modification. In: IEEE International Conference on Data Mining Workshops, pp. 649–653 (2008)
15. 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)
16. Lin, C.W., Hong, T.P., Lu, W.H.: An effective tree structure for mining high utility itemsets. Expert Systems with Applications 38, 7419–7424 (2011)
17. Lin, C.W., Hong, T.P.: A new mining approach for uncertain databases using cufp trees. Expert Systems with Applications 39, 4084–4093 (2012)
18. Lin, C.W., Lan, G.C., Hong, T.P.: An incremental mining algorithm for high utility itemsets. Expert Systems with Applications 39, 7173–7180 (2012)
19. Lin, C.W., Hong, T.P., Lee, H.Y., Wang, S.L.: Maintenance of pre-large FUSP trees in dynamic databases. In: International Conference on Innovations in Bio-inspired Computing and Applications, pp. 199–202 (2011)
20. 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)
21. Srikant, R., Agrawal, R.: Mining sequential patterns: Generalizations and performance improvements. In: The International Conference on Extending Database Technology: Advances in Database Technology, pp. 3–17 (1996)
22. Wang, C.Y., Hong, T.P., Tseng, S.S.: Maintenance of sequential patterns for record modification using pre-large sequences. In: IEEE International Conference on Data Mining, pp. 693–696 (2002)