International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 1 Issue 3, May - 2012

A new tree data structure to extract frequent pattern from transactional database

Hemali Shah
GEC,Modasa,
Gujarat, India

Prof. J.S.Dhobi
GEC, Modasa
Gujarat, India

Abstract

Frequent Pattern mining has attracted wide attention in both research and application area recently. In this paper, we propose a novel tree data structure to extract all frequent patterns from transactional database with single database scan. It is also supports Interactive and Incremental mining without rescan the original database. Experimental results show that our new tree data structure is efficient for frequent pattern mining, interactive, and incremental mining with single database scan. It extracts frequent patterns faster than most current one like CP-Tree.

Keyword: Data mining, frequent pattern, Association rule, Incremental mining, Interactive mining.

1. Introduction

Finding frequent patterns (or itemsets) plays an essential role in data mining and knowledge discovery techniques, such as association rules, classification, clustering, etc. The Apriori algorithm was the first attempt to mine association rules from a large dataset. It has been presented in [6] for the first time. Apriori-based, that is, they depend on a generate-and-test paradigm. To improve efficiency of the mining process, Han et al. [4, 5] proposed an alternative framework, namely a tree-based framework. The algorithm they proposed in this framework constructs an extended prefix-tree structure, called Frequent Pattern tree (*FP*-tree), to capture the content of the transaction database. Rather than employing the generate-and-test strategy of Apriori-based algorithms, such a tree based algorithm focuses on frequent pattern growth–which is a restricted test-only approach. This FP-Growth algorithm is useful for static frequent pattern mining. So, Koh and Shieh [7] proposed the AFPIM (Adjusting FP-Tree for Incremental Mining) algorithm, for incremental frequent pattern mining. Therefore, it requires two database scans. CATS tree [8] is a single-pass solution but it still suffers from complex tree construction process. The above two limitations are well-addressed in CanTree [3] that captures the complete information in a canonical order of items from database into a prefix-tree structure in order to facilitate it for incremental and interactive mining using FP-growth mining technique. Since the items in the CanTree are not stored in frequency-descending order, it usually yields poor compaction in tree size compared to FPtree. CP-tree (Compact Pattern tree)[1], that constructs a compact prefix-tree structure with one database scan and provides the same mining performance as the FP-growth technique by efficient tree restructuring process. CP-tree keeps all the items in the tree after restructuring phase.

In this paper, we introduce a new tree data structure that takes advantage of both **CP-tree**, that constructs a compact prefix-tree structure with one database scan and **FP-tree,** that keeps only frequent items in the tree.

2. Overview of Proposed data structure:

Construction of tree contains mainly two phases: 1) **Insertion phase** and 2) **Restructure phase**

First, In Insertion phase it scans transactions(s), inserts them into the tree according to item appearance order. It also maintains I-list and updates frequency count of respective items in I-list when inserts transaction into the tree. So, after insertion of all transactions, tree data structure has total support count of all items in database in its I-list.

Second, In Restructure phase it rearranges the I-list according to frequency descending order of items and only kept only frequent item (whose frequency count is greater than user specified minimum support) and restructures the tree nodes according to this newly rearranged I-list. In tree restructuring it use Branch Sorting Method (BSM) like used in CP-Tree[2] restructuring phase, but keep only frequent items in tree after restructure phase. In

this method if path is not sorted according to new I-list order, it is removed from the tree, deleted non-frequent items, sorted according to new I-list order into a temporary array and then again inserted into tree in order.

After construction of tree it mines frequent pattern using FP-Growth algorithm like in case of all other tree structures.

Let's see one by one step for my proposed data structure for given database.

Figure 1(see pg 4) shows I-list and tree after Insertion phase for database DB given in following table, it is same as Insertion phase in CP-Tree. For simplicity of figures we do not show the node traverse pointers in tree, however, they are maintained in a fashion like FP-tree does.

