Finding Multiple-Level Association Rules from a Transactional Database Using FP-Tree

Abstract

In many database applications, information stored in a database has a built-in hierarchy consisting of multiple levels of concepts. A concept hierarchy, for example, may reflect a categorization of items sold in a department store, such as electronics, computers, desktop computers, etc. In such a database, users may want to find out association rules among items only at the same level or association rules that span over different levels. This task is called *multiple-level association rule mining*. An Apriori-based algorithm was proposed to do this task [3,4] and, therefore, suffers the same disadvantages as the Apriori algorithm does. Namely, it has to scan the database many times as determined by the length of the longest frequent itemset. In this paper we introduce a new algorithm, called *MFP-tree*, for mining multiple-level association rules from a transactional database, which is based on the FP-tree algorithm. The proposed algorithm requires only two scans of the database and our performance study shows that the MFP-tree algorithm outperforms the Apriori-based algorithm for mining multiple-level association rules.

Key Words.

1. Introduction

The association rule mining task is to find association relationships among a set of data items in a database. Let $I = \{i_1, i_2, \ldots, i_m\}$ be a set of m distinct items. A transactional database $D = \{T_1, T_2, \ldots, T_n\}$, where $T_i \subseteq I$. Typically, each transaction T_i has a unique identifier *TID* and is represented as $< TID, i_1, i_2, \ldots, i_k >$. An itemset (or a set of items) with k items is called a *k-itemset*. The support of an itemset A, $sup(A)$, is defined as the fraction of the transactions in D that contain A. An association rule is an expression $A \Rightarrow B$, where itemsets A, $B \subset I$ and $A \cap B = \phi$. Confidence of the rule $R : A \Rightarrow B$, which is defined as $conf(R) = sup(A \cup B)/sup(A)$, is the conditional probability that a transaction contains B, given that it contains A. The problem of association rule mining can be decomposed into two sub-problems. The first one is finding all frequent itemsets (which is also called *frequent patterns*) that satisfy a specified minimum support, and the second one is forming implication rules among the frequent itemsets. The entire performance is mainly determined by the first sub-problem, which generates all frequent itemsets. Various algorithms have been proposed to discover frequent itemsets [1, 2, 5, 6, 8, 9]. Among these algorithms, *Apriori* [2] was the first efficient algorithm to mine association rules. This algorithm consists of multiple steps and each step mines frequent patterns of a certain length. Apriori requires as many database scans as the length of the longest frequent itemsets. The FP-tree algorithm [5] avoids this problem by using a data structure called *Frequent Pattern tree* (*FP-tree*). FP-tree algorithm needs only two database scans to discover the complete set of frequent itemsets, and was reported to be an order of magnitude faster than the Apriori algorithm.

In many applications, transactional databases contain data with a hierarchical structure, which is called *concept hierarchy*. Figure 1 shows an example of such a hierarchy. In such applications users may be interested in association rules among items only at the same level or association rules that span over different levels since more interesting rules can be derived by taking the hierarchy into account [3, 4].

The Apriori-based algorithm adopts a progressively deepening method. First it examines large itemsets at high levels and then steps down to examine their descendants at lower levels. With a slight modification this algorithm can also mine cross-level rules. Since this algorithm is based on Apriori, the database must be scanned multiple times to find frequent itemsets at each level. In this paper, we propose a new algorithm to discover multiple-level association rules without candidate generation. The proposed algorithm requires only two scans of database and therefore is much faster than the Apriori–based algorithm.

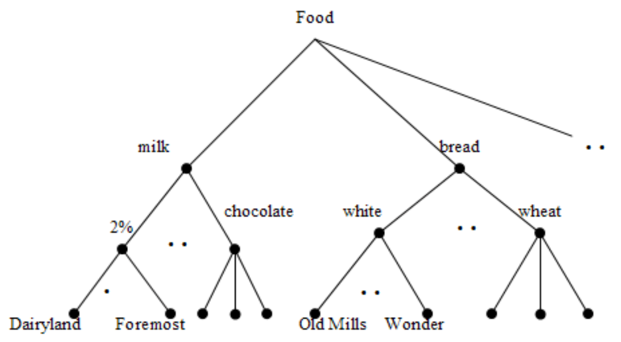

Figure 1. An example of hierarchy information stored in a transactional database.

