Batch Processing for Incremental FP-tree Construction

Shashikumar G. Totad
Department of CSE, GMRIT,
Rajam, Srikakulam District
AndraPradesh, India.

Geeta R. B.
Department of IT, GMRIT,
Rajam, Srikakulam District
AndraPradesh, India.

PVGD Prasad Reddy
Department of CS & SE,
Andhra University, Visakhapattnam
AndraPradesh, India.

ABSTRACT
Frequent Patterns are very important in knowledge discovery and data mining process such as mining of association rules, correlations etc. Prefix-tree based approach is one of the contemporary approaches for mining frequent patterns. FP-tree is a compact representation of transaction database that contains frequency information of all relevant Frequent Patterns (FP) in a dataset. Since the introduction of FP-growth algorithm for FP-tree construction, three major algorithms have been proposed, namely AFPIM, CATS tree, and CanTree, that have adopted FP-tree for incremental mining of frequent patterns. All of the three methods perform incremental mining by processing one transaction of the incremental database at a time and updating it to the FP-tree of the initial (original) database. Here in this paper we propose a novel method to take advantage of FP-tree representation of incremental transaction database for incremental mining. We propose "Batch Incremental Tree (BIT)" algorithm to merge two small consecutive duration FP-trees to obtain a FP-tree that is equivalent of FP-tree obtained when the entire database is processed at once from the beginning of the first duration to the end of the second duration. For large databases, our experimental results show significant reduction in runtime of the BIT algorithm compared to the runtime of sequential incremental algorithms.

General Terms
Data mining, FP-tree, Prefix-tree Frequent Patterns, Incremental mining.

Keywords
Batch Incremental Mining, Batch Incremental tree, Sequential Incremental Mining, minSup.

1. INTRODUCTION
Large databases, some times distributed over several remote locations, are becoming more common in the contemporary *Global Economy* scenario. The local databases which were initially small, have grown, growing continually and getting distributed to several remote sites as a result of globalization. Many of the conventional data mining algorithms are ineffective and inefficient for handling large and growing data sets [1] [2]. Hence, the scalable and incremental data mining has become an active area of research with many challenging problems. The large set of evolving and distributed data can be handled efficiently by *Incremental Data mining.* Incremental data mining algorithms perform knowledge updating incrementally to amend and strengthen what was previously discovered [5] [7] [12]. Incremental data mining algorithms incorporate database updates without having to mine the entire dataset again.

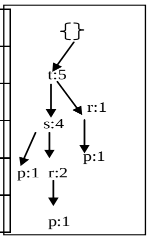
Frequent pattern is a pattern of items or events that appear frequently in a data set. Frequent patterns are very important in knowledge discovery and data mining process, such as mining of association rules, correlations etc. Since the introduction of the concept of frequent patterns in 1993, by R. Agrawal et al. [3], there have been many considerable studies[2] [4] [6] proposing different approaches for discovering various kinds of frequent patterns and their applications. Prefix-tree-based approach is one of the contemporary approaches for mining frequent patterns. A pattern P is said to be frequent in a given data set D if its support count sup(P, D) is greater than or equal to a predefined threshold called ***minSup***. Given a data set D and a support threshold m, the collection of all frequent item sets in D, is F(m, D) and is called "*space of frequent patterns*".

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

(a)

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

(b)



(c)

Figure 1. a) Initial Dataset b) Projected Dataset with min-threshold= 50% c) FP-tree

The prefix-tree compactly represents the transactions of a data set. Prefix-tree enables fast computation of support counts of all the frequent patterns of a dataset. Frequent patterns can be generated by traversing the prefix-tree, avoiding multiple scanning of the dataset. The *"Frequent-Pattern"* tree (FP-tree) is a prefix-tree, first proposed in 2000 by Han et al., in ACM-SIGMOD international conference[13] and later published in 2004[8]. FP-Tree is a compact representation of transaction database that contains frequency information of all relevant patterns in a dataset. To construct a FP-Tree for a given dataset, first, the data set is transformed into "*projected dataset*". The projected data set contains only the frequent items (with support count>min-threshold) and each transaction is sorted in the descending order of their support count. The transactions in projected dataset are added to prefix-tree one by one. The Figure1 shows the dataset, projected data set and the corresponding FP-tree constructed for the given dataset.

