Efficient Tree-based Discovery of Frequent Itemsets

Byung Joon Park

Department of Computer Science,
Kwangwoon University, Seoul, Korea
bjpark@kw.ac.kr

Abstract

Various types of data structures and algorithms have been proposed to extract frequently occurring patterns from a given data set. In particular, several tree structures have been devised to represent the input data set for efficient pattern discovery. One of the fastest frequent pattern mining algorithms known to date is the CATS algorithm, which can efficiently represent the whole data set and allow mining with a single scan over the database. In this paper, we propose an efficient tree structure and its associated algorithm that provides a considerable performance improvement over CATS in terms of memory usage and processing time. We have demonstrated the effectiveness of our algorithm and performance improvement over the existing approach by a series of experiments.

Keywords: f*requent itemset, pattern discovery, tree structure*

1. Introduction

As various applications such as text, transaction, and image processing generate a large volume of data, discovery of frequent patterns from a huge collection of data has been a topic of active research in the information processing community. Various types of data structures and algorithms have been proposed to discover sets of items frequently occurring together from a given data set [1, 2]. In particular, several tree structures have been devised to represent the input data set for efficient pattern discovery. Most of these tree structures allow efficient incremental mining with a single pass over the database as well as efficient insertion or deletion of transactions at any time [3]. One of the fastest frequent pattern mining algorithms known to date is the CATS algorithm, which can efficiently represent the whole data set and allow mining with a single scan over the database [4]. The CATS algorithms enable frequent pattern mining with different support values without rebuilding the tree structure. This paper describes our work on improvement over the original CATS approach in terms of memory usage and processing time. The proposed tree structure allows insertion or deletion of transactions at any time like CATS, but usually results in more condensed types of tree enabling a more efficient pattern discovery process.

This paper is organized as follows. Section 2 describes some related work and the background knowledge on CATS. In Section 3, we present the proposed tree structure, its construction algorithm, and comparison with CATS FELINE approach. In Section 4, we discuss experimental results and performance evaluation of our approach. And finally Section 5 draws a conclusion..

2. Related Work

FP-tree [5] is one of the earliest tree structure-based approaches that do not use candidate generation adopted by Apriori. Given a set of transactional data, it first

constructs a tree structure in which all items are arranged in descending order of their frequency. After the construction of FP-tree, the frequent patterns can be mined using an iterative algorithm, FP-growth, which looks up the header table and selects the items with the minimum support. Then the infrequent items are removed from the prefix-path of an existing node and the remaining items are regarded as the frequent item-sets of the specified item.

Can-Tree [2] constructs a tree structure similar to FP-tree, but does not require a rescan of the original database when it is updated. CAN-tree contains all the transactions in some canonical order, hence the ordering of the nodes in CAN tree will be unaffected by any frequency changes due to incremental updates like insertion, deletion, and modification of the transactions.

CATS-tree [3] is a tree-based approach our proposed algorithm is directly related to. A CATS-tree is a prefix tree and contains all elements of FP-Tree including the header, the item links etc. Paths from the root to the leaves in a CATS-Tree represent sets of transactions. Once a CATS-tree has been constructed from a database, it enables frequent pattern mining with different supports without the need to rebuild the tree structure. In the mining process with a CATS-tree, the CATS-FELINE algorithm builds a conditional condensed CATS-tree for each frequent item *p* by gathering all transactions that contain *p*. A conditional condensed CATS-tree is one in which all infrequent items are removed and is different from a conditional FP-tree. In order to ensure that all frequent patterns are captured by CATS-FELINE, it has to traverse both up and down the CATS-tree.

3. The Construction Algorithm for a Conditional Condensed Tree

Now we describe our proposed tree structure and its related algorithm. It is the process of constructing a conditional condensed CATS-tree that is the main focus of our work. Like CATS-FELINE, the overall mining process proceeds in three phases:

Step 1: Convert the CATS tree generated from a database scan into a condensed tree with nodes having the frequency count less than the minimum support removed.

Step 2: Construct conditional condensed CATS-trees (also known as alpha-trees) for items in the header table with frequency counts greater than the minimum support.

Step 3: For each alpha-tree generated in step 2, item sets with at least minimum support are mined.

