ISSN No. 0976-5697

Volume 3, No. 5, Sept-Oct 2012

International Journal of Advanced Research in Computer Science

RESEARCH PAPER

Available Online at www.ijarcs.info

An Adaptive and bounded Approach to Mine Frequent Pattern in Large Scale Databases

C. Vinothini*
Assistant professor,Department of CSE,
Dr.NGP Institute of Technology,
Coimbatore,India
Vinucrazy56@gmail.com

E. Meenachi
Assistant professor,Department ofCSE,
Park College of Engineering and Technology,
Coimbatore,India
meenakshy88@gmail.com

Abstract: Frequent Patterns are very important in knowledge discovery and data mining process such as mining of association rules, correlations etc. Many existing incremental mining algorithms are Apriori-based, which are not easily adoptable to solve association rule mining. In FP-tree isa compact representation of transaction database that contains frequency information of all relevant Frequent Patterns (FP) in a dataset. Mining association rules among items in a large database has been recognized as one of the most important data mining problems. An earlier approach proposes a model that is capable of mining in transactional database, but that approach is not capable of managing the problem of changing the memory dynamically. In order to solve this problem we have been proposed a hybrid of two algorithms that could be able to handle the dynamic change of memory, dynamic databases and also to solve the problem of association rule mining problems. So memory can be utilized effectively in large scale transaction database.

Index terms: Frequent Patterns, transcation database, Apriori algorithm, Association rule, FP tree

I. INTRODUCTION

Data mining is a term that refers to searching a large data set in an attempt to detect hidden or low-level patterns. Data mining is becoming increasingly common in both the private and public sectors. Industries such as banking, insurance, medicine, and retailing commonly use data mining to reduce costs, enhance research, and increase sales. In the public sector, data mining applications initially were used as a means to detect fraud and waste, but have grown to also be used for purposes such as measuring and improving program performance. However, some of the homeland security data mining applications represent a significant expansion in the quantity and scope of data to be analyzed.

In this paper association rule mining defines finding frequent patterns, associations, correlations, or causal structures among sets of items or objects in transaction databases, relational databases, and other information repositories. The main Applications are Basket data analysis, cross-marketing, catalog design, loss-leader analysis, clustering, classification, etc.[1]

Association rule mining is to find out association rules that satisfy the predefined minimum support and confidence from a given database. The problem is usually decomposed into two sub problems. One is to find those itemsets whose occurrences exceed a predefined threshold in the database; those itemsets are called frequent or large itemsets. The second problem is to generate association rules from those large itemsets with the constraints of minimal confidence.

The task of discovering all frequent itemsets is quite challenging. The search space is exponential in the number of items occurring in the database. The support threshold limits the output to a hopefully reasonable subspace. Also, such databases could be massive, containing millions of transactions, making support counting a tough problem. Frequent pattern mining has been studied for over a decade with tons of algorithms developed

- Apriori
- FPgrowth (it only considered in this paper)

Large data are data sets characterized by a large number of columns (i.e., attributes)and few rows (i.e., transactions). We propose in this paper a method to mine frequent from a large data without transposing the data set. The key idea is to use extension of a pattern to check these constraints, because the extension has few objects in large databases.[3]We show a new property to compute the extension of a pattern and a new pruning criterion. Their simultaneous use is on the core of the hybrid of adaptive and bounded algorithm that we propose to extract the frequent patterns from large database.

II. RELATED WORK

Han et al. proposed FP-growth algorithm [4] to discover frequent patterns from FP-tree. FP-growth traverses the FP-tree in a depth-first manner. It requires only two scans of the dataset to construct FP-tree, unlike Apriori algorithm [6] that makes multiples cans over the dataset. Since the introduction of FP-growth algorithm three major algorithms have been proposed, namely AFPIM, CATS tree, and Can Tree that have adopted FP-tree for incremental mining of frequent patterns. Several efficient algorithms have been proposed for finding frequent itemsets and the association rules are derived from the frequent item sets, such as the Apriori[4] and DHP algorithms.