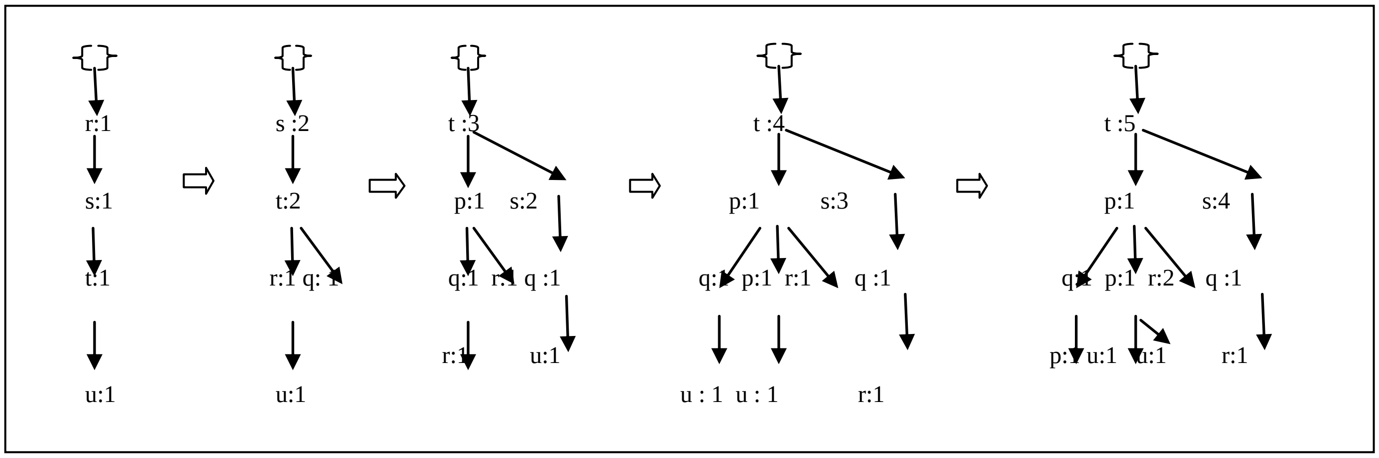


Figure 2. Step wise construction of CATS tree while processing each transaction

2. RELATED WORK

Han et al. proposed FP-growth algorithm [8] [13] 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 Aprori algorithm [3] that makes multiple scans over the dataset. Since the introduction of FP-growth algorithm three major algorithms have been proposed, namely AFPIM, CATS tree, and CanTree that have adopted FP-tree for incremental mining of frequent patterns.

AFPIM: Koh and Shieh proposed "*Adjusting FP-Tree for Incremental Mining*" (AFPIM) algorithm [9].This algorithm updates previously constructed FP-tree that contains frequent items based on user specified minimum support threshold *minSup*, by scanning only the incremental part of the dataset. As items are arranged in descending order of support count based on original dataset, AFPIM re-sorts the items according to new values of support count based on incremental dataset through bubble-sort. There are two major drawbacks of AFPIM: First, computational expensiveness of sorting process. Second, when new frequent patterns emerge, as a result of scanning of incremental dataset, AFPIM has to construct a new FP-Tree.

CATS Tree: CATS tree (Compressed and Arranged Transaction Sequence Tree) [10] addresses the limitations of AFPIM algorithm. Unlike AFPIM, the CATS tree considers all the items in the transactions for representation into tree, regardless of whether items are frequent or not. This allows CATS tree to represent even new emerging frequent patterns from incremental dataset. CATS arranges the nodes based on their local support count, which helps to achieve high compactness of the tree. For incremental mining CATS tree updates the existing tree by considering the transactions of the incremental dataset one by one and merging them with existing tree branches. Figure 2 shows how 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 are locally sorted.