Now, let's see Restructure phase step by step. Input of Restructure phase is tree and I-list after insertion of all transaction of database. In our example input of restructuring phase is tree shown in Figure 1. Figure 2(see pg 4,5) shows overall processing of Restructure phase by Branch Sorting Method. In this I-list items are not arranged in frequency-descending order. Therefore, to restructure the tree using the BSM in such order, I is sorted first to generate a new I-list, Isort as shown in Fig. 2a. Tree restructuring, then, starts with the first branch, which is, say the left most branch from the root of the tree in Fig. 1. Since the first path {f:1?a:1?c:1?d:1?g:1?i:1?m:1?p:1} of the branch is an unsorted path, it is removed from the tree (Fig. 2b), sorted using the into a temporary array to the order {f:1?c:1?a:1?m:1?p:1} by satisfying Isort order (Fig. 2c), and then again inserted into T in sorted order. Fig. 2d shows the tree structure after sorting the first path. Fig 2e shows the final tree after Restructure phase. Fig 2f shows CP-Tree for given database. From figure 2e and 2f we can say that our proposed data structure has compact tree structure than CP-Tree (Number of nodes in our proposed tree data structure are less than CP-Tree). We include only frequent items in tree after restructuring phase.

Interactive mining:

Our proposed tree structure also supports Interactive mining. In interactive mining user specified minimum support can be changed for the same database. In this case, we can save tree after insertion of all transactions. Then we can restructure tree according to user specified minimum support. So, there is no need to rescan the database for interactive mining in this tree structure like FP-Tree. We have support of all the items in our I-list. So, we can keep only frequent items according to different minimum support without the need of rescan the database.

Incremental mining:

Our proposed tree structure also supports Incremental mining. In incremental mining transaction(s) are added and deleted in original database. In this case, we can save tree after insertion of all transactions in original database. When new transaction(s) are added or some transaction(s) are deleted, at that time we can add new transaction(s) and delete those transaction(s) in that saved tree easily and restructure the tree without the need of rescan the original database like FP-Tree.

3. Experiment Results:

We have downloaded following datasets from http://fimi.ua.ac.be/data/ website (Frequent Itemset Mining Implementations).

Programs are written in Java (jdk 6) and run on Windows 7 operating system on a 2.13 GHz machine with 4 GB of main memory.

Table-1 and Table-2 shows total execution time required to extract all frequent patterns for mushroom and chess dataset respectively by my proposed data structure and CP-Tree for different support and also shows total frequent patterns for respective support.

Table-1:Execution time for Mushroom dataset for different suppot

Support	Total frequent pattern	CP-Tree (ms)	My proposed tree structure(ms)
0.35	1188	24710	1780
0.30	2734	24790	1870
0.25	5544	24842	2020
0.20	53662	25850	2550
0.15	98574	26068	2642

Table-2:Execution time for Chess dataset for different support

Support	Total frequent pattern	CP-Tree (ms)	My proposed tree structure(ms)
0.90	627	13670	1860
0.80	8281	13752	2260
0.70	48968	13982	2780
0.60	2,55,984	14642	3200

Chart-1 and chart-2 shows performance study of CP-Tree and my proposed tree structure for mushroom dataset and chess dataset respectivel.

Chart-1: Mushroom dataset

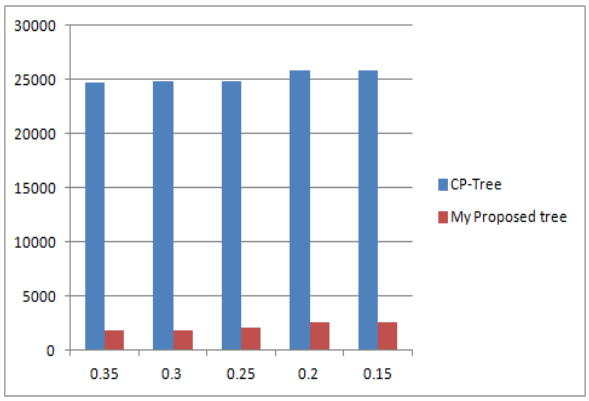
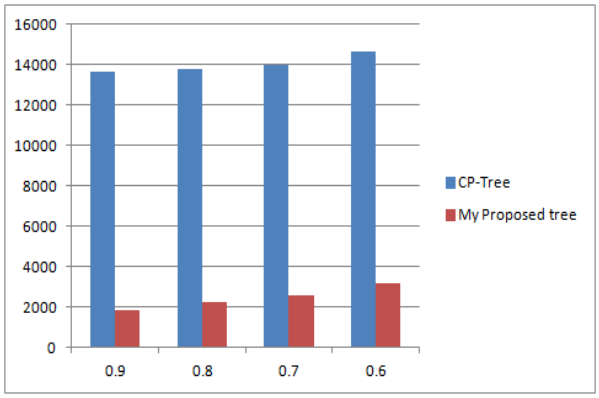