APRIORI: 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 following two nontrivial costs:

a) It is costly to handle a huge number of candidate sets.

b) It is tedious to repeatedly scan the database and check a large set of candidates by pattern matching, which is especially true for mining long patterns.[14]

a. Cats tree:

CATS tree (Compressed and Arranged Transaction Sequence Tree) [5] mainly for interactive mining. The CATS tree extends the idea of the FP-tree to improve storage compression, and allows frequent-pattern mining without the generation of candidate itemsets. Once cats are built, it can be used for multiple frequent pattern mining with different supports. The idea of tree construction is as follows. It requires one database scan to build the tree. New transactions are added at the root level. At each level, items of the new transaction are compared with children (or descendant) nodes. If the same items exist in both the new transaction and the children (or descendant) nodes, the transaction is merged with the node at the highest frequency level. The remainder of the transaction is then added to the merged nodes, and this process is repeated recursively until all common items are found. Any remaining items of the transaction are added as a new branch to the last merged node. If the frequency of anode becomes higher than its ancestors, then it has to swap with the ancestors so as to ensure that its frequency is lower than or equal to the frequencies of its ancestors. Let us consider the following example to gain a better understanding of how the CATS tree is constructed.[4]

Example 1 *Consider the following database*

Tid	Transactions
1	r,s,t,u
2	q, s, t
3	p,q,r, t
4	p,s, t,u
5	p,r,s, t

Tid	Transactions
1	t, s, r
2	t, s
3	t, r, p
4	t, s, p
5	t, s, r, p

Figure 1. a) Initial Dataset b) Projected Dataset with minthreshold=50%

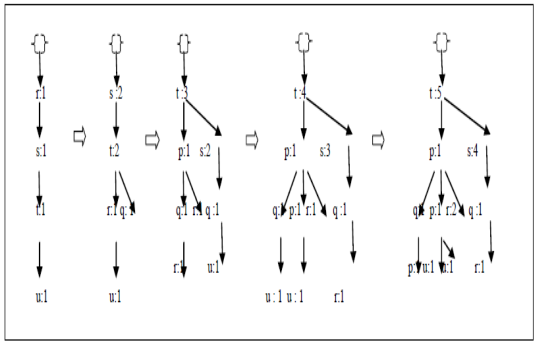

Figure: 2 Step wise Construction of CATS tree while processing each Transaction

Figure 2 show CATS tree is constructed considering the dataset of Figure 1. However, CATS tree too has two limitations. First, for each new transaction it is required to find the right path for the new transaction to merge in. Second, it is required to swap and merge the nodes during the updates, as the nodes in CATS tree arelocally sorted.[8]

b. Can tree:

CanT ree (**Can**onical-order **Tree***)*, that captures the content of the transaction database and orders tree nodes according to some canonical order.[7] The canonical order can be determined by the user prior to mining process. Canonical ordering can be lexicographic or based on certain property values of items By exploiting its nice properties, the Can Tree can be easily maintained when database transactions are inserted, deleted, and/or modified.[10] For example, the Can Tree does not require adjustment, merging, and/or splitting of tree nodes during maintenance. No rescan of the entire updated database or reconstruction of a new tree is needed for incremental updating.[12]

c. Fp tree:

In this paper, we consider the FP-growth approach, Han *et al.* [4] suggested that the FP-tree can be saved in the secondary memory if it cannot be accommodated in main memory. FP tree is a fundamental data structure for FP growth. Their proposal is a disc-resident version of the FP-tree that would use the *B*+-tree, which is a very popular structure. They proposed that the top level nodes of the *B*+-tree can be split based on the roots of item prefix subtrees, and the second level based on the common prefix paths, and so on. They also proposed a group accessing scheme to improve the I/O access related to the disc-based nodes so that one would traverse nodes that are in memory first before fetching other nodes from disc.