Our algorithm differs from CATS-FELINE in step 2. Instead of recursively constructing alpha trees for each frequent item set, our algorithm generates a single conditional condensed tree for each item using a pre-order traversal of the original CATS-tree. To illustrate the basic idea behind the algorithm, we will use the database shown in Table 1 as an example (same as the sample database in [3]) and the original CATS-tree constructed from a database scan and its condensed one will look like the following (assuming minimum support of 3) [3]:

Table 1. Sample Database

TID	Original Transactions
1	F, A, C, D, G, I, M, P
2	A, B, C, F, L, M, O
3	B, F, H, J, O
4	B, C, K, S, P
5	A, F, C, E, L, P, M, N

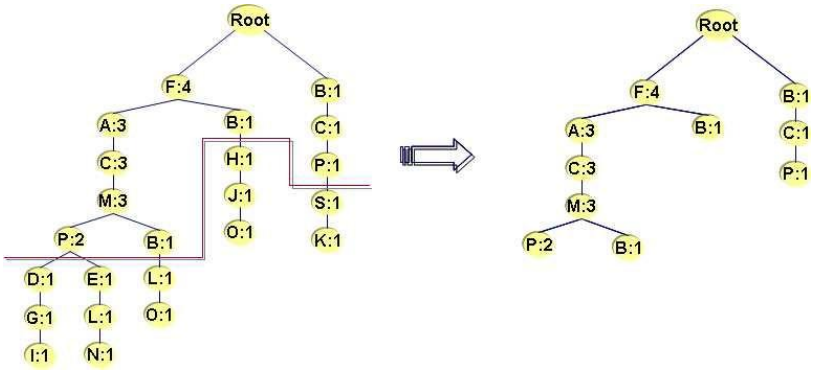

Figure 1. CATS-tree and its Condensed One

This condensed tree, a header table containing all the frequency counts for each item, and the required minimum support will be the actual input to our algorithm called FPM(Frequent Patterns Merge). Given the above condensed tree, FPM starts building an alpha tree for each frequent item (i.e. alpha item) as follows:

Figure 2. Initial Round of Constructing Alpha Tree for c

Since C is an item with the highest frequency in the header table, FPM constructs an alpha tree for c first. By traversing the leftmost path of the tree of Figure 1(b) in preorder, it will construct a partial tree Figure 2 consisting of a single path C-F-A-M (all items with support at least 3). Note that the order of nodes and the frequency count of some node have been slightly changed from Figure 1(b) to Figure 2. The node for item c has been moved to the root because it is the current alpha item under consideration. And the frequency of node F has been changed to 3 from 4 because the branch F-B in Figure 1(b) does not contain item C and thus the frequency of F has been decremented by 1.

After C-F-A-M has been added to the current alpha-tree, the node for 'P:2' will be encountered in the preorder traversal. In this case, P is not frequent (with count 2) and there is no node for P in the current alpha tree. Then, a node is created for 'P:2' and will be inserted to the current alpha tree as a child of the root. This is the major difference between the CATS-FELINE and FPM. FPM attempts to reduce the number of nodes in the alpha tree by condensing even infrequent items. The same process applies to node 'B:1' and Figure 3

shows the resulting alpha tree after the left subtree of the original condensed tree has been traversed.

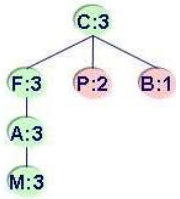

Figure 3. Next Round of Constructing Alpha Tree for c

Now, the right subtree of the input tree is to be traversed. It has one node for C and thus the root count of the current alpha tree should be incremented by 1, making it 4. And also the counts of node B and P should also be incremented by 1 respectively. Figure 4 shows the final alpha tree constructed for item C:

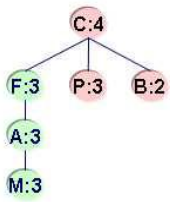

Figure 4. Final Round of Constructing Alpha Tree for c

The following summarizes the procedure for generating the alpha tree for an alpha item given a CATS-tree and a required minimum support:

1. Prune the condensed tree further by removing from it all paths that do not contain the alpha item and update the node's frequency counts accordingly.
2. Let the alpha tree be a single node containing the alpha item and count 0.
3. If the root item of the condensed tree is frequent, either insert the root into the current alpha tree at proper position (if not present already) or update the count of the existing node by adding its new count.
4. Otherwise, either insert the root into the current alpha tree as a child of the root (if not present already) or update the count of the existing node by adding its new count.
5. Next, for each subtree of the condensed tree, repeat steps 3 and 4 recursively.

4. Experimental Results

Given the same database used in Section 3, alpha trees constructed by CATS-FELINE and ours will be different as shown in Figure 5. The difference is due to the way how the infrequent items are dealt with. As shown in Figure 4, CATS-FELINE keeps many separate nodes(P:2 and P:1 in Figure 4) for infrequent items such as P although they share the same alpha node. Hence, it needs more memory space for storing infrequent items. However, FPM results in a more condensed alpha tree in most cases since separate infrequent items are collapsed into single child nodes of the alpha item(root).

