An FP Tree based Approach for Extracting Frequent Pattern from Large Database by Applying Parallel and Partition Projection

Jagrati Malviya
BU Bhopal
Department of Computer Science & Engineering
BUIT, BU Bhopal

Anju Singh
BU Bhopal
Assistant Professor
Department of Computer Science & Information

Technology UTD, BU Bhopal
Divakar Singh
BU Bhopal
HOD
Department of Computer Science & Engineering
BUIT, BU Bhopal

ABSTRACT

There are lots of data mining tasks such as association rule, clustering, classification, regression and others. Among these tasks association rule mining is most prominent. One of the most popular approaches to find frequent item set in a given transactional dataset is Association rule mining. Frequent pattern mining is one of the most important tasks for discovering useful meaningful patterns from large collection of data. The FP Growth algorithm is currently one of the fastest approaches to frequent item set mining.

This paper proposed an efficient and improved FP Tree algorithm which used a projection method to reduce the database scan and save the execution time. The advantage of PFP Tree is that it takes less memory and time in association mining.

Experimental result showed that the improved PFP Tree algorithm performs faster than FP growth Tree algorithm and partition projection algorithm. It is more efficient and scalable in the case of large volume of data. The effectiveness of the method has been justified over a sample our one super market database.

Keywords

Association Rule mining, Data Mining, Frequent Pattern Mining, Parallel Projection, Partition projection.

1. INTRODUCTION

Data mining has become an important field of research and has found a wide range of applications across to various areas. Data mining is a technique useful for attaining useful information from vast databases [12].Frequent item set mining (FIM) is a useful tool for discovering frequently co-concurrent items. Since its inception, a number of significant FIM algorithms have been developed to speed up mining performance [2]. Association rule mining is one of the most important data mining problems. The conviction of association rule mining is the discovery of association relationship among a set of items [4, 11].

In 1994, Agrawal proposed the famous Apriori algorithm, but there are two drawbacks in it. First, because it repeatedly scans the transaction database, it needs a lot of I/O load; second, it will cause huge candidate set. FP-Growth algorithm is a good result to the above two problems. The biggest advantage of the FP-Growth algorithm is that it only scans database twice. It directly compresses the database into a frequent pattern tree instead of using a candidate set and finally generates association rules through FP-tree [6]. FP-Growth Tree is more efficient than tree projection but it is difficult to maintain it in memory so tree projection is used in tree projection two types of projection is used parallel projection is good but it takes more memory but partition projection takes more time is execution but takes less apace compare to parallel projection[5].

2. RELATED WORK

According to the past research in the field of frequent pattern generation the FP Growth is used most widely. FP-growth method is efficient algorithm to mine frequent patterns, in spite of long or short frequent patterns. By using compact tree structure method, partitioning-based method and divide-and-conquer method, it reduces the search costs substantially.

The first known proposed method for extracting frequent patterns is Apriori algorithm proposed by Agrawal [1,3]. There have been developed enormous modified versions to improve it. Since the main drawback for Apriori-based algorithms was involving multiple database scans and generation of a large number of candidates, it was not appropriate.

Association mining using Apriori algorithm perform better but in case of large database it performs slow because it has to scan the full database each time while scanning the transaction. In comparison with Apriori algorithm FP is much superior in case of efficiency. But problem with traditional FP is that it produces a huge number of conditional FP-Tree [14].

Divide-and-conquer technique is used to decompose the mining task into a set of smaller tasks for mining confined patterns in conditional databases, which dramatically reduces the search space, frequent pattern mining and association rule mining we call this temp database as Projection Database, we can create a temp database for storing all the frequent items ordered by the list of frequent items which is used for projecting, reduce the expensive costs of individual node computation The case that may happen in a very large database[13].

FP-Growth is the first successful tree base algorithm for mining the frequent item sets. As for large databases its structure does not fit into main memory therefore new techniques come into pictures which improve the efficiency of FP Growth tree by applying projection techniques. A database projection method has been developed to cope with the situation when an FP-tree cannot be held in main memory [8, 9].

3. MINING FREQUENT PATTERN USING FP GROWTH TREE

3.1 FP Growth Tree

FP-Growth works in a divide and conquers way. It requires two scans of the database. In first scan of the database FP-Growth first computes a list of frequent items sorted by frequency in descending order (F-List). In second scan of the database, the database is compressed into a FP-tree. After that FP-Growth starts to mine the FP-tree for each item whose support is larger than ξ by recursively building its conditional FP-tree.