The rest of paper is organized as follows. Section 2 briefly reviews the FP-tree algorithm. Section 3 describes the problem definition of multiple-level association rule mining. Section 4 discusses the Apriori-based multiple-level association rule mining algorithm. Section 5 describes, in detail, the proposed multiple-level association rule mining algorithm, which is based on FP-tree. Section 6 discusses the extension of the proposed algorithm to mine association rules that involve multiple concept levels. Section 7 presents the experiments we performed and their results. Section 8 concludes the paper.

2. FP-tree algorithm for mining single-level association rules

To avoid multiple scans of a database, the FP-tree algorithm was proposed in [5]. Unlike the *Apriori* algorithm, FP-tree does not generate candidate itemsets. It first represents the database in a compact data structure called FP-tree. Then, frequent itemsets are mined directly from the tree. Once all frequent itemsets are found, we can mine rules in the same way as we do with *Apriori*. With Apriori, the database must be scanned as many times as required by the length of the longest frequent itemset. FP-tree, however, requires only two scans of the database. It was shown in [5] that FP-tree is an order of magnitude faster than Apriori. The high efficiency of FP-growth is achieved in three aspects:

1. It utilizes a compact data structure to store relevant database information to avoid multiple database scans.
2. It adopts a pattern fragmentation growth mining method to avoid costly candidate generation.
3. A partition-based, divide and conquer method is used as a search technique. This dramatically reduces the size of conditional pattern bases generated at the subsequent levels as well as the size of corresponding conditional FP-trees.

3. Multiple-level association rule mining

The problem of mining multiple-level association rules, first introduced in [3] and [8], is an extension of Apriori. Let I, T, D, support of an itemset, and confidence of an association rule be the same as defined in Section 1. An association rule is an expression $A \Rightarrow B$, where itemsets $A, B \subset I$, $A \cap B = \phi$, and A and B are at the same level of a concept hierarchy. To find frequent itemsets and strong association rules, a user or an expert defines two thresholds; minimum support σ and minimum confidence φ. Different minimum supports and minimum confidences are typically used at different concept levels. An itemset A is large (frequent) at level j if the support of A is no less than the minimum support threshold at level j, σ_j, and each ancestor (if any) of every item in A is large at its corresponding level. The confidence of a rule $A \Rightarrow B$ is high at level j if its confidence is no less than its minimum confidence threshold at level j, φ_j . A rule $A \Rightarrow B$ is strong at level j if $A \cap B$ is large at level j and the confidence of $A \Rightarrow B$ is high at level j. The multiple-level association rule mining problem is to discover all strong association rules at each concept level. In the following section, we briefly review the Apriori-based algorithm for mining multiple- level association rules proposed in [3]

4. Apriori-based algorithm for multiple-level association rule mining

The Apriori-based algorithm adopts a progressively deepening method. It is based on the observation: one may often be interested only in those items and their association relationships if their corresponding high level ancestors have reasonably large supports. The progressive deepening algorithm first mines large itemsets at high levels and then steps down to examine their descendants at lower levels. The mining at each level adopts the Apriori algorithm. The Apriori-based multi-level association rule mining algorithm discovers association rules among the items at the same level of a hierarchy. This restriction can be relaxed to allow exploration of rules that involve items from different levels. Such a rule is called a *cross-level* rule. An example of such a rule is "2% Dairyland Milk $\Rightarrow$ White Bread", where the former is at level 3 and the latter is at level 2. A modified Apriori-based algorithm was proposed in [3, 4] to achieve this task. When mining frequent 2-itemsets at level *j*, we consider all 1-itemsets *at all levels i*, $1 \leq i < j$ (note that in the previous level-by-level rule mining algorithm, only those at the same level were considered). The Apriori-based multiple-level association rule mining algorithm suffers the same drawback as the original Apriori algorithm. At each level, it has to scan the database once for each *k* (the length of an itemset). This observation led us to develop an algorithm that is based on FP-tree, which is presented in the following section.

5. FP-tree-based multiple-level association rule mining algorithm (MFP-tree)

In this section we present a detailed description of our FP-tree-based algorithm for mining multi-level association rules. Our algorithm uses a data structure, called *MFP-tree*, similar to the one used by the FP-tree algorithm for mining single-level association rules.

