WCCI 2012 IEEE World Congress on Computational Intelligence
June, 10-15, 2012 - Brisbane, Australia FUZZ IEEE

Incremental Multiple Fuzzy Frequent Pattern Tree

Tzung-Pei Hong[1,2], Chun-Wei Lin[1*]
[1]Dept. of Computer Science and Information Engineering
National University of Kaohsiung
Kaohsiung, Taiwan
[2]Dept. of Computer Science and Engineering
National Sun Yat-sen University
Kaohsiung, Taiwan
tphong@nuk.edu.tw, jerrylin@ieee.org

Tsung-Ching Lin[1], and Shyue-Liang Wang[3]
[1]Dept. of Computer Science and Information Engineering
National University of Kaohsiung
Kaohsiung, Taiwan
[3]Department of Information Management
National University of Kaohsiung
Kaohsiung, Taiwan
m0985507@mail.nuk.edu.tw, slwang@nuk.edu.tw

***Abstract*—In the past, the multiple fuzzy frequent pattern tree (MFFP tree) was proposed for extracting multiple fuzzy frequent itemsets from quantitative transactions. It kept the multiple transformed fuzzy regions of an item to form the multiple fuzzy frequent itemsets. In this paper, an incremental algorithm is proposed for efficiently mining multiple fuzzy frequent itemsets based on the FUP concepts and the MFFP-tree structure. Experimental results show that the proposed incremental algorithm runs faster than the batch one.**

Keywords-fuzzy set; incremetnal mining; dynamic database; transaction insertion; data mining

I. INTRODUCTION

Data mining techniques have been recently developed to derive useful knowledge from datasets [1-3, 12, 16]. Among them, mining association rules from a transaction database is especially commonly seen in data-mining research [4, 20]. In the past, most of the mining approaches for association rules were based on the Apriori algorithm [1, 3], which generated and tested candidate itemsets level by level. Han *et al.* proposed the Frequent-Pattern-tree (FP-tree) structure for efficiently mining association rules without generation of candidate itemsets [8]. Cheung *et al.* [7] proposed the Fast UPdated (FUP) algorithm to adopt the pruning techniques used in the DHP (Direct Hashing and Pruning) algorithm [20]. The FUP determined whether re-scanning the original database was needed, thus reducing the computational cost in maintaining the association rules.

In these years, the fuzzy-set theory [24] has been used more and more frequently in intelligent systems because of its simplicity and similarity to human reasoning [13]. Hong *et al.* proposed a fuzzy mining algorithm for managing quantitative data [9-10], which was based on the Apriori algorithm. Papadimitriou *et al.* proposed an approach based on FP-trees to find fuzzy association rules [19]. Lin *et al.* proposed the fuzzy FP tree to efficiently derive fuzzy frequent itemsets from a quantitative dataset [17].

The above approaches are executed in a batch way to generate the fuzzy association rules. When new transactions are inserted into the original database, these batch approaches must re-process the entire updated database to form the updated rules. Thus, the already derived knowledge becomes useless in knowledge maintenance.

In this paper, an incremental multiple fuzzy frequent-pattern tree (abbreviated as *incMFFP-tree*) algorithm for handling newly inserted transactions based on the FUP concepts [7] is thus proposed to maintain the MFFP-tree structure [11]. The initial MFFP tree is firstly constructed from the original quantitative database. Different from the traditional FP-tree construction, the transformed fuzzy regions of each item are used to construct the MFFP tree as long as they are frequent. When new transactions are inserted into the database, the proposed incremental algorithm is then executed to update the tree structure correspondingly. The *MFFP-growth* algorithm is also used to efficiently derive the multiple fuzzy frequent itemsets from the updated tree structure [11]. Experimental results show that the proposed approach has a better performance than the batch one.

II. REVIEW OF RELATED WORKS

In this section, mining fuzzy association rules and the FUP concepts are shortly reviewed.