Chart-2: Chess dataset



4. Conclusion:

Based on analysis of experiment results following conclusions is made:

1) My new data structure reduces overall processing time to extract frequent pattern from database compared to CP-Tree.
2) It also supports Interactive mining like CanTree and CP-Tree, means if user specified minimum support is changed then also it can extract frequent patterns without the need to rescan the database.
3) It also supports Incremental mining like CanTree and CP-Tree, means later if transaction(s) are added or old transaction(s) are deleted then also it can extract frequent patterns without the need to rescan the original database.

5. References:

[1] Syed Khairuzzaman Tanbeer, Chowdhury Farhan Ahmed, Byeong-Soo Jeong, and Young-Koo Lee, "CP-Tree: A Tree Structure for Single-Pass Frequent Pattern Mining." In: Springer-Verlag Berlin Heidelberg 2008,pp 1-6

[2] Syed Khairuzzaman Tanbeer, Chowdhury Farhan Ahmed, Byeong-Soo Jeong , Young-Koo Lee, "Efficient single-pass frequent pattern mining using a prefix-tree" In: Information Sciences 179 (2008) pp 559–583

[3] Leung, C.K., Khan, Q.I., Li, Z., Hoque, T., "CanTree: A Canonical-Order Tree for Incremental Frequent-Pattern Mining." Knowledge and Information Systems 11(3),(2007) pp 287-311

[4] Han J, Pei J, Yin Y, "Mining frequent patterns without candidate generation." In: Chen W, Naughton JF, Bernstein PA (eds) Proceedings of the SIGMOD 2000. ACM Press, New York, pp 1–12

[5] Han J, Pei J, Yin Y, Mao R, Mining frequent patterns without candidate generation: a frequent-pattern tree approach." Data Min Knowledge Dis 8(1) (2004) pp 53–87

[6] Agrawal, Rakesh; Srikant, Ramakrishnan: "Fast Algorithms for Mining Association Rules." Proc. 20th Int. Conf. Very Large Data Bases, VLDB, 1994.

[7] Koh J-L, Shieh S-F (2004) "An efficient approach for maintaining association rules based on adjusting FP-tree structures." In: Lee Y-J, Li J, Whang K-Y, Lee D (eds) Proceedings of the DASFAA 2004. Springer-Verlag, Berlin Heidelberg New York, pp 417–424

[8] Cheung, W., Zaïane, O.R., "Incremental Mining of Frequent Patterns without Candidate Generation or Support Constraint." In: Seventh International Database Engineering and Applications Symposium (IDEAS) (2003)