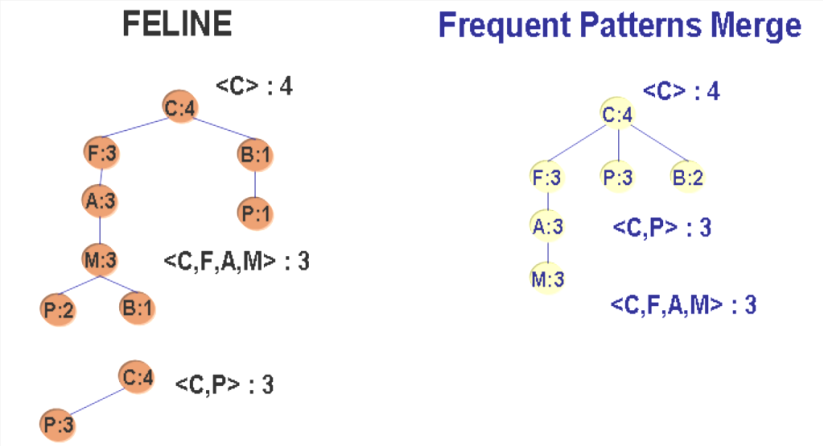

Figure 5. Different Types of Alpha Tree

We conducted a series of experiments to compare the processing time with different sizes of transactions for each of algorithms. In the experiments, we set the minimum support to 3. Figures 6 shows the relative performance of FPM compared to CATS-FELINE in mining time.

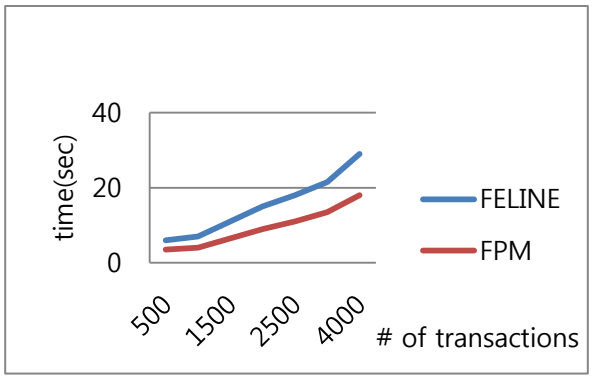

Figure 6: mining time comparison

5. Conclusion

In this paper, we described an efficient tree structure for representing a conditional condensed tree (alpha tree) for a frequent item in transactions. The proposed tree structure allows insertion or deletion of transactions at any time like CATS, but usually results in more condensed types of tree enabling a more efficient pattern discovery process. We demonstrated the performance improvement of FPM over CATS-FELINE by conducting a series of experiments with various minimum support values and different sizes of databases. A considerable performance improvement over CATS in terms of memory usage and processing time has been achieved by our proposed tree structure..

Acknowledgements

The present research has been supported by the Research Grant of Kwangwoon University (No. 60012008199).

References

[1] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules", Proc. of the 20th Very Large Data Bases International Conference, **(1994)**, pp. 487-499, Santiago, Chile.

[2] C. K.-S. Leung, Q. Khan and T. Hoque, "Cantree: A tree structure for efficient incremental mining of frequent patterns", Proc. of 5th IEEE International Conference on Data Mining, **(2005)**, pp.274–281, Los Alamitos, CA.

[3] K. Sasireka, G. Kiruthiga and K. Raja, "A Survey about Various Data Structures for Mining Frequent Patterns from Large Databases", International Journal of Research and Reviews in Information Sciences, **(2011)** Vol. 1, No. 3, pp. 74-76.

[4] W. Cheung, and O. Zaiane, "Incremental Mining of Frequent Patterns Without Candidate Generation or Support Constraint", Proc. of 7th International Database Engineering and Applications Symposium, **(2003)**, pp. 111–116, Los Alamitos, CA.

[5] 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, **(2004)**, Vol. 8, No.1, pp. 53–87.

Authors

Byung Joon Park

He is currently an Associate Professor at the Computer Science Department, Kwangwoon University in Seoul, Korea. His research interests include artificial intelligence and web/smart applications, especially machine learning and data/web mining. He received his Bachelors degree in Computer Engineering from Seoul National University, Korea, Masters Degree in Computer Science from University of Minnesota, Minneapolis, USA, and Ph.D. in Computer Science from University of Illinois at Urbana-Champaign, USA, respectively.