The general idea of mining frequent pattern using FP tree is divide and conquer method, it happens recursively grow frequent pattern path. The method of mining the frequent pattern is for each item, it constructs its conditional pattern base, then its conditional FP tree. Again, repeat the process until the resulting FP tree is empty, or it contains only one path. For this we considered an algorithm as Adaptive and Bounded memory based approach to mine frequent pattern.[13]

III. FP TREE CONSTRUCTION

We mainly present a running example to demonstrate the different aspects related to Han *et al.*'s [2] FP-growth and FP-tree as the underlying structure because both are required to thoroughly understand the approach proposed in this paper.

In this section, we describe the construction procedure of the FP-tree and its algorithm via an illustrative example.

Lets consider an example for mine frequent pattern,

Table: 1 A Transaction data Base as runing 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*

Table I shows a transactional database with six transactions. Every row is identified by TID and consists of a set of items that occur together in a single transaction. In order to construct the FP-tree from this transactional database, first scan the database once to collect the count of the items present in the database. Then, sort the items according to their frequencies in descending order to build the frequent item list. Only items that meet the minimum

support threshold are considered for building the FP-tree. All items with their frequency counts are shown are the items sorted according to their frequencies in descending order. Only items that meet the minimum support threshold are shown in tabulation. Finds and arranged the frequent pattern with table (1).Once the tree is constructed; it is possible to mine for frequent patterns using the FP-Growth approach, which is a recursive mining approach that divides the mining task into separate smaller ones. The FP-Growth approach starts mining by looking up the header table and selecting items that meet the specified minimum support criteria.

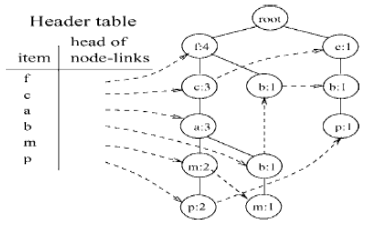

Figure: 3TheFP- Tree in Example 1

After construction of FP tree, child node is added to the branch while maintaining the prefix path property of the FP-Tree. But the time when this prefix path is created and where in the linear memory space the nodes of the prefix path are placed, depend on the order by which the transactions are read and added to the file. So nodes are numbered and store in linear memory as array like fashion. Basically FP growth suffers from spatial and temporal localities. To improve the spatial and temporal localities from FP growth are:

a. Approximate Sorting
b. Reorganizing Data, I/O Conscious FP-Tree
c. Reorganizing Computation, Page Blocking
d. Issues Associated With Relying on VMM

IV. PROBLEM DEFINITION

In this section, we will describe the problem in this paper. Let's consider raw transactional databases, total main memory (M), secondary storage(S) are available. Based on this, finds all frequent patterns which corresponds to database and its support threshold. Similarly, an ARM model, finds from database, all association rules that meet the desired thresholds for support σ and confidence c. traditionally, there are no restrictions on the use of M and S. A particular mining model may use any portion of M and/or S that the model requires to find the frequent patterns/association rules. Moreover, the model must assume that its memory is limited and hence cannot grow beyond the already specified size; it is possible that other parts of the main memory may be assigned to other tasks that are independent of the mining task.[6]

V. BOUNDED AND ADAPTIVE APPROACH

In this section, we present the architecture of the proposed frequent pattern mining model , which can mine frequent patterns management unit of the model allocates a contiguous portion of main memory *M.B*, where structures required for the mining task, which includes the following:

a) *Size Of* (*M.B*) = *B* × *size Of* (*M*)[9]. This allocated contiguous block of memory stores all the data using a bounded portion of the main memory. Each component of the proposed architecture shown in Figure. Based on this ratio *B*, the memory 1) FP-nodes list (which is an array of FP-nodes where the initial FP-tree, and subsequently, the conditional FP-trees are realized);
b) 2) Memory Tree Location Table (MTLT);
c) 3) Disc Tree Location
d) Table (DTLT);
e) 4) Item Location Table (ILT);
f) 5) Count Location Table (CLT); and
g) 6) FB to load blocks of the disc-based prefix tree

A. Secondary storage:

Our proposed mining model realizes the secondary storage *S* as a collection of file-based structures. If at any certain point of the mining process, the MMU cannot accommodate the data structures in *M.B*, it translates the overflowed data structures into linear file-based structures.[2]

B. Memory management unit:

The MMU lies all the translations necessary to save the overflowed memory-based data structures (like FP-trees, MTLT, DTLT, etc.) into the secondary storage *S* as file-based structures. It also loads the data structures from the file-based structures to the desired memory-based structures when required.[9] The tree translation unit translates a memory-based FP-tree which are located at the FP-nodes-list into a disc(file)-based version of I/O-conscious prefix-tree. Next, the tree is divided into two blocks, where each block size is less than the size of FB. Next, each block is saved into a file structure as an I/O-conscious prefix-tree.