A. *Mining Fuzzy Association Rules*

Chan *et al.* proposed an F-APACS algorithm to mine fuzzy association rules [5]. Kuok *et al.* proposed a fuzzy mining approach to handle numerical data in databases and to derive fuzzy association rules [15]. Hong *et al.* proposed a fuzzy mining algorithm to mine fuzzy rules from quantitative transaction data [9-10]. Papadimitriou *et al.* proposed an approach based on FP trees to find fuzzy association rules [19]. Lin *et al.* then proposed the fuzzy FP tree to derive the fuzzy frequent itemsets from a quantitative dataset [17]. It used the one with the maximum cardinality among the transformed fuzzy regions to represent a frequent item for constructing a fuzzy FP tree. Hong *et al.* then proposed the multiple fuzzy frequent pattern tree (MFFP-tree) algorithm and designed MFFP-tree structure to keep all the fuzzy frequent regions derived from a database no matter whether they are generated from the same items or not [11]. The corresponding *MFFP-growth* mining algorithm is also designed to derive the fuzzy frequent itemsets from the tree structure by fuzzy operations.

*Corresponding author

Here, an example is used to construct the MFFP-tree structure. Assume Table I shows an original quantitative database to be used in the example for demonstrating the construction process of MFFP tree [11]. It consists of 6 transactions and 5 items, denoted *A* to *E*. The numbers represent the quantities of items.

TABLE I. A QUANTITATIVE DATABASE

TID	***Items***
1	(*A*:5}, (*C*:10), (*D*:2), (*E*:9)
2	(*A*:8}, (*B*:2), (*C*:3)
3	(*B*:3), (*C*:9)
4	(*A*:7}, (*C*:9), (*D*:3)
5	(*A*:5), (*B*:2), (*C*:5)
6	(*A*:3), (*C*:10), (*D*:2), (*E*:2)

The fuzzy membership functions are the same for all the items for simplicity shown in Figure 1. In this example, amounts are represented by three fuzzy regions: *Low*, *Middle* and *High*. Thus, three fuzzy membership values are produced for each item in a transaction according to the predefined membership functions. The minimum support threshold is then set at 30%.

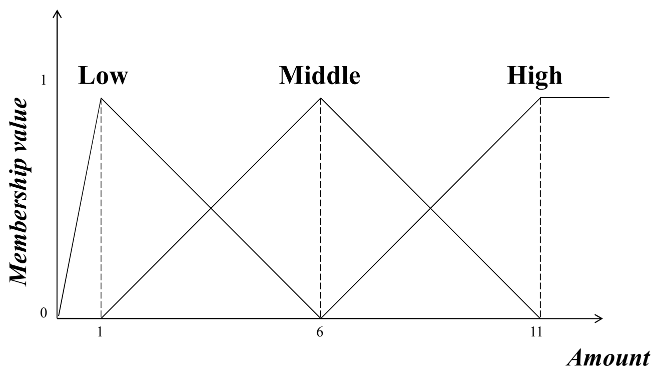

Figure 1. The membership functions used in this example.

In this example, all the frequent fuzzy regions transformed from Table I are used to construct the initial MFFP tree [11]. The constructed MFFP tree is then shown in Figure 2. Thus, the multiple fuzzy frequent patterns can be extracted by the *MFFP-growth* algorithm [11]. The other methods for mining fuzzy association rules have been stated in [14, 21-23].

B. FUP Concepts

In real-world applications, the derived association rules may be changed in a dynamic database. Conventional batch-mining algorithms solve this problem by re-processing the entire updated databases, thus requiring lots of computational time and waste existing mined knowledge. Considering an original database and some new inserted transactions, the following four cases (illustrated in Figure 3) may arise [7].

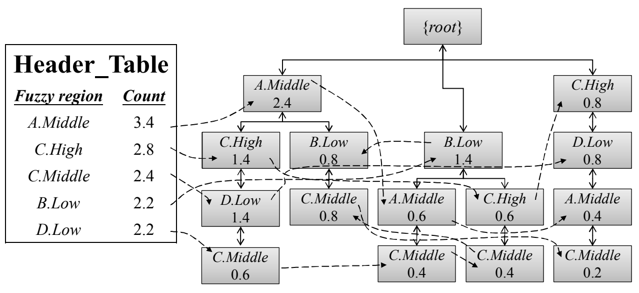

Figure 2. The constructed MFFP tree.

