A Comparative Study on Transaction Database Mining Algorithms

Hay Mar Soe, Nang Saing Moon Kham
University of Computer Studies, Yangon
haymarsoeucsy@gmail.com

Abstract

Association Rules is the process of identifying relationships among set of items in transaction database. Finding frequent itemsets is the most expensive step in Association rule discovery. Real world datasets are sparse, dirty and contain hundreds of items. In such situations, discovering interesting rules (results) using traditional frequent itemset mining approach is not appropriate. In this paper, mining association rules for Transaction database using FP-Growth and DynFP-Growth algorithms is presented. FP-Growth algorithm avoids or reduces candidate generation. Moreover, it greatly reduces the need to traverse the database. DynFP-Growth algorithm works in the same process as FP-Growth with lexicographic order, and stores at least one item is detected. Although resulting FP-Tree is too large to store in memory, it is more flexible with different support values. For the empirical study, three synthetic datasets by IBM Quest data generator is used: (a) T10I4D100K (b) T40I10D30K, and (c) T40I10D100K.

1. Introduction

Data mining has attracted a great deal of attention in the information industry is due to the wide availability of huge amounts of data and the immediate need for turning such data into useful information and knowledge. Data Mining is the process of extracting knowledge from large data sets. Association Rule is one of the major techniques or tasks in data mining, which can be simply defined as finding interesting rules from large collections of data. Association rule mining searches for interesting relationships among items in a given data set. After being usage profiles, to get interesting patterns (user like) with association rules mining. The rule A → B holds in the transaction set D with **support s**, where s is the percentage of transactions in D that contain AUB (i.e., both A and B). This is taken to be probability, P (A U B). The rule A → B has **confidence c** in the transaction set D if c is the percentage of transactions in D containing A that also contain B. This is taken to be the conditional probability, P (B|A). That is

$$\text{Support } (A \rightarrow B) = P(A \cup B)$$

$$\text{Confidence } (A \rightarrow B) = P(B \mid A)$$

Frequent itemset finding is the most expensive of the two steps, since the number of item sets grows exponentially with the number of items. FP Growth uses the FP-tree data structure to achieve a condensed representation of the database transactions and employs a divide-and-conquer approach to decompose the mining problem into a set of smaller problems. It mines all the frequent itemsets by recursively finding all frequent 1-itemsets (itemsets containing only 1 item) in the conditional pattern.

FP-Tree is efficiently constructed with the help of a node link structure. In order to optimize the performance of the FP-Growth algorithm, dynFP-Growth algorithm is presented to improve performance with different minimum support values.

The rest of the paper is organized as follows. Section 2 presents related work. Data mining and Association rule mining are described in section 3. Section 4 includes the proposed system design and system implementation. Experimental results of FP and Dynamic FP algorithms are presented in Section 5. Section 6 is the conclusion of the paper.

2. Related Work

An association rule is defined as the relation between the itemsets.

Frequent-pattern mining plays an essential role in mining associations. Most of the previous studies, adopt an Apriori-like approach, which is based on the anti-monotone Apriori heuristic [1]: if any length k pattern is not frequent in the database, its length (k + 1) super-pattern can never be frequent. The essential idea is to iteratively generate the set of candidate patterns of length (k+1) from the set of frequent-patterns of length k (for $k \geq 1$), and check their corresponding occurrence frequencies in the database.

The Apriori heuristic achieves good performance gained by (possibly significantly) reducing the size of candidate sets. However, in situations with a large number of frequent patterns, long patterns, or quite

low minimum support thresholds, an Apriori-like algorithm may suffer from the two nontrivial costs: (1) It is costly to handle a huge number of candidate sets. (2) It is tedious to repeatedly scan the database and checks a large set of candidates by pattern matching, which is especially true for mining long patterns.

FP-growth approach for mining frequent itemsets without candidate generation was proposed by Han in [2]. Its scalable frequent patterns mining method has been proposed as an alternative to the Apriori-based approach. The pattern growth approach adopts the divide-and-conquer methodology to produce the frequent itemsets. This algorithm creates a compact tree-structure, FP-Tree, representing frequent patterns, which moderates the multi-scan problem and improves the candidate itemset generation. This algorithm is faster than others in the literature.

Several algorithms implicate the methodology of the FP-growth algorithm. In [3] Pei used the same approach for Mining closed frequent itemsets and max-patterns. Likewise, Pei suggested to Mining sequential patterns in [4]. In [5,6,7], similar approach of Han et al [2] is adapted for mining the frequent itemsets from the transactional database. These algorithms are more efficient than FP-growth.