5.1 Constructing MFP-Tree

Our algorithm utilizes the FP-tree data structure described in [5] with the following modifications. In what follows *l* denotes the number of levels in the concept hierarchy. For the concept hierarchy of Figure 1, *l* is 3.

- There are *l* header tables (one header table for each level). An item is included in the header table *j* if its support satisfies the minimum support constraint for level *j* and all of its ancestors are frequent. Each entry in a header table consists of two fields, (1) *Name* and (2) *Head* of node link. For the header table *j*, *name* represents the item name with respect to level *j*. *Head* of node link points to the first node in the MFP-tree carrying the same item *name* with respect to level *j*.
- The node-link field in each node in the tree is extended to have *l* node-links. The *j-th* link points to the next node in the MFP-tree carrying the same item name with respect to level *j* or null if there is none.

The algorithm for constructing MFP-tree is given in Figure 5. Example 2 demonstrates this algorithm.

Example 2: Let T[1] be the encoded (in accordance with the concept hierarchy of Figure 1) transaction table where taxonomy information for each item in the transactional database is encoded as a sequence of digits as shown in Table 1. For example the item *2% foremost milk* is encoded as '112', where the first digit '1' represents *milk* at level 1, the second '1' for *2% milk* at level 2, and the third '2' for the brand *Foremost* at level 3. *2% milk*, which is at level 2, is encoded as 11*, and *milk* at level 1 is encoded as 1**. Let minimum support thresholds (*minsup*) be 4, 3, 3 for level 1,level 2, and level 3, respectively. The frequency lists are:

- OL_1: {(1**:5), (2**:5)}
- OL_2: {(11*:5), (12*:4), (21*:4), (22*:4)}
- OL_3: {(111:4), (211:4), (221:3)}
- ML= {111, 112, 121, 122, 211, 221, 222}

Table 1. Encoded transaction table T[1].

Transaction ID	Items
T1	{111,121,211,221}
T2	{111,211,222,323}
T3	{112,122,221,411}
T4	{111,121}
T5	{111,122,211,221,413}
T6	{211,323,524}
T7	{323,411,524,713}

Three header tables are created with OL_1, OL_2, and OL_3, one for each level as shown in Figure 6. The scan of the first transaction in Table 1 leads to the construction of the first branch in the tree <(111:1), (121:1), (211:1), (221:1)>, as shown in Figure 6. Notice that the transaction is ordered according to ML. The heads of node links for items (1**, 2**), (11*, 12*, 21*, 22*), (111, 121, 211, 221) in the header tables for level 1, level 2, and level 3, respectively, are setup to point to the first occurrence of the corresponding item. For example, 1** 's head of node link will point to 111 while the 2**'s head of node link will point to 211. The same process is repeated for all the transactions and Figure 7 shows the final MFP-tree after processing all transactions.

5.2 Multi-level frequent pattern generation

To generate the frequent patterns at each level, we propose an algorithm called MFP-growth, which is an extension of the FP-growth algorithm. The input is the MFP-tree constructed above and predefined minimum supports. MFP-growth is executed in the same way as described in Section 5.1 with two possible different cases:

- Case 1: When collecting the conditional pattern base (CPB) for item α at level *j*, an item may appear more than once in α's prefix path. In this case only one item is kept. For example, consider the initial CPB for item 2** at level 1: CPB = {<(1**:1), (1**:1)>, <(2**:1), (1**:1)>, <(1**:1)>, <(1**:1)>}. After deleting repeated occurrences of items from each prefix path the CPB becomes: CPB = {<1**:1>, <1**:1>, <1**:1>, <1**:1>}.
- Case 2: The CPB for a given item at level *j* may contain some items which are infrequent with respect to level *j*. In this case, any infrequent item should be ignored. For example, the initial CPB for item 211 at level 3 is {<(121:1), (111:1)>, <(111:1)>, <(122:1), (111:1)>}. Items 121 and 122 are infrequent at level 3, and hence should be ignored. The resulting CPB for item 211 is {<111:1>, <111:1>, <111:1>}. MFP-growth algorithm is given in Figure 8.

```
Algorithm 2. MFP-growth: mining frequent patterns with MFP-tree by pattern fragment growth
Input: MFP-tree constructed based on algorithm 1 using TD and a minimum support threshold minsup[j] for concept level j.
Output: the complete set of frequent itemsets at level j.
Method: call FP-growth (Tree, null)
1.  Procedure FP-growth(Tree, $\alpha$)
2.  If  Tree contains a single path P with respect to level j
3.      Then For all combination $\beta$ of the nodes at level j in the path P do
4.          Generate itemset $\beta \cup \alpha$ with support = minimum support of nodes in $\beta$
5.      End for
6.  Else
7.      For all $a_i$ in header table of Tree at level j do
8.          Generate itemset $\beta = a_i \cup \alpha$ with support = $a_i$. support
9.          Construct $\beta$'s conditional pattern base by considering only nodes with names in the header table at
10.             level j and only one occurrence  per node with the same name, and then construct $\beta$'s conditional FP- tree Tree $_\beta$ .
11.         If   $Tree_\beta \neq \Phi$
12.             Then call FP-growth(Tree,  $\beta$)
13.         End if
14.     End For
15. End if
```

Figure 8. Algorithm 2, MFP-growth.

From the MFP-tree construction process, it is obvious that it needs only two database scans where the first scan collects the frequent 1-itemsets at each concept level, and the second constructs the MFP-tree. We can also observe that each transaction in TD is mapped into one and only one path in the MFP-tree, and all the information in the tree is from the database. So, the MFP-tree is complete and does not contain any redundancy. Therefore, the frequent itemsets that are mined from the MFP-tree are identical to those that would be mined from the original database. The height of the MFP-tree is bounded by the maximum number of frequent items in transactions. The size of MFP-tree is bounded by the size of TD since each transaction will contribute at most one path to the MFP-tree.

```
Algorithm 1. Construction of MFP-tree
Input: (1)TD, a hierarchy-information-encoded and task relevant set of transaction database, in format of <TID,Itemset> and (2) the
minimum support threshold (minsup[j] ) for each concept level j, , $1\leq j\leq l$ , and $l$ the is number of concept levels.
Output: its MFP-tree
1.  Scan the TD once, generate a list ($L_j$) of 1-iemsets and their counts at each concept level j.
2.  Generate an ordered frequency list ($OL_j$) for each concept level j, by filtering out infrequent items( items who do not pass the
    minimum support threshold
3.    at level j and/or ancestors(if any) are infrequent ) in $L_j$ , and sorting it in frequency descending order. These ordered lists are
    used to build header
4.    tables.
5.  Generate an ordered frequent list ML from $L_l$ by filtering out items whose ancestors at the highest level (level 1) are infrequent
    and sorting it in
6.    descending order according to $OL_1$ as the first sorting key, $OL_2$ as the 2nd sorting key,…., and $OL_l$ as the $l$th sorting key*.
7.  Create the root of the FP-tree T with label "Null"
8.  For each transaction trans in TD do the following.
9.    Select and sort frequent items in trans according to ML.
10.   Let the sorted item list in trans be [p/P], where p is the first element and P is the remaining list.
11.   Call Insert_tree ([p/P],T).
12. End for
13.
14. Function Insert_tree([p/P],T)
15. If T has a child N such that N.itemName =p.itemName
16.   Then N.count=N.count+1
17. Else
18.   Create new node N with count=1, parent linked to T
19.     For each concept level j
20.       If there is no node x in the path from the root reaching node N, such that x.itemName= N.itemName with respect to level
    j
21.         Then Add N to the chain of nodes with same name (with respect to level j) via the node link structure for level j.
22.       End if
23.     End For
24. End if
25. If $P \neq \Phi$
26.   Then Insert_tree (P,N)
27. End if
```

Figure 5. Algorithm 1, MFP-tree construction.

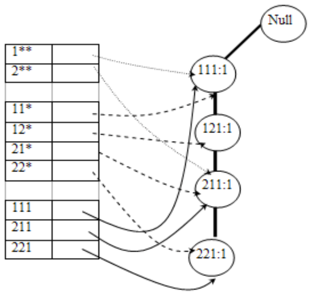

Figure 6. MFP-tree after inserting the first transaction.