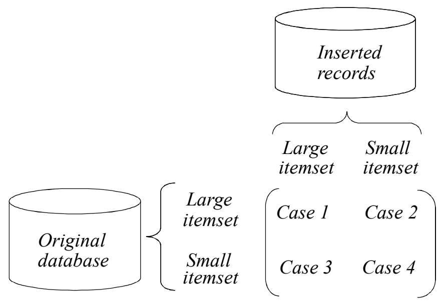

Figure 3. Four cases arising from inserted transactions to an existing database.

Case 1: An itemset is frequent both in an original database and in newly inserted transactions.
Case 2: An itemset is frequent in an original database but not frequent in newly inserted transactions.
Case 3: An itemset is not frequent in an original database but frequent in newly inserted transactions.
Case 4: An itemset is not frequent both in an original database and in newly inserted transactions.

In FUP concepts, **Cases 1** and **4** will not affect the final large itemsets. **Case 2** and **Case 3** may, however, remove existing large itemsets and add new large itemsets, respectively.

III. THE PROPOSED INCMFFP-TREE ALGORITHM

The proposed incremental MFFP-tree (*incMFFP-tree*) algorithm is stated as follows.

***The incMFFP-tree algorithm*:**

INPUT: An original quantitative database with *d* transformed transactions, an already built MFFP tree with its corresponding Header_Table, a set of membership functions, a set of *t* inserted transactions, and a predefined minimum support threshold *s*.

OUTPUT: An updated MFFP tree.

STEP 1: Transform the quantitative value v_i of each item *I* in the new transactions into a fuzzy region f_{ij} represented as ($f_{i1}/R_1 + f_{i2}/R_2 + \ldots + f_{ij}/R_j$) using the given membership functions, where *h* is the number of fuzzy regions of each item *I*, R_j is the *j*-th fuzzy region of each item *I*, $1 \leq j \leq h$, and f_{ij} is v_i's fuzzy membership value in region R_j.

STEP 2: Calculate the scalar cardinality of each fuzzy region R_j in the new transactions as:

$$S^T(R_j) = \sum_{i=1}^{t} f_{ij}\ .$$

STEP 3: Divide the fuzzy regions R_j in the new transactions into two parts according to whether they are large (appearing in the Header_Table), or small in the original database.

STEP 4: For each fuzzy region R_j which is large both in the original database (appearing in the Header_Table) and in the new transactions, do the following substeps (**Case 1**):

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

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

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

Substep 4-2: Update the count of R_j in the Header_Table as $S^U(R_j)$.

Substep 4-3: Put R_j in the set of *Insert_Regions*, which will be further processed in STEP 9.

STEP 5: For each fuzzy region R_j which is large in the original database (appearing in the Header_Table) but small in the new transactions, do the following substeps (**Case 2**):

Substep 5-1: Set the updated count $S^U(R_j)$ in the entire updated database as:

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

Substep 5-2: If $S^U(R_j) \geq (d+t) \times s$, fuzzy region R_j will still be large after the database is updated; update the count of R_j in the Header_Table as $S^U(R_j)$ and add R_j to the set of *Insert_Regions*;

Otherwise, fuzzy region R_j will become small after the database is updated; Remove R_j from the Header_Table, connect each parent node of R_j directly to the corresponding child node of R_j, and remove R_j from the MFFP tree.

STEP 6: For each fuzzy region R_j which is small in the original database (not appearing in the Header_Table) but large in the new transactions, do the following substeps (**Case 3**):

Substep 6-1: Rescan the transformed *d* transactions in the originally quantitative database to find out the transactions with fuzzy region R_j; calculate the count $S^D(R_j)$ of R_j in the original database.

Substep 6-2: Set the updated count $S^U(R_j)$ in the entire updated database as:

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

where $S^D(R_j)$ is the count of R_j obtained from Substep 6-1 and $S^T(R_j)$ is the count of R_j in the new transactions.