3. Data Mining and Association Rules

Data mining refers to extracting or mining knowledge from large amounts of data. It can be also defined as Knowledge Discovery in Databases, or KDD. Data mining is the process of extracting interesting information or patterns from large information repositories such as: relational database, data warehouses, XML repository, etc. The main process of KDD is the data mining process. In this process different algorithms are applied to produce hidden knowledge. Then, comes another process called post-processing, this evaluates the mining result according to users' requirements and domain knowledge.

Various data mining techniques are applied to the data source; different knowledge comes out as the mining result. That knowledge is evaluated by certain rules, such as the domain knowledge or concepts.

Given a set of transactions, where each transaction is a set of items, an association rule is a rule of the form $X \rightarrow Y$, where X and Y are sets of items (also called itemsets). The meaning of this rule is that the presence of X in a transaction implies the presence of Y in the same transaction. An example of an association rule in the market basket analysis domain is: "90% of transactions that contain bread and butter also contain milk; 30% of all transactions contain the three of them". Here, X = {bread, butter}, Y = {milk}, 90% is called the confidence of the rule, and 30% the support of the rule. The confidence of a rule measures the degree of the correlation between itemsets, while the support of a rule measures the significance of the correlation between itemsets. The problem of mining association rules is to find all association rules that are above the user-specified minimum support and minimum confidence.

Support $(X \rightarrow Y) = P(X \cup Y)$

Confidence $(X \rightarrow Y) = P(Y \mid X) = P(X \cup Y) / P(X)$

Process of mining association rules consists of two steps.

(1) **Find all frequent itemsets**: Each of these itemsets will occur at least as frequently as a pre-determined minimum support count.

(2) **Generate strong association rules from the frequent itemsets**: These rules must satisfy minimum support and minimum confidence.

3.1. FP-Growth

Information from transaction databases is essential for mining frequent patterns. Therefore, if we can extract the concise information for frequent pattern mining and store it into a compact structure, then it may facilitate frequent pattern mining. FP-tree stores complete but no redundant information for frequent pattern mining. Since only the frequent items will play a role in the frequent-pattern mining, it is necessary to perform one scan of transaction database to identify the set of frequent items.

If the set of frequent items of each transaction can be stored in some compact structure, it may be possible to avoid repeatedly scanning the original transaction database. If multiple transactions share a set of frequent items, it may be possible to merge the shared sets with the number of occurrences registered as count. It is easy to check whether two sets are identical if the frequent items in all of the transactions are listed according to a fixed order.

If two transactions share a common prefix, according to some sorted order of frequent items, the shared parts can be merged using one prefix structure as long as the count is registered properly. If the frequent items are sorted in their frequency descending order, there are better chances that more prefix strings can be shared. A frequent pattern tree is a tree structure defined below.

It consists of one root labeled as "root", a set of item prefix sub-trees as the children of the root, and a frequent-item header table. Each node in the item prefix sub-tree consists of three fields: item-name, count, and node-link, where item-name registers which item this node represents, count registers the number of transactions represented by the portion of the path reaching this node, and node-link links to the

next node in the FP-tree carrying the same item-name, or null if there is none.

Each entry in the frequent-item header table consists of two fields, (1) item-name and (2) head of node-link, which points to the first node in the FP-tree carrying the item-name. Table 1 presents the small set of transaction database and descending sorted items by frequent count. Figure 1 describes example generation of FP-Tree and Header table.

Table 1: Transactions and sorted transactions

TID	*Items*	*(ordered) frequent items*
001	{I6, I1, I3, I2}	{I1, I2, I3, I6}
002	{I3, I1, I4, I5, I2}	{I1, I2, I3, I4, I5}
003	{I4, I1, I2, I5}	{I1, I2, I4, I5}
004	{I2, I1, I3}	{I1, I2, I3}

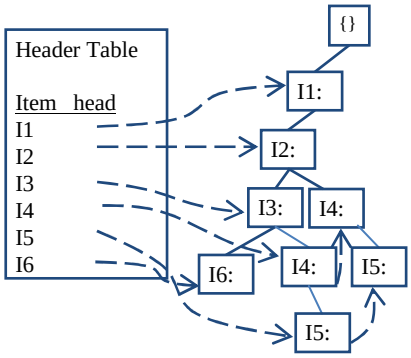

Figure 1: FP Tree Construction

3.2. Dynamic FP-Growth Algorithm

In the FP-Growth algorithm, whenever database is updated or different minimum support is used, FP-Tree has to be rebuilt. Therefore it costs the building FP-Tree for every changes. In the Dynamic FP-Growth algorithm, FP-Tree stores all items (in FP-Tree only frequent items are stored in the tree). For above reason, whenever minimum support is changed, FP-Tree does not need to be rebuilt. In storing the link structure in the FP-Tree, child node also saves the parent node's address. Therefore, in updating the database, only tree structure and header table has to be updated and tree does not need to rebuilt.