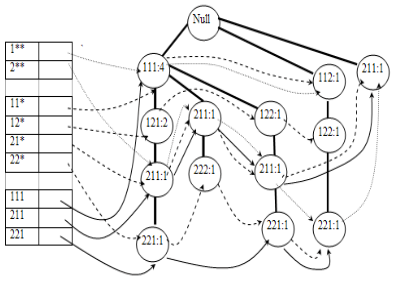

Figure 7. MFP-tree after processing all transactions.

6. Mining cross-level association rules

The MFP-tree-based algorithm, which was described above, generates association rules among items at the same levels. Cross-level association rules are not confined to those at the same level and may include items across different levels. To do this, we first need to mine frequent patterns that may include items from different levels (e.g. {111, 2**}). In this section, we describe an algorithm to mine frequent patterns that may include items from different levels by extending the previously discussed MFP-growth algorithm. The construction of the MFP-tree is the same as described in Algorithm 1, but MFP-growth is extended as follows. To mine frequent patterns at level *j*, where $1 \leq j \leq l$, for each item α in the header table *j*, α's conditional pattern base and its conditional FP-tree are built for *all levels i*, $1 \leq i \leq l$. When constructing a conditional pattern base, all ancestors of α are excluded (because, for example, we are not interested in finding associations between *milk* and *2% milk*). Then, frequent patterns are mined, for each level, from the FP-trees using the same MFP-growth algorithm of Figure 8. Example 3 illustrates how this process works.

Example 3: For the same encoded transaction of Table 1 with the same minimum supports, we show the mining of cross-level frequent patterns that include items at different levels.

Mining at level 1

- Item 2**
  - For level 1: Item 2**'s conditional pattern base is {<1**:1>, <1**:1>, <1**:1>, <1**:1>}. The conditional FP-tree is built from this conditional pattern base. Then, frequent patterns are mined from the conditional FP-tree at level 1. The result has only one frequent pattern, that is {2**, 1**:4}.
  - For level 2: Item 2**'s conditional pattern base is {<12*:1, 11*:1>, <11*:1>, <12*:1, 11*:1>, <12*:1, 11*:1>}. The frequent patterns at level 2 are {2**, 11*:4}, {2**, 12*:3}, and {2**, 11*, 12* :3}.
  - For level 3: Item 2**'s conditional pattern base is {<111:1>, <111:1>, <111:1>}. Note that 121 is not considered because it is not frequent at level 3. The only frequent pattern at this level is {2**, 111:3}.
- Item 1**'s conditional pattern bases at level 1, 2, and 3 are empty.

Mining at level 2

- Item 22*
  - For level 1: Item 22*'s conditional pattern base is {<1**:1>, <1**:1>, <1** :1>, <1**:1>}. The frequent pattern at level 1 is {22*, 1**:4}.
  - For level 2: Item 22*'s conditional pattern base is {<21*:1, 12*:1, 11*:1>, <21*:1, 11*: 1>,<21*: 1, 12*:1, 11*:1>, <12*:1, 11*:1>} . The frequent patterns at level 2 are {22*, 11**:4}, {22*, 21* :3}, {22*, 12* :3}, {22*, 11*, 21* :3}, and {22*, 11*, 12* :3}.
  - For level 3: Item 22*'s conditional pattern base is {<211:1, 111:1>, <211:1, 111:1>, <211:1, 111:1>}. The frequent patterns at level 3 are {22*, 111:3}, {22*, 211:3}, and {22*, 111, 211 :3}.,

Conditional pattern bases and conditional FP-trees for the rest of items in header table 2 are constructed in the same way. Mining at level 3 proceeds in the same manner too.

7. Experiment

7.1 Datasets and parameters

To study the performance of our proposed algorithm, we implemented both FP-tree-based algorithm and the Apriori-based algorithm [3,4] using C++ and ran the programs on a Linux machine with an Intel CPU. We used a set of synthetic transactional databases generated using a randomized item set generation algorithm similar to that described in [2]. The basic parameters we used are: (1) the total number of items, $N = 1000$; (2) the total number of transactions $|D| = 100{,}000$; and (3) the number of maximal potentially large itemsets $|L| = 2000$. Table 2 shows two databases we used while varying other parameters. Here, $|I|$ is the average size of the maximal potentially large itemsets, and $|T|$ is the average size (# of items) of a transaction .