The FP-tree is a compressed representation of the transactions, and it also al- lows quick access to all transactions that share a given item. Once the tree has been constructed, the subsequent pattern mining can be performed. However, a compact represent- ton does not reduce the potential combinatorial number of candidate patterns, which is the bottleneck of FP-Growth [2].

3.2 Algorithm to find Frequent Item sets using FP-Growth algorithm

The FP- Growth algorithm for mining frequent patterns using FP-Tree is follows:

Input: A transaction database (D) and minimum support threshold (ξ).

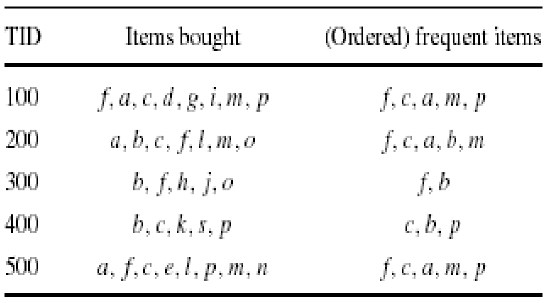
Output: The complete set of frequent patterns.

Method:

```
Call FP-growth (FP-tree, null)
Procedure FP-growth (Tree, A)
  {
If (Tree contains a single path P)
Then for each (combination (denoted as B) of the nodes in the path P)
Do
generate pattern B∪A with support = minimum support of nodes in B;
else (for each ai in the header of Tree) do
  {
generate pattern B = ai∪A with support = ai.support;
construct B's conditional pattern base and then B's conditional FP-Tree Tree B;
   if (Tree B ≠ ∅)
  {
    call FP-growth (Tree B, B)
   }
  }
}
```

3.3 Example of FP Growth Tree

Table 1 Show a simple example



TID	Items bought	(Ordered) frequent items
100	*f, a, c, d, g, i, m, p*	*f, c, a, m, p*
200	*a, b, c, f, l, m, o*	*f, c, a, b, m*
300	*b, f, h, j, o*	*f, b*
400	*b, c, k, s, p*	*c, b, p*
500	*a, f, c, e, l, p, m, n*	*f, c, a, m, p*

With the above observations, one may construct a frequent-pattern tree as follows.

First, a scan of DB derives a list of frequent items, (f: 4), (c: 4), (a: 3), (b: 3), (m: 3), (p: 3) in which items are ordered in frequency-descending order. This ordering is important since each path of a tree will follow this order.

Second, the root of a tree is created and labeled with "null".

The FP-tree is constructed as follows

1. The scan of the first transaction leads to the construction of the first branch of the tree: (f: 1), (c: 1), (a: 1), (m: 1), (p: 1) Notice that the frequent items in the transaction are listed according to the order in the list of frequent items.
2. For the second transaction, since its (ordered) frequent item list f, c, a, b, m_ shares a common prefix f, c, a_ with the existing path f, c, a, m, p_, the count of each node along the prefix is incremented by 1, and one new node (b:1) is created and linked as a child of (a:2) and another new node (m:1) is created and linked as the child of (b:1).
3. For the third transaction, since its frequent item list f, b shares only the node f with the f -prefix sub tree, f 's count is incremented by 1, and a new node (b:1) is created and linked as a child of (f :3).
4. The scan of the fourth transaction leads to the construction of the second branch of the tree, (c: 1), (b: 1), (p: 1) _.
5. For the last transaction, since its frequent item list _ f, c, a, m, p_ is identical to the first one, the path is shared with the count of each node along the path incremented by 1.

4. CONSTRUCTING FP GROWTH TREE USING PROJECTION

4.1 Projection

Database projection means partition a database into a set of projected database.Construct and mine FP Tree once the projected Databasse can fit into main memory.

There are two types of database projection: parallel projection and partition projection.

Parallel projection is implemented as follows: Scan the database to be projected once, where the database could be either a transaction database or aα-projected database. For each transaction T in the database, for each frequent item ai in T, project T to the ai- projected database based on the transaction projection rule, species in the dentition of projected database. Since a transaction is projected in parallel to all the projected databases in one scan, it is called parallel projection.

Partition projection is implemented as follows. When scanning the database (original or α-projected) to be projected, a transaction T is projected to the ai-projected database only if ai is a frequent item in T and there is no any other item after ai in the list of frequent items appearing in the transaction. Since a transaction is projected to only one projected database at the database scan. After the scanning of the database, it is partitioned by projection into a set of projected databases, and hence it is called partition projection.