Disc Tree Location Table(DTLT) is an R-tree- [8]based structure that stores the knowledge of Prefix-Tree disc *node-Id* ranges associated with each prefix-tree Moreover, it makes sure that the DTLT is always maintained using only a bounded portion of *M.B*. It also keeps a hash table associated with the DTLT structure to easily identify nodes of the DTLT structure that are actually in memory and those that are in a disc-based file. Actually, if a node is not present in the hash table, this subunit assumes the node is stored on disc and subsequently loads the node in to memory and creates a corresponding entry in the hash table.

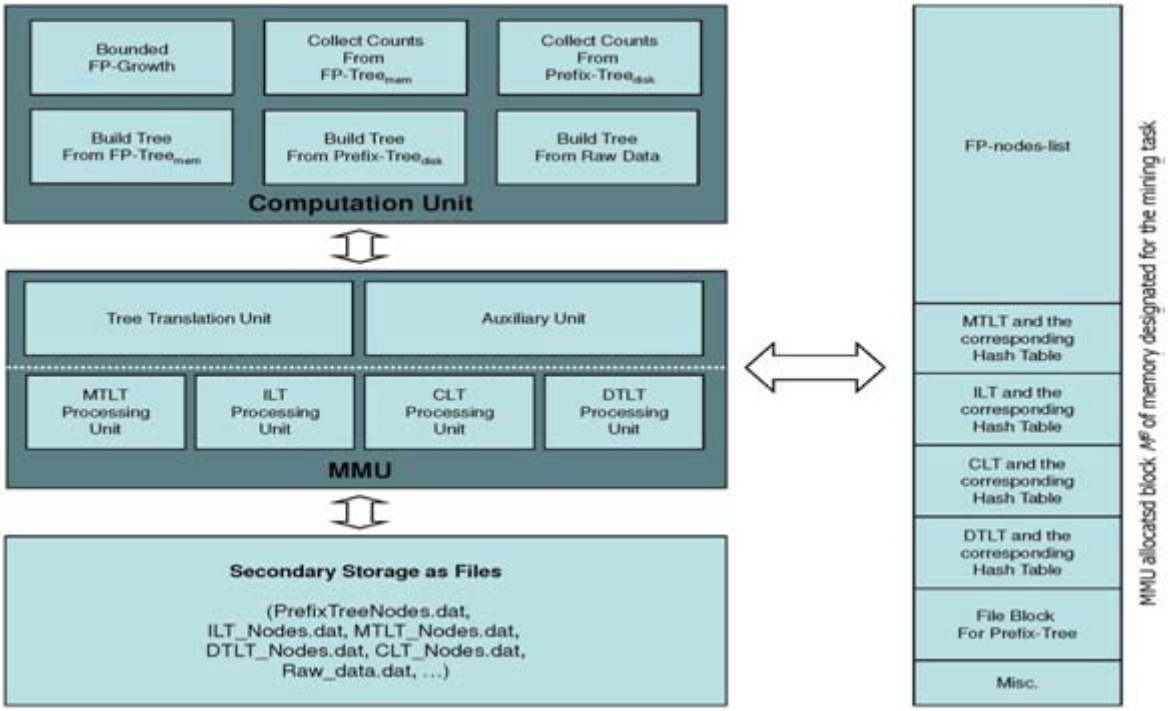

Figure: 4 -Architecture of Bounded and Adaptive

The ILT structure is also an R-tree based structure, which stores the *node-link* pointers of the prefix-trees.[11] Each ILT internal node contains dimensionality information as tree Id Range, item Id Range, and node Id Range; whereas, a leaf node holds the node Ids of a particular item in a particular prefix tree block, in addition to tree Id, item Id, and node Id Range. The ILT processing unit does all the manipulation of the ILT structure in a way similar to how the DTLT processing unit processed.[1]