Algorithm for building dynamic FP tree is shown as below:

Algorithm BuildDynFPTree
Input: Transaction database TDB
Output: Header Table and Dynamic FP Tree
Begin
FPTree T = new FPTree();
T.root = CreateRoot (null);
header = new HeaderTable();
foreach trans in TDB
header.add(trans.item);
trans.sort (header.mastertable);
p = trans.firstitem;
P = trans.remainingitems;
InsertTree(p, P, T);
end for
End

Algorithm InsertTree(p, P, T)
Begin
If (T.children.contains(p.itemname))
T.children.IncreaseCount(p.itemname);
Else T.children.AddNode(p.itemname, 1);
If (P <> null)
InsertTree(P.firstitem, P.remainingitems, T);
End

Figure 2 describes how dynamic FP tree is generated from transactions in Table 1.

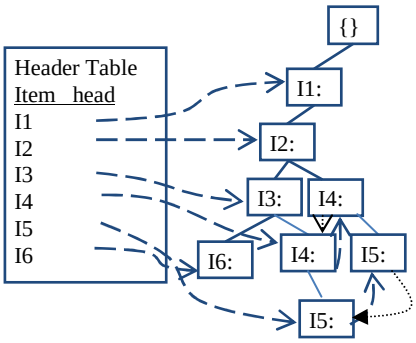

Figure 2: Dynamic FP-Growth

4. Proposed System

This paper presents the comparison of FP-Growth algorithm and dynamic FP-Growth algorithm. Transaction database is input to this system, and it is applied to both FP-Growth algorithm and dynamic FP-Growth algorithm. Frequent patterns will be generated from both algorithms. Then performance analysis for processing time is performed for both algorithms.

4.1. System Overview

FP-growth algorithm is an efficient method of mining all frequent itemsets without candidate's generation. The algorithm mine the frequent itemsets by using a divide-and-conquer strategy as follows: FP-growth first compresses the database representing frequent itemset into a frequent-pattern tree, or FP-tree, which retains the itemset association information as well. FP-Tree has to be rebuilt for the database update or minimum support change.

Dynamic FP-Growth algorithm enhances it by replacing single linked list with doubly linked list, i.e, parent link stores children nodes as well as children

nodes also save parent links. Although the resulting FP-tree could be too large to be stored in the main memory, and it is suitable with different minimum supports, it would be more practical from time consuming point of view. Using the dynamic FP-Tree, it does not need to rebuild the FP-tree even if the actual database is updated or minimum support value is changed. Figure 3 shows the overview of the system.

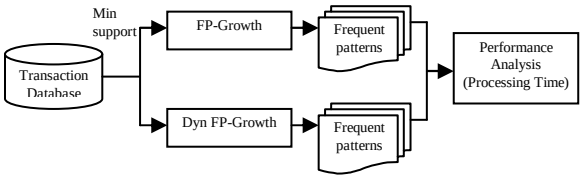

Figure 3: System overview

4.2. Transaction Database

In this paper three synthetic datasets by IBM Quest data generator are used. They are (a) T10I4D100K with transaction length 24, and the number of transactions 34693; (b) T40I10D30K, with transaction length 26, and number of transactions 31000, and (c) T40I10D100K with transaction length 24 and number of transactions 34310. Those data sets are obtained from (http://fimi.cs.helsinki.fi/data/).

4.3 System Implementation

This system is implemented using Microsoft Visual Studio 2008. ASP .Net C# is used to implement the system. Process flow of the system is shown in Figure 4.

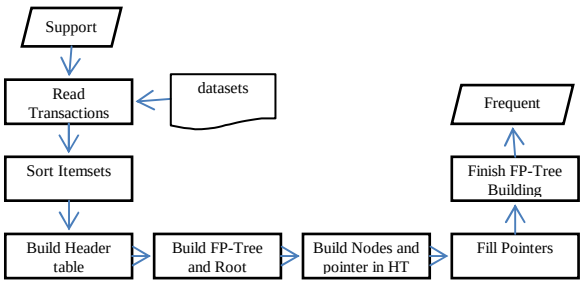

Figure 4: Process Flow of the System

As in Figure 4, first it reads transactions from datasets. Then items are sorted in a descending order according to their support count and transaction sets are prepared in such order. Then header table and FP-Tree are built. In FP-Growth algorithm, header table contents are only large items (i.e., items whose support ≥ min sup), while dynamic FP-Growth contains all items even though only one occurrence exists in Transaction database. In the header table and fp-tree of FP-Growth, header information stores only children's position and in those of dynamic FP-Growth contains both parent and children's positions.