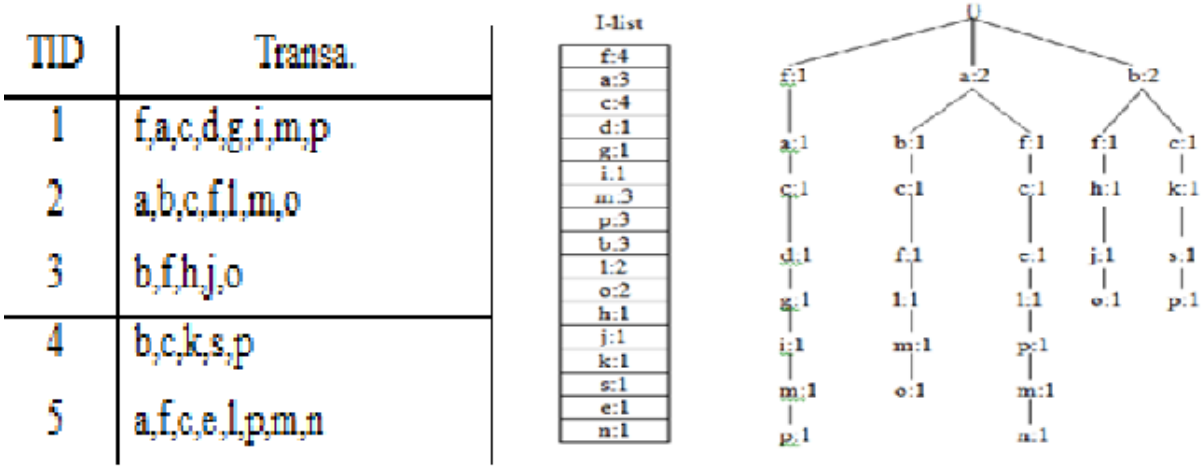
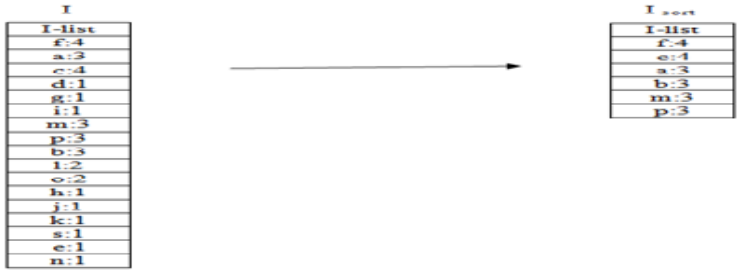
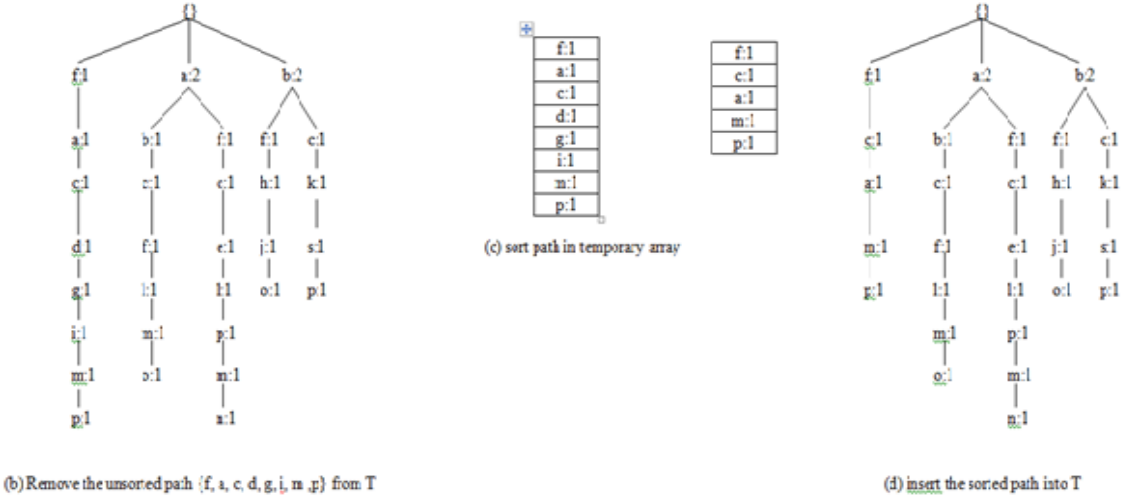
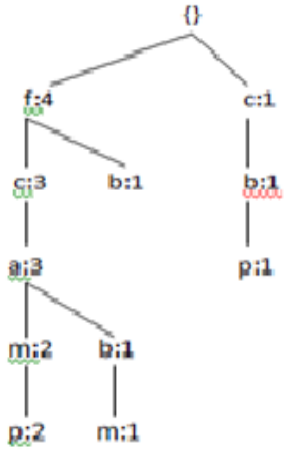


Figure 1: Tree and I-list after Insertion phase for Database DB (for my proposed data structure)



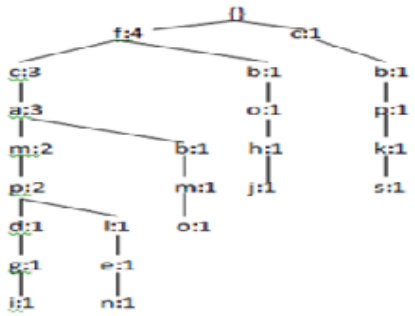
(a) Restructure I to construct I_{sort}





(e) Final tree after Restructure phase in My proposed data structure

f:4
c:4
a:3
b:3
m:3
p:3
l:2
o:2
d:1
e:1
g:1
h:1
i:1
j:1
k:1
n:1
s:1



(f) Tree for given Database for CP-tree.

Figure 2: Total processing of Restructure phase of my proposed data structure