MTLT is an R-tree-based structure which keeps track of the FP-trees that are currently present in memory. MTLT structure uses this id as dimensionality information. The leaf node of the MTLT structure contains a pointer to the root node of the particular FP-tree in memory and also a file pointer to the header table which if not used is saved on the disc.[11]

The CLT structure stores the count/frequency list of all the items in a particular prefix tree. Auxiliary unit is the subunit manipulates the data structures that are placed on the Misc and the (File Block)FB portion of *M.B*; M is the total main memory and B is the bounded portion of primary memory .It is mainly used by some of the subunits of the MMU and the computation unit.[1]

VI. RESULTS

The performance of the proposed frequent pattern mining model on data sets of different properties with different main memory usage constraints was studied. The proposed mining model was configured in such a way that about 80% of the bounded main memory was allocated for FP-Nodes-list, 5% for FB, 12% for data structures ILT, DTLT, MTLT, CLT, and their corresponding hash tables, and 3% for other Misc data structures.

For mining model, undoubtedly, the FP-Nodes-list needs the memory the most; so it gets around 80% of the whole available quota of primary memory. To avoid frequent page faults for our MMU, around 3% of the memory was assigned to each of the R-tree-based structures. Exact values of these parameters should be guided by the data analyst, and performance of the system may vary based on different parameter values. However, the above chosen parameter values seemed to work reasonably well for our experimental setup and a slight variation from the above specified values did not seem to make a great impact on the running time if majority of the memory is assigned to FP-Nodes-list.[1]

The comparative study of the proposed work with the existing methods has been presented. For the comparison of the techniques presented in the approach, the following performance metrics have been used such as time and the memory usage. The following screenshots represents the comparative study of the both approaches.

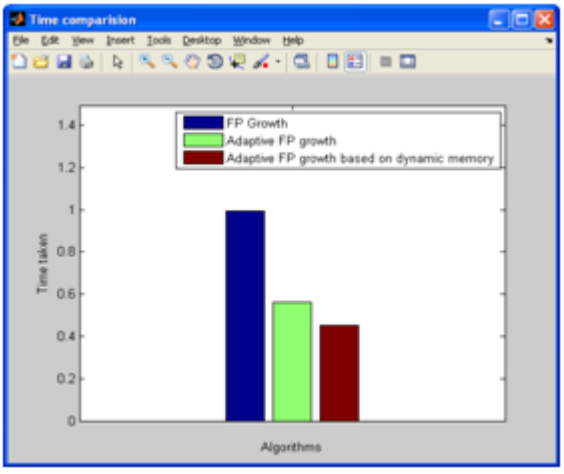

Figure5 Comparitive chart for time taken

The above presented Graph point out the comparative chart represents the time taken for the frequent pattern mining for the three approaches namely the FP-Growth, adaptive FP-Growth and the adaptive FP-Growth based on the dynamic memory allocation. From the above graph it is known that the time taken for the frequent pattern mining by the proposed technique is relatively lesser than the previous approach also by providing the dynamic memory allocation.

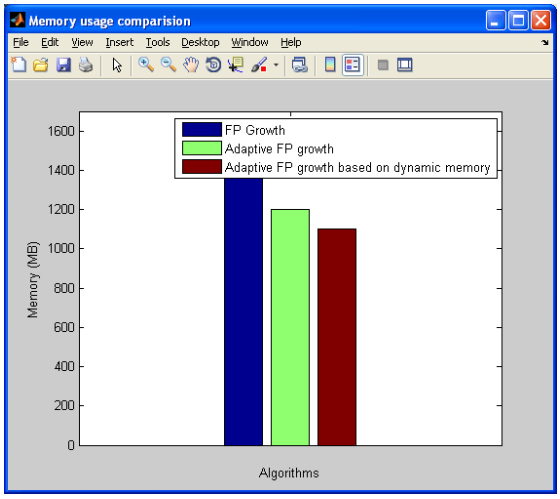

Figure: 5 Comparitive chart for memory usage

From the above graph it is known that the memory taken for the frequent pattern mining by the proposed technique is relatively lesser than the previous approach also by providing the dynamic memory allocation.