The projected databases are mined in the reversed order of the list of frequent items .That is, the projected database of the least frequent item is mined first and so on. Each time when a projected database is being processed, to ensure the remaining projected databases obtain the complete information, each transaction in it is projected to the aj-projected database, where aj is the item in the transaction such that there is no any other item after aj in the list of frequent items appearing in the transaction [5].

4.2 Construction Process of PFP Tree

(1) we can create a temp database for storing all the frequent items ordered by the list of frequent items L. we call this temp database as Projection Database (or PDB for short), which is used for projecting, reduce the expensive costs of individual node computation.

(2) We can project the PDB, two columns at a time. One column is used to compute the count of each different item, the other (previous) column is used to distinguish the node's parent node of current column. By this way, we can insert one **level** of nodes into the tree at a time, not compute frequent items one by one. Then, the algorithm performance is only related to the **depth** of tree, namely the number of frequent items of the longest transaction in the database η, not the sum of frequent items in the database [10].

(3) Because we only project two columns at a time, only save the information of the current nodes and their parent nodes, if there exist the case as follows: the current nodes' parent nodes are identical, but their parent nodes are different, we couldn't judge how to deal with it.

If we add their count regarding them as the same node, we make a mistake; because they are different nodes belong to different parent nodes. When encounter this case, we can add the parent nodes' name as the TAI to the current nodes, and save to the PDB. Then when the next projection, we can distinguish these nodes by their parent nodes' TAI [7, 8].

4.3 Proposed Algorithm

In this paper new algorithm is proposed which combined FP Growth Tree algorithm and projection algorithm.

Algorithm PFP-tree construction

Input: A transaction database and a minimum support threshold ξ.

Output: PFP-tree

Method:

(1). First scan the transaction database once. Collect the set of frequent items F and their supports. Sort set of frequent items F in support descending order as L.

(2). Select and sort the frequent items in transaction according to the order of L, the result is saved in the PDB.

(3). Create the root of an FP-tree, T, and label it as "null". Let column number in PDB be j, the initial value of j is 1.

If j = 1

The process is implemented as follows:

First project the column (j-1) and column (j), then add 1 to j, and project column (j-1) and column (j) circularly, and so on, until project the last column of PDB.

Then Do

{

Project the column (1), collect the set of frequent items and their supports, let the result be [q: n], where q is the frequent item, n is the count; Insert these nodes as the root's **child** nodes into the PFP-tree.

}

Else Do

{

(1) Project both parent column (j-1) and current column (j), compare the set of Binary-frequent items and collect their supports1. Let the result be [px, q:n], where p is the parent frequent item of column(j-1), x. is p's TAI and q is the current frequent item of column (j), n. Linked to the nodes with the same item-name via the node-link structure.

(2) Compare the result sets of [px, q: n], if their current frequent item name, q are identical, then add the count.

(3) Insert The node [qy:n]or[q:n] as the child nodes of px into the PFP-tree and let their node-link be linked to the nodes with the same item-name via the node-link structure.

}

(4) Delete all the TAI in the PFP-tree and PDB (this step can be cancelled).

5. EXPERIMENTAL EVALUATION AND PERORMANCE STUDY

In this section we present a performance comparison of FP Growth with FP Parallel Projection and FP Partition Projection algorithm. All the experiments of three algorithms are performed on computer with a 3-GHz processor Pentium PC machine with 512MB main memory. The algorithm implemented on Microsoft Windows/NT. The proposed algorithm is implemented in Microsoft Visual studio .net using (C# 7.0).

Please note that run time used here means the total execution time, that is, the period between input and output, instead of CPU time measured in the experiments in some literature.

The experimental results are showed in Figure 1 and Figure 2 respectively. Experimental results shows that PFP Tree algorithm works much faster than FP Growth Tree algorithm and partition FP Growth tree algorithm because it doesn't need to generate 2-candidate item sets and reduce the search space. PFP tree algorithm runs faster than FP-growth, because in the case, FP-growth needs to construct a large of conditional sub trees, it is not only time-consuming but also high memory cost. But in the case , parallel projection is good but it takes more memory but partition projection takes more time is execution but takes less apace compare to parallel projection. PFP doesn't need too much extra spaces and time on the mining process, so PFP tree algorithm has a better scalability.