CanTree: CanTree (Canonical-order Tree) is proposed by Leung et al. [11]. Construction of CanTree is very much similar to CATS tree except that, in CanTree items are arranged according to some canonical order. 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. Since the canonical order is fixed and not based on the support count, CanTree allows easy insertion of nodes. Unlike the CATS Tree, transaction insertions in CanTree require no extensive searching of mergeable paths. CanTree too has some limitations. It generates compact tree if and only if majority of the transactions contain common pattern-base in canonical order. It generates skewed tree with too many branches and hence with too many nodes, otherwise. Further, though the CanTree takes less time for tree construction it requires more memory and more time for extracting frequent patterns from the generated CanTree.

All of the three incremental prefix-tree based algorithms discussed above perform sequential incremental mining. That is, for incremental mining they consider one transaction of the incremental dataset at a time. However, in real scenario it is required to perform periodical mining of transaction databases for frequent pattern generation. The above discussed algorithms fail to take advantage of this periodical mining of frequent patterns. Supposing two data analysis are available for the first and second quarter of a year, in the form of FP-trees. And supposing it is required to obtain FP-tree for the first eight months of a year. All of the above discussed methods consider the FP-tree for the first quarter and perform incremental mining by processing one transaction of the second quarter database at a time. These methods do not take the advantage of the FP-tree of the second quarter that is readily available.

Here in this paper we propose a novel method to take advantage of such previously obtained periodical FP-tree, i.e., FP-tree representation of incremental transaction database, for incremental mining. We propose an "Batch Incremental Tree (BIT)" algorithm to merge the small consecutive duration FP-tree to obtain a FP-tree that is equivalent of FP-tree obtained when the entire database is processed at once from the beginning of the first duration to the end of the second duration.

In this section we discuss about working of the BIT algorithm for incremental mining of frequent patterns. BIT algorithm takes FP-tree of the two periodic datasets. It then reads the itemsets of one of the FP-tree (T1) one by one along with their frequency counts and searches for the mergeable prefix path of the other FP-tree (T2). It then merges the itemset of T1 with the mergeable prefix by updating frequency count of the items and inserting remaining non-prefix items(if any) by extending the tree branch after the last matching prefix item of the mergeable pattern. The algorithm given below precisely tells the steps involved in batch incremental processing.

3. BATCH INCREMENTAL TREE (BIT) ALGORITHM

ALGORITHM **BatchIncrementalTree(FP-tree T1,FP-tree T2)**

```
1.  Get itemsets from T2 by considering each of the leaves one by
     one.
2.  FP-tree T= T1
3.  For each itemset i obtained from T2 do the following steps, up
     to 18
4.  { Read the next itemset i of T2.
5.    Get the next  item nk  to compare, from T  // Initially 1st
                                                  //  child of root of T
6.    For each item  j  in the itemset  i  do the following steps ,
      up to 18
7.      if item nk is equal to item j  then
8.        if nk represents leaf node then
9.        { Update node represented by nk.
10.         Get the remaining items from the itemset i  and
            add each item as descendants of nk
            one below the other.
11.       }
12.       else   // if nk is not leaf node
13.       {  Update node represented by nk.
14.          nk = first child of nk.
15.       }
16.       else    // if item  nk is not equal to item j
17.          if nk has any more child then nk = next child of nk.
18.         else   Get the remaining items from the itemset i  and
                   add each item as descendants of nk
                   one below the other.
19. }
20. Return T.
21.
```

4. TIME COMPLEXITY ANALYSIS

For incremental data mining, CanTree reads the itemsets (transactions) of incremental database (D_2) one at a time, and upends each itemset to the FP-tree (T_1) of the original database (D_1), whereas the BIT algorithm gets the itemsets from the FP-tree of the incremental database (D_2) and upends each itemset to the FP-tree (T_1) of the original database (D_1). Hence, the process of merging is essentially same for both the algorithms. The advantage of the BIT algorithm lies in the fact that it processes the multiple occurrences of the same itemset (represented with the occurrence frequency in the FP-tree T_2) only once for merging, where as CanTree performs merging for every occurrence of the itemset. In the following section we bring out this difference by way of time complexity analysis.