5. Experimental Results

FP-Growth and Dyn-FP-Growth algorithms are tested with different datasets with different data set sizes, and also with different minimum support values. Performance analysis is performed mainly on the processing time in milliseconds. Experimental results of the system are shown in Table 2.

Table 2: Experimental results in milli-seconds

No	File	Count	Alg	minimum support	Tree building time	Pattern Extraction time
1.	T10I4D100K.dat	34692	FP	0.01	20.92	44.27
2.	T10I4D100K.dat	34692	Dyn-FP	0.01	20.09	38.67
3.	T10I4D100K.dat	34692	FP	0.02	22.04	19.22
4.	T10I4D100K.dat	34692	Dyn-FP	0.02	0	19.45

Figure 5 is the experimental results of tree building time for FP and Dynamic FP with different minimum supports. Minimum support used in this experiment is in decimal value. The minimum support count is min_sup x the number of transactions. According to experimental results, Dynamic FP takes time only in tree building process. And tree is built only once in dynamic FP. Therefore, for multiple minimum support values, it takes time only in the first run for the tree building process. In next continuous runs it takes only a small amount time nearly zero as in Figure 5.

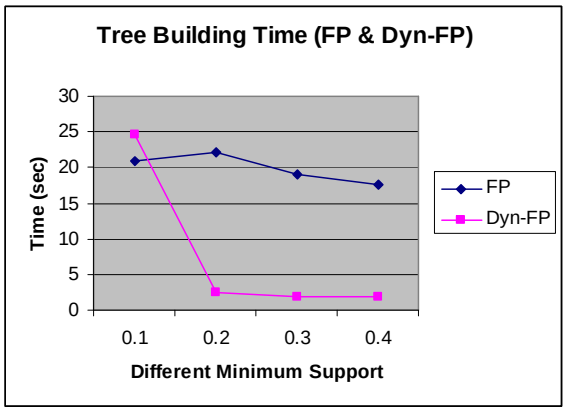

Figure 5: Experimental Results for Tree building time of FP and Dynamic FP

6. Conclusion

Performance analysis of FP-Growth algorithm and dynamic FP-Growth algorithm is presented in this paper. FP-growth algorithm is an efficient method of mining all frequent itemsets without candidate's generation. An optimization approach is added to FP-Growth algorithm in order to avoid rebuilding the FP-Tree in database updates. The data is challenging due to the number of characteristics which are the number of the records, and the sparseness of the data (each records contains only small portion of items). For empirical study, different synthetic datasets are used to prove the efficiency of the algorithms. According to the experimental results, performance of dynamic FP-Growth is better than FP-Growth algorithm starts from its second run.

7. References

[1] Agarwal, R., Aggarwal, C. and Prasad, V. V. V., "Depth-Firrst generation of large itemsets for association rules". IBM Tech. Report RC21538, July 1999.

[2] Han, J. , Pei, J. , & Yin, Y. "Mining frequent patterns without candidate generation". In Proc. ACM-SIGMOD Int. Conf. Management of Data (SIGMOD '96), Page 205-216, 2000.

[3] J. Pei, J. Han & R. Mao. CLOSET: An Efficient Algorithm for Mining Frequent Closed Itemsets", DMKD'00, 2000.

[4] J. Pei, J. Han, H. Pinto, Q. Chen, U. Dayal, and M.-C. Hsu. PrefixSpan: Mining Sequential Patterns Efficiently by Prefix-Projected Pattern Growth. ICDE'01, 2001.

[5] Liu,G. , Lu ,H. , Yu ,J. X., Wang, W., & Xiao, X.. "AFOPT:An Efficient Implementation of Pattern Growth Approach", In Proc. IEEE ICDM'03 Workshop FIMI'03, 2003.

[6] Grahne, G. , & Zhu, J. "Fast Algorithm for frequent Itemset Mining Using FP-Trees",IEEE Transactions on Knowledge and Data Engineer,Vol.17,NO.10, 2005 .

[7] Gao, J. "Realization of new Association Rule Mining Algorithm" Int. Conf. on Computational Intelligence and Security ,IEEE, 2007.