Table 2. Transaction databases.

Database	$\|I\|$	$\|T\|$
DB1	4	10
DB2	6	20

Table 3. Encoding Parameters.

Item table	#_nodes at level 1	M2	M3	M4
I1	8	5	5	5
I2	15	6	3	4

Each transactional database is converted into an encoded transaction table according to the concept hierarchy. The total number of levels of the concept hierarchy is set to 4. The fan-outs at the lower levels are selected based on a normal distribution with the mean value of M2, M3, and M4 for the levels 2, 3, and 4, respectively, and a variance of 2.0. These parameters are summarized in Table 3.

There are four encoded transaction tables used in our experiments. Table 4 shows these encoded transaction tables.

Table 4. Encoded transaction tables.

Encoded transaction table	Database	Item table
MDB11	DB1	I1
MDB12	DB1	I2
MDB21	DB2	I1
MDB22	DB2	I2

Table 5. Minimum support thresholds.

Threshold	Minimum support thresholds
1	[60,40,10,5]
2	[40,20,10,5]
3	[20,10,5,2]
4	[10,8,4,1]

For each of these four databases, four different sets of minimum support thresholds are used as shown in Table 5. The minimum support threshold of [60, 40, 10, 5] denotes that the minimum support of level 1 is 60% and those of levels 2, 3, and 4 are 40%, 10%, and 5%, respectively.

7.2 Result and discussion

Figures 10, 11, 12, and 13 show the execution times of the FP-tree-based algorithm and the *Apriori*–based algorithm without level crossing (level wise) on MDB11, MDB21, MDB12, and MDB22, respectively. All of the four tests show that the FP-tree-based algorithm has better performance than the *Apriori*-based algorithm. The difference in the performance increases as the minimum support threshold decreases, which means an increase in the number and length of discovered frequent itemsets.

Figures 14, 15, 16, and 17 show the comparison of the execution times of the FP-tree-based and the *Apriori*–based algorithms with level crossing on MDB11, MDB21, MDB12, and MDB22, respectively. All of the four tests again show that FP-tree-based algorithm has better performance than The *Apriori*-based algorithm. Again, as the minimum support threshold decreases, the performance difference increases.

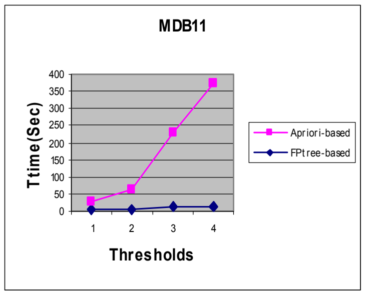

Figure 10. Execution time of FP-tree-based and *Apriori*-based algorithms on MDB11.

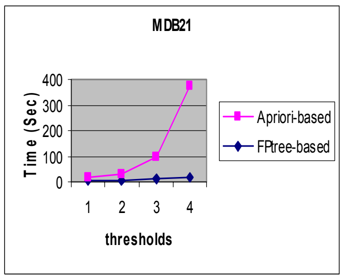

Figure 11. Execution time of FP-tree-based and *Apriori*-based algorithms on MDB21.

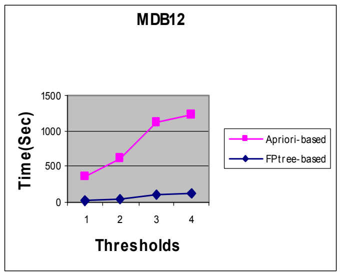

Figure 12. Execution time of FP-tree-based and *Apriori*-based algorithms on MDB12.

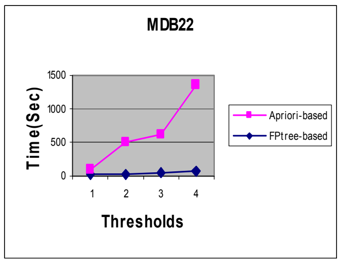

Figure 13. Execution time of FP-tree-based and *Apriori*-based algorithms on MDB22.

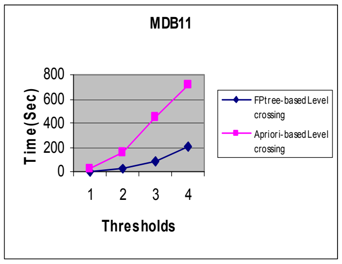

Figure 14. Execution time of FP-tree-based and *Apriori*-based algorithms on MDB11.