Our First transaction Dataset:
This paper explores the PFP Tree algorithm on the sample of super market dataset; it is a synthesized data set, which is illustrated in Table 2**.**

TRANSACTION_ID	ITEM SET
T01	BREAD,MILK, BISCUIT, CORNFLAKES
T02	BREAD,TEA, BOURNVITA
T03	JAM ,MAGGI,BREAD,MILK
T04	MAGGI,TEA, BISCUIT
T05	BREAD,TEA, BOURNVITA
T06	MAGGI,TEA, CORNFLAKES
T07	MAGGI,BREAD,TEA,BISCUIT
T08	JAM ,MAGGI,BREAD,TEA
T09	BREAD,MILK
T10	COFFEE,COCK, BISCUIT, CORNFLAKES
T11	COFFEE,COCK, BISCUIT, CORNFLAKES
T12	COFFEE,SUGER, BOURNVITA
T13	BREAD,COFFEE,COCK
T14	BREAD,SUGER, BISCUIT
T15	COFFEE,SUGER,CORNFLAKES
T16	BREAD,SUGER, BOURNVITA
T17	BREAD,COFFEE,SUGER
T18	BREAD,COFFEE,SUGER
T19	TEA,MILK,COFFEE, CORNFLAKES
T20	MILK, BREAD, BISCUIT

(A) Comparison on the basis of execution time and minimum support count between FP-Growth Tree conditional pattern, FP-Growth Tree with DB parallel projection and Partition projection.

Table 3: Comparison Table of 3 algorithms on the basis of minimum support count and execution time

Minimum support count	Time taken to execute (In millisecond) FP-Growth Tree	Time taken to execute (In millisecond) FP-Growth Tree with Data base Parallel Projection	Time taken to execute (In millisecond) FP-Growth Tree with Data base Partition projection
2	132	102	153
3	124	85	131
4	84	60	95
5	74	45	79

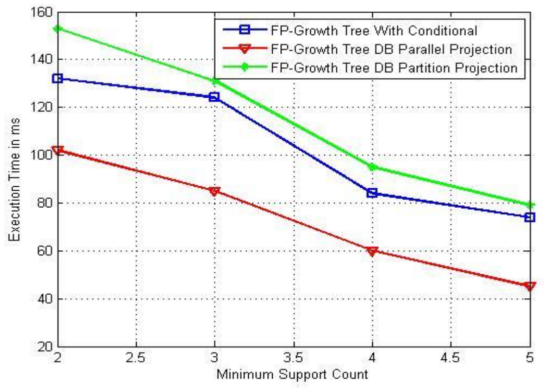


Fig 1: Figure representing the comparison of FP-Growth Tree, Data base Parallel Projection and Partition projection when minimum support count varying

(B) Comparison on the basis of execution time and no of records between FP-Growth Tree conditional pattern, FP-Growth Tree with DB parallel projection and Partition projection.

Table 4: Comparison Table of 3 algorithms on the basis of number of records and execution time

No of records	Time taken to execute (In millisecond) FP-Growth Tree	Time taken to execute (In millisecond) FP-Growth Tree with Data base Parallel Projection	Time taken to execute (In millisecond) FP-Growth Tree with Data base Partition projection
200	84	60	91
300	97	70	102
400	145	132	156
500	196	180	222

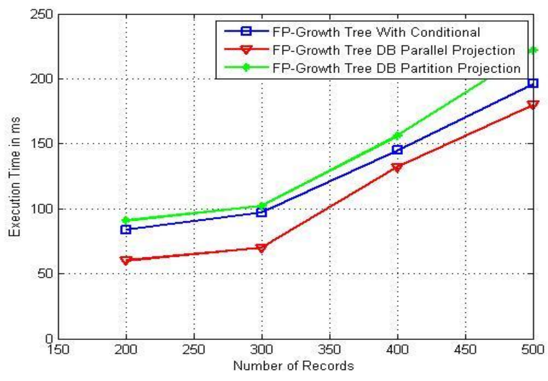


Fig 2: Figure representing the comparison of FP-Growth Tree, Data base Parallel Projection and Partition projection when no of records varying

6. CONCLUSION

This paper proposed a new algorithm which combined FP Growth Tree algorithm and projection algorithm which deals with large size database.

The experimental results show that this new algorithm (PFP Tree) works much faster than FP Growth Tree algorithm. Proposed algorithm is the enhancement of FP Tree technique of association with parallel and partition projection integration on it. Traditional FP Tree method performs well but generates redundant trees resulting efficiency degrades. To achieve better efficiency in association mining parallel and partition projection techniques helps out.

Result shows that proposed method perform well and handle very large size of data set.