Substep 6-3: If $S^U(R_j) \geq (d+t) \times s$, fuzzy region R_j will be large after the database is updated; Add fuzzy region R_j both in the set of *Insert_Regions* and in the set of *Rescan_Regions* for the further processing.

STEP 7: If the set of *Rescan_Regions* is ***null***, nothing has to be done in this step; Otherwise, do the substeps as follows:

Substep 7-1: Sort the fuzzy regions in the *Rescan_Regions* in descending order of their updated counts.

Substep 7-2: Insert the items in the *Rescan_Regions* to the end of the Header_Table according to the descending order of their counts.

STEP 8: For each original transaction with a fuzzy region *J* existing in the *Rescan_Regions*, do the substep as follows:

Substep 8-1: Remove the fuzzy regions of the items not existing in the Header_Table.

Substep 8-2: Sort the remaining frequent fuzzy regions in each transaction by their membership values in a descending order.

Substep 8-3: For each updated transaction with an fuzzy region R_j existing in the *Rescan_Regions*, if R_j has been at the corresponding branch of the MFFP tree for the transaction, add the membership value of R_j in the transaction to the node of R_j in the branch; Otherwise, add a node of R_j at its corresponding position in the branch, set the count of the node as the membership value of R_j, and insert a link from the node of R_j in the last branch to the current node. If there is not such a branch with the node of R_j, insert a link from the entry of R_j in the Header_Table to the added node.

STEP 9: For each new transaction with a fuzzy region *J* existing in the *Insert_Regions*, do the substep as follows:

Substep 9-1: Remove the fuzzy regions of the items not in Header_Table from the new transactions of the transformed database.

Substep 9-2: Sort the remaining frequent fuzzy regions in each new transaction by their membership values in a descending order.

Substep 9-3: Insert the updated new transactions into the MFFP tree tuple by tuple. If a fuzzy region R_j in a transaction has been at the corresponding branch of the MFFP tree for the transaction, add the membership value of R_j in the transaction to the node of R_j in the branch; Otherwise, add a node of R_j at the end of the corresponding branch, set the count of the node as the membership value of R_j, and insert a link from the node of R_j in the last branch to the current node. If there is not such a branch with the node of R_j, insert a link from the entry of R_j in the Header_Table to the added node.

In STEP 8, a corresponding branch is the branch generated from a transformed transaction according to the descending order of the membership values of the fuzzy regions in it.

IV. An Illustrative Example

Assume the already constructed MFFP tree from Table I was shown in Figure 2. Assume the two records shown in Table 2 are inserted into the original database shown in Table 1. The minimum count for the entire updated database is calculated as $(6 + 2)\times 0.3$ (= 2.4).

TABLE II. Two Inserted Transactions

TID	*Items*
7	(A:6), (C:11), (E:9)
8	(A:5), (D:3)

The quantitative values of the items in the newly inserted transactions in Table 2 are then transformed to represent as the fuzzy sets. After that, the scalar cardinality of each fuzzy region is calculated as the count value. The transformed fuzzy regions are then divided into two parts according to whether they are large (appearing in the Header_Table) or small in the original database. Each part is then processed by its own way.

Case 1: For each fuzzy region which is large both in the original database (appearing in the Header_Table) and in the new transactions is processed. In this example, fuzzy regions {*A.Middle*}, {*C.High*} and {*D.Low*} are then processed. After the designed algorithm has been processed, *Insert_Regions* = {*A.Middle*, *C.High*, *D.Low*}. The MFFP tree and the Header_Table are then correspondingly updated.

Case 2: For each fuzzy region in which is large in the original database (appearing in the Header_Table) but small in the new transactions is then processed. In this example, fuzzy regions {*B.Low*} and {*C.Middle*} are then processed. After the designed algorithm has been processed, *Insert_Regions* = {*A.Middle*, *C.High*, *C.Middle*, *D.Low*}. The MFFP tree and the Header_Table are then correspondingly updated.

Case 3: For each fuzzy region which is small in the original database (not appearing in the Header_Table) but large in the new transactions is then processed. In this example, only fuzzy region {*E.High*} satisfied the condition and then processed. The originally transformed database is then re-scanned to find the count of {*E.High*}. The count of {*E.High*} is then updated to determine whether its count is larger than or equal to the updated minimum count. After the algorithm has been processed, the set of *Rescan_Regions* is ***null*** in this example, and nothing has to be done in this step.