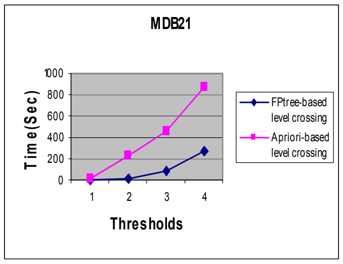

Figure 15. Execution time of FP-tree-based and *Apriori*-based algorithms on MDB21.

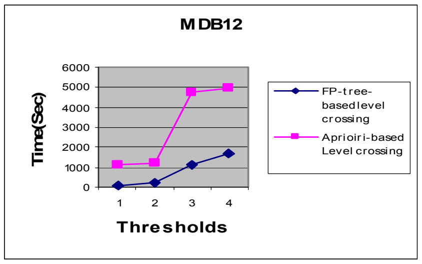

Figure 16. Execution time of FP-tree-based and *Apriori*-based algorithms on MDB12

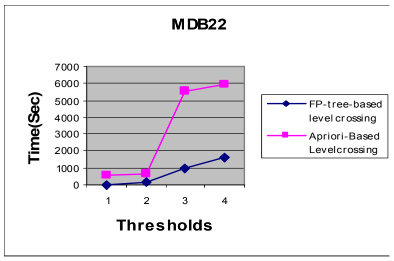

Figure 17. Execution time of FP-tree-based and *Apriori*-based algorithms on MDB22.

8. Conclusion

Transaction databases in many applications contain data that has built-in hierarchy information. In such databases, users may be interested in finding association rules among items only at the same level or association rules that span over multiple levels in the hierarchy. In this paper, we presented an efficient multiple-level association rule mining algorithm, which is based on FP-tree. Unlike the Apriori-based algorithm that requires multiple scans of a database, the proposed algorithm scans the database only twice. As a result, its execution time is much smaller than that of the Apriori-based algorithm. We conducted extensive experiments and the results confirmed our analysis. We also observed that the difference in the performance increases as the minimum support thresholds decrease. This is because the number and lengths of discovered frequent itemsets increase with the decrease of minimum support thresholds and this negatively affects the performance of the Apriori-based algorithm, while the degree by which the MFP-tree algorithm is affected by this is much smaller.

References

[1] R. Agrawal, T. Imielinski, and A. Swami, "Mining Association Rules Between Sets of Items in Large Databases," Proceedings of the ACM SIGMOD, Washington, DC, May 1993, pp. 207-216.

[2] R. Agrawal and R. Srikant, "Fast Algorithm for Mining Association Rules," Proceedings of the VLDB, Santiago de Chile, Chile, September 1994, pp. 487-499.

[3] J. Han, and Y. Fu, "Discovery of Multiple-Level Association Rules from Large Databases," Proceedings of the VLDB, Zurich, Switzerland, 1995.

[4] J. Han, and Y. Fu, "Mining Multiple-Level Association Rules in Large Databases," IEEE Transaction on Knowledge and Data Engineering, Vol. 11, No. 5, September/October 1999,pp. 798-805.

[5] J. Han, J. Pei, and Y. Yin, "Mining Frequent patterns without candidate Generation," Proceedings of the ACM SIGMOD, Dallas, TX, May 2000, pp.1-12.

[6] J. S. Park, M. S. Chen, and P. S. Yu, "An Effective Hash-Based Algorithm for Mining Association Rules," Proceedings of the ACM SIGMOD, San Joes, CA, May 1995, pp. 175-186.

[7] S. Srikant and R. Agrawal, "Mining Generalized Association Rules," Proceedings of the VLDB , Zurich, Switzerland, September 1995, pp. 407-419.

[8] A. Savasere, E. Omiecinski, and S. Navathe, "An efficient Algorithm for Mining Association Rules in Large Databases," Proceedings of VLDB, Zurich, Switzerland, September 1995, pp. 432-444.

[9] R. Srikant, Q. Vu, and R. Agrawal, "Mining Association Rules with Item Constraints," Proceedings of KDD, Newport Beach, CA, August 1997, pp. 67-73.