VII. CONCLUSION AND FUTUREWORK

In this work, an approach of frequent pattern mining based on the adaptive FP-Growth approach is presented by handling the memory allocation dynamically. Also the proposed approach have been compared with the present work in case of the performance metrics such as the time taken and the memory usage by the approaches for the frequent pattern mining. From the results it was found that the proposed work scale up the performance of the approach of frequent pattern mining from the large database and also able to have a dynamic memory management by developing a specialized memory management unit and by eradicating the concept of behavior of depending up on the virtual Memory Management. This method can be further used for research studies in order to make the approach more specific and to be handled for any range of the large database handling the different types of the data.

VIII. REFERENCES

[1]. Muhaimenul Adnan and RedaAlhajj," A Bounded and Adaptive Memory-BasedApproach to Mine Frequent Patterns From Very Large Databases", IEEE Transactions on Systems, Man, and Cybernetics—Part b: Cybernetics, Vol. 41, no. 1, February 2011.

[2]. Sotiris Kotsiantis, DimitrisKanellopoulos,"Association Rules Mining: A Recent Overview",GESTS International Transactions on Computer Science and Engineering, Vol.32 (1), 2006, pp. 71-82

[3]. R.C. Agarwal, C.C. Aggarwal, and V.V.V. Prasad."Depth first generationof long patterns". In Ramakrishnan et al. [32], pages 108–118.

[4]. R. Agrawal, T. Imielinski, and A. N. Swami."Mining association rules between sets of items in large databases". In ACM SIGMOD'93, pages 207–216, Washington, D.C., 1993.

[5]. C´elineH´ebert and Bruno Cr´emilleux,"Mining Frequent δ-Free Patterns in Large Databases", A. Hoffmann, H. Motoda, and T. Scheffer (Eds.): DS 2005, LNAI 3735, pp. 124–136, 2005_c Springer-Verlag Berlin Heidelberg 2005.

[6]. R.J. Bayardo. "Efficiently mining long patterns from databases".In*Proc. SIGMOD 1998*, pp. 85–93.

[7]. 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,8(1), 2004, pp.53-87.

[8]. W. Cheung and O.R. Za¨ıane. Incremental mining of frequent patterns without candidate generation or support constraint. In *Proc. IDEAS2003*, pp. 111–116.

[9]. Agrawal R., Imielinski, T., and Swami, A. 1993. "Mining association rules between sets of items in large databases". In Proc. of ACM-SIGMOD, 1993 (SIGMOD" 93), pp. 207–216

[10]. Carson Kai-Sang Leung*Quamrul I. Khan TariqulHoque,"CanTree: A Tree Structure forEfficient Incremental Mining of Frequent Patterns"Proceedings of the Fifth IEEE International Conference on Data Mining (ICDM'05)1550-4786/05 $20.00 © 2005 IEEE

[11]. A. Guttman, "R-trees: A dynamic index structure for spatial searching," in *Proc. ACM SIGMOD*, B. Yormark, Ed., 1984, pp. 47–57.

[12]. 7.Leung,C. K.-S., Q. I. Khan, and T. Hoque, (2005) "Cantree: A tree structure for efficientincremental mining of frequent patterns," in Proc. IEEE Int. Conf. Data Mining, Los Alamitos, CA,pp. 274–281.

[13]. Rácz.R, (2004) "nonordfp: An FP-growth variation without rebuilding the FP-tree," in Proc.FIMI.

[14]. Cheung,W. and O. R. Zaiane, (2003) "Incremental mining of frequent patterns without candidate generation or support constraint," in Proc. IEEE Int. Conf. Database Eng. Appl., Los Alamitos, CA,pp. 111–116.