The MFFP tree is updated according to the new transactions with fuzzy regions existing in the set of *Insert_Regions*. The corresponding branches for the new transactions with any these fuzzy regions are then found and sorted in descending order according their fuzzy values in the transaction. The MFFP tree and its Header_Table are then correspondingly updated. After that, the final updated MFFP tree is thus shown in Figure 4.

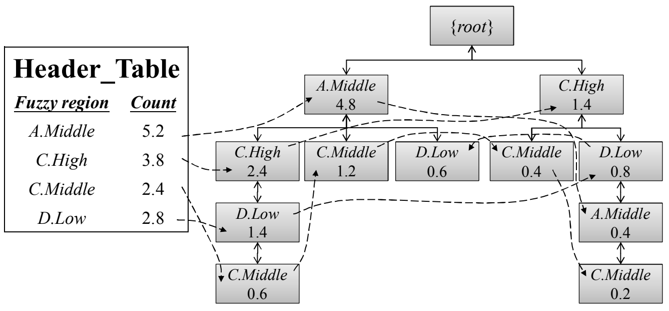

Figure 4. The final updated MFFP tree.

The desired multiple fuzzy frequent patterns in Figure 4 can thus be extracted by the *MFFP-growth* [11].

V. Experimental Results

A real dataset called FOODMART from an anonymous chain store was used in the experiments [18]. It contains quantitative transactions about the products sold in the chain store. There were totally 21,556 transactions with 1,600 items in the dataset.

In the experiments, three fuzzy regions (*High* and *Middle* and *Low*) for items were transformed. The first 19,556 transactions were extracted from FOODMART dataset to construct the initial MFFP tree. The next 400 transactions were then sequentially used each time as new transactions for the proposed *incMFFP-tree* algorithm. Experiments were then made to evaluate the efficiency of the proposed mining algorithm at different minimum support thresholds, with the results shown in Figure 5.

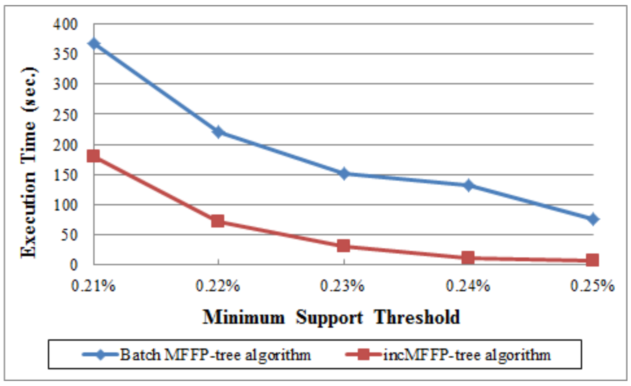

Figure 5. The execution times for two algorithms in the different minium support thresholds.

From Figure 5, it could be seen that the proposed *incMFFP-tree* algorithm has a better performance than the batch one in execution time, including the construction and mining phases.

VI. Conclusions

In this paper, an incremental multiple fuzzy frequent pattern tree (*incMFFP-tree*) algorithm is proposed for efficiently handling transaction insertion in dynamic fuzzy data mining. The proposed approach uses the multiple transformed fuzzy regions of an item to initially construct the MFFP tree, which is more realistic in real-world applications. When the new transactions are inserted into the original database, the proposed algorithm will divide the fuzzy frequent regions into four parts according to whether they are large or small in the original database and in the new transactions. Experimental results show that the proposed incremental algorithm has a better performance than the batch one for handling new transactions.

References

[1] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules in large databases," *The International Conference on Very Large Data Bases*, pp. 487-499, 1994.

[2] R. Agrawal and R. Srikant, "Mining sequential patterns," *The International Conference on Data Engineering*, pp. 3-14, 1995.

[3] R. Agrawal, T. Imielinski, and A. Swami, "Mining association rules between sets of items in large databases," *The International Conference on Management of Data*, pp. 207-216, 1993.