There are several advantages of PFP-growth over other approaches:

(1) It constructs a highly compact PFP-tree, which is usually substantially smaller than the original database, and thus saves the costly database scans in the subsequent mining processes.

(2) By using projection technique into the process of tree-construction, we save the expensive frequent items scans in PFP Algorithm, which hugely shortens the time of tree-construction. And the performance is much more scalable than the FP-Growth method.

(3) It applies a pattern growth method, which avoids costly candidate generation.

(4) It applies a partition -based divide-and-conquer method, which dramatically reduces the size of the subsequent conditional pattern bases and conditional PFP-trees.

Our performance study shows that the method mines both short and long patterns efficiently in large databases.

Parallel projection takes more memory space but less time than FP Growth Tree and Partition projection. But partition projection takes less memory space but more time than FP Growth Tree and Parallel projection. The parallel projection is more scalable and efficient than partition projection and FP Growth Tree.

7. FUTURE WORK

Another issue related to FP-tree materializations how to incrementally update an FP-tree, such as when adding daily new transactions in to a database containing records accumulated for months. If the materialized FP-tree takes 1 as its minimum support (i.e., it is just a compact version of the original database), the update will not cause any problem since adding new records is equivalent to scanning additional transactions in the FP-tree construction. However, a full FP-tree may be an undesirably large. Thus setting 1 as its minimum support may not be a good solution. When support is very low, FP-tree becomes bushy. In such cases, the degree of sharing in branches of FP-tree becomes low. The overhead of links makes the size of FP-tree large. Therefore, instead of building FP-tree, we should construct projected databases. That is the reason why we build FP-tree for transaction database/projected database only when it passes certain density threshold. From the experiments, one can see that such a threshold is pretty low, and easy to touch. Therefore, even for very large and/or sparse database, after one or a few rounds of database projection, FP-tree can be used for all the remaining mining tasks.

8. REFERENCES

[1] R. Agrawal and R. Srikant.," Fast algorithms for mining association rules", VLDB, 1994, pp 487-499.

[2] Li Haoyuan, Yi Wang, Zhang Dong, Zhang Ming, Chang Edward, "PFP: Parallel FP Growth for query Recommendation".

[3] Jiawei Han, M. Kamber, "Data Mining-Concepts and Techniques", Sam Francisco 2009, Morgan Kanufmann Publishers.

[4] R Agrawal, T. Imielinski, and A. Swami, "Mining association rules between sets of items in large databases", In Proc.1993 ACM-SIGMOD Int. Conf. Management of Data, May 1993, Washington, D.C., pp 207–21.

[5] Jiawei Han, Jian Pei, Runying Mao," Mining Frequent Patterns without Candidate Generation: A Frequent-Pattern Tree Approach", Data Mining and Knowledge Discovery, April 2001, Kluwer Academic Publishers, Manufactured in The Netherlands.

[6] Lijuan Zhou , Xiang Wang, "Research of the FP Growth algorithm based on Cloud Environment" , Journal of Software, March 2014,volume 9, N0. 3.

[7] J. Han, J. Pei, B. Mortazavi-Asl, Q. Chen , U. Dayal , and Hsu, M.-C. FreeSpan, "Frequent pattern-projected sequential pattern mining", ACM SIGKDD, 2010.

[8] J. Han, J. Pei, and Y. Yin , "Mining Frequent Patterns without Candidate Generation", SIGMOD 2000, pp 1-12.

[9] H. Huang , X.W , and R. Relue , "Association Analysis with One Scan of Databases", Proceedings of the IEEE International Conference on Data Mining, 2002.

[10] R. Agrawal, C.C. Aggarwal , and V. V. V. Prasad," A Tree Projection Algorithm For Generation of Frequent Itemsets", Journal on Parallel and Distributed Computing[(Special Issue on High Performance Data mining)], 2010.

[11] Jagrati Malviya , Anju Singh , " A comparative study of various database techniques for frequent pattern generation", ACSIT Nov 2014.

[12] Vikram Garg , Anju Singh , Divakar Singh , "A Hybrid Algorithm for Association Rule Hiding using Representative Rule", International Journal of Computer Applications (IJCA)2014.

[13] C.C. Agarwal, "An Introduction to uncertain data algorithm and applications", Advances in Database Systems, 2009, 35; pp 1–8.

[14] Rashmi Shikhariya, Nitin Shukla , "An improved association rule mining with FP Tree using positive and negative integration",Journal of global research in computer science (JGRCS), Oct 2012. Volume 3, No. 10.