A Prefix Tree-Based Algorithm for Efficient Discovery of Frequent Patterns

Byung Joon Park, Sung Hee Kim

Department of Computer Science,
Kwangwoon University
Seoul, Korea
{bjpark,nana80}@kw.ac.kr

Abstract. Mining frequent patterns from a large database often involves construction of intermediate structures such as trees for the purpose of efficiency. Various researchers have proposed many types of trees such as CATS-tree and Can-tree. However, these tree-based mining algorithms require a large amount of computing time during the tree construction phase or need generation of extra trees. In this paper, we propose a prefix tree-based mining algorithm which does not require extra tree construction and can reduce the number of potential data exchanges during the intermediate tree contruction. We demonstrate the effectiveness of our algorithm and performance improvement over the existing approach by a series of experiments.

Keywords: frequent pattern discovery, prefix tree structure, incremental mining

1 Introduction

Discovery of frequent patterns from a large volume of data has been a topic of active research in the information processing community. To efficiently discover sets of items frequently occurring together from a given data set, various types of data structures and algorithms have been proposed[1][2]. In particular, various 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 using a tree structure called CATS-tree[4]. The CATS algorithm enables 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 processing time. The proposed prefix-tree structure allows insertion or deletion of transactions at any time like CATS, but can reduce the number of potential data exchanges during the intermediate tree contruction.

This paper is organized as follows: In Section 2, we present the proposed prefix-tree structure and its construction algorithm. In Section 3, we discuss experimental

results and performance evaluation of our approach. And finally Section 4 draws a conclusion.

2 A Prefix Tree for a Compact Representation of Itemsets

Now we describe our proposed prefix-tree structure and its related algorithm. Our algorithm first scans the database of itemsets to obtain the global frequency count of each item and then sorts the items of each itemset in descending order of their frequency counts. This process of reordering items can avoid unnecessary swapping of nodes in the tree construction phase needed to keep more frequent nodes higher in each path of the tree. It also makes the final tree less sensitive to the order of itemsets in the database as well as to the order of items in each itemset. Once a prefix tree has been constructed from the input database, our mining algorithm does not need extra tree construction for each frequent item and it can proceed in buttom-up fashion unlike CATS-FELINE which has to mine the whole tree both in upwards and downwards fashion. The entire mining process of our approach proceeds in the following steps:

Step 1: Scan the database scan to obtain the frequency count of each item and sort the items in each itemset in descending order of their frequency counts.

Step 2: Construct a single prefix tree for all sorted itemsets in the database. Each node of this tree has both of the global frequency count and the local one for each item.

Step 3: From the prefix-tree generated in step 2, item sets with at least minimum support are mined.

To illustrate the basic idea behind algorithm, we will use the following database as an example (same as the sample database in [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

Given the above database, the original CATS-tree constructed from a database scan and its condensed one will look like the following (assuming minimum support of 3):

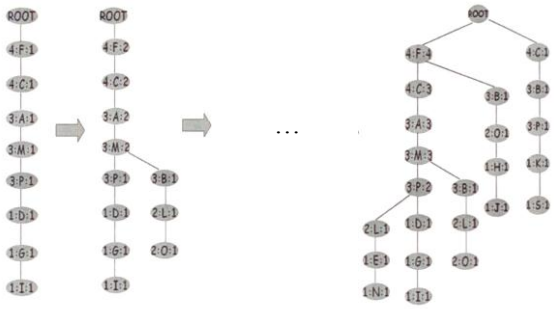


Figure 1: Construction of a prefix tree from the input database

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 mining algorithm. To illustrate how the actual mining algorithm works, suppose that we want to find the frequent patterns containing 'P'. For frequent patterns containing 'P', our algorithm first finds {F,C,A,M,P} and {C,B,P} in the tree using the header table. These two itemsets are the conditional pattern base for item P. By taking the intersection of these two sets, it will find {C,P} as frequent patterns.

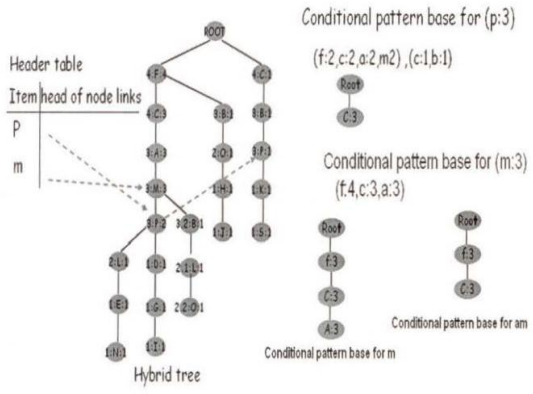
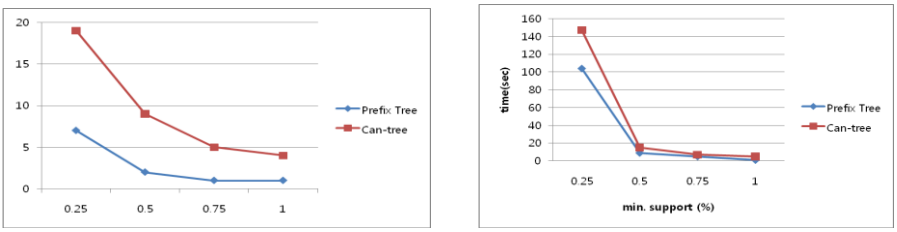


Figure 2: the mining results for items 'P'

3 Experimental Results

In order to demonstrate the performance improvement of our algorithm over existing approaches, we have conducted a series of experiments with various minimum support values. We compared our algorithm with CAN-tree which showed a better performance over CATS-FELINE[3]. We assumed that each transaction could contain up to 26 items from A through Z and that there are no duplicated items in any transaction. For experiments, we have implemented the algorithms in C++ and used a

PC with Core2 Quad CPU 2.5GHz, 4G RAM. Figures 3(a) and 3(b) both show Prefix-tree's considerable improvement over CAN-tree in mining time.



(a) (b)

Figure 3: mining time comparison (80,000 transactions)

4 Conclusion

In this paper, we described an efficient tree structure for representing all items in transactions. The proposed tree structure allows insertion or deletion of transactions at any time like CATS, but usually requires less number of data exchanges in the tree construction and does not need construction of extra trees called alpha trees, thus enabling a more efficient pattern discovery. We demonstrated the performance improvement of Prefix-tree 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 and CAN-tree in terms of memory usage and processing time has been achieved by our proposed tree structure.

References

1. Agrawal, R. and Srikant, R.: Fast algorithms for mining association rules, In: Proc. of the 20th Very Large Data Bases International Conference, pp.487--499 (1994).
2. Leung, C. K.-S., Khan, Q. I., Hoque, T.: Cantree: A tree structure for efficient incremental mining of frequent patterns, In: Proc. IEEE International Conference on Data Mining, pp. 274–281, Los Alamitos, CA (2005).
3. Sasireka, K., Kiruthiga, G., Raja, K.: A Survey about Various Data Structures for Mining Frequent Patterns from Large Databases, International Journal of Research and Reviews in Information Sciences, Vol. 1, No. 3, pp. 74--76 (2011).
4. Cheung, W., Zaiane, O.: Incremental Mining of Frequent Patterns Without Candidate Generation or Support Constraint, In: Proc of 7th International Database Engineering and Applications Symposium, Los Alamitos, CA, pp. 111–116 (2003).
5. Han, J., Pei, J., Yin, Y., Mao, R.: Mining frequent patterns without candidate generation: a frequent-pattern tree approach, Data Mining and Knowledge Discovery, Vol. 8, No.1, pp. 53–87 (2004).