[4] R. Agrawal, T. Imielinski, and A. Swami, "Database mining: A performance perspective," *IEEE Transactions on Knowledge and Data Engineering,* vol. 5, pp. 914-925, 1993.

[5] K. C. C. Chan and W. H. Au, "Mining fuzzy association rules," *The International Conference on Information and Knowledge Management*, pp. 209-215, 1997.

[6] M. Delgado, M. D. Ruiz, D. Sánchez, and J. M. Serrano," A formal model for mining fuzzy rules using the RL representation theory," *Information Sciences*, vol. 181, Issue 23(1), pp. 5194-5213, 2011.

[7] D. W. Cheung, H. Jiawei, V. T. Ng, and C. Y. Wong, "Maintenance of discovered association rules in large databases: An incremental updating technique," *The International Conference on Data Engineering*, pp. 106-114, 1996.

[8] J. Han, J. Pei, Y. Yin, and R. Mao, "Mining frequent patterns without candidate generation: A frequent-pattern tree approach," *Data Mining and Knowledge Discovery,* vol. 8, pp. 53-87, 2004.

[9] T. P. Hong and J. B. Chen, "Finding relevant attributes and membership functions," *Fuzzy Sets and Systems,* vol. 103, pp. 389-404, 1999.

[10] T. P. Hong, C. S. Kuo, and S. L. Wang, "A fuzzy aprioritid mining algorithm with reduced computational time," *Applied Soft Computing,* vol. 5, pp. 1-10, 2004.

[11] T. P. Hong, C. W. Lin, and T. C. Lin, "Mining complete fuzzy frequent itemsets by tree structures," *IEEE International Conference on Systems Man and Cybernetics*, pp. 563-567, 2010.

[12] K. Hu, Y. Lu, L. Zhou, and C. Shi, "Integrating classification and association rule mining: A concept lattice framework," *The International Workshop on New Directions in Rough Sets, Data Mining, and Granular-Soft Computing*, pp. 443-447, 1999.

[13] A. Kandel, *Fuzzy expert systems*, 1992.

[14] M. Kaya and R. Alhajj, "Mining multi-cross-level fuzzy weighted association rules," *IEEE International Conference Intelligent Systems*, pp. 225-230, 2004.

[15] C. M. Kuok, A. Fu, and M. H. Wong, "Mining fuzzy association rules in databases," *SIGMOD Record,* vol. 27, pp. 41-46, 1998.

[16] B. Lent, A. Swami, and J. Widom, "Clustering association rules," *The International Conference on Data Engineering*, pp. 220-231, 1997.

[17] C. W. Lin, T. P. Hong, and W. H. Lu, "Linguistic data mining with fuzzy fp-trees," *Expert Systems with Applications,* vol. 37, pp. 4560-4567, 2010.

[18] Microsoft, *Example database foodmart of microsoft analysis services*, http://msdn.microsoft.com/en-us/library/aa217032(SQL.80).aspx.

[19] S. Papadimitriou and S. Mavroudi, "The fuzzy frequent pattern tree," *The WSEAS International Conference on Computers*, pp. 1-7, 2005.

[20] J. S. Park, M.-S. Chen, and P. S. Yu, "Using a hash-based method with transaction trimming for mining association rules," *IEEE Transactions on Knowledge and Data Engineering,* vol. 9, pp. 813-825, 1997.

[21] W. Shitong, K. F. L. Chung, and S. Hongbin, "Fuzzy taxonomy, quantitative database and mining generalized association rules," *Intelligent Data Analysis,* vol. 9, pp. 207-217, 2005.

[22] R. Srikant and R. Agrawal, "Mining quantitative association rules in large relational tables," *SIGMOD Record,* vol. 25, pp. 1-12, 1996.

[23] J. S. Yue, E. Tsang, D. Yeung, and D. Shi, "Mining fuzzy association rules with weighted items," *IEEE International Conference on Systems, Man, and Cybernetics*, pp. 1906-1911 vol.3, 2000.

[24] L. A. Zadeh, "Fuzzy sets," *Information and Control,* pp. 338-353, 1965.