Following notations are used for performance analysis:

m - Total number of items available. (This corresponds to maximum number of children for the root of a tree)

n – Number of leaf nodes of tree $T_{2.}$

q_i – Number of nodes / items in branch i (item set i) of $T_{2.}$

l – Number of node items of T_1 that match with the items of itemset i (i.e size of the matching prefix of T_1 for itemset i of T_2).

t – Total running time of the merging process.

t_i – Time required for processing each itemset i of $T_{2.}$

t_{cm} – Time required to **C**ompare and **M**ove to the next node in forward or downward direction (if comparison fails).

$t_{ca\,–}$ Time to **C**reate and **A**dd node, corresponding to an item of the itemset i of T_2, as descendant.

Consider the (worst case) scenario wherein while comparing the items of itemset i of T_2 at every level of the tree, the extreme right node item matches and the remaining items of itemset i are added as descendants of the extreme right leaf node of FP– tree T_1. Figure 3 below shows the worst case scenario for FP-tree T1.

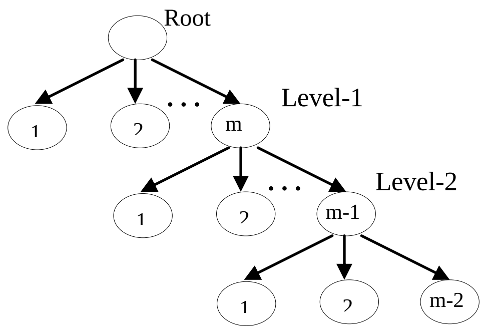


Figure 3. FP-tree T1 showing worst case scenario

Time, t_i =Time required for comparing items of ith itemset of T_2 and moving forward and downward + Time for adding all the remaining items of ith itemset of T_2.

Assuming, $q_i > l$

$$\therefore t_i = \sum_{j=0}^{l} [(m-j)*t_{cm}] + (q_i - l)*t_{ca}$$

In the worst case maximum items (level) of FP – tree would be equal to m-1, containing m-1 items in a branch.

i.e l = m-1.

$$\therefore t_i = \sum_{j=0}^{m-1} [(m-j)*t_{cm}] + (q_i - (m-1))*t_{ca}$$

$$\therefore t_i = \sum_{j=0}^{m-1} [(m-j)*t_{cm}] + (m - (m+1))*t_{ca},$$

as q_i=m, in the worst case

$$\therefore t_i = [m*t_{cm} + \ldots\ldots + 1*t_{cm}] + t_{ca}$$

$$= (1+2+\ldots\ldots\ldots\ldots.m)\, t_{cm} + t_{ca}$$

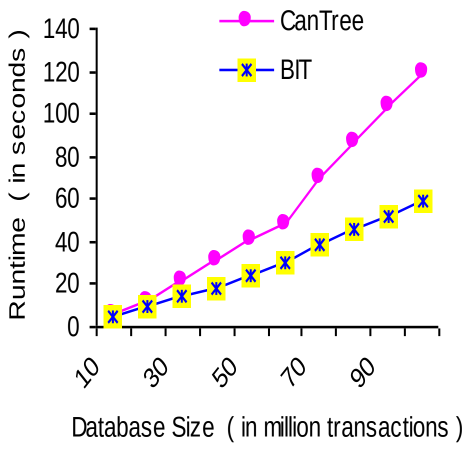
$$\text{i.e } t_i = \frac{m(m+1)}{2} t_{cm} + t_{ca}$$

There fore, the running time for entire merge process, (in the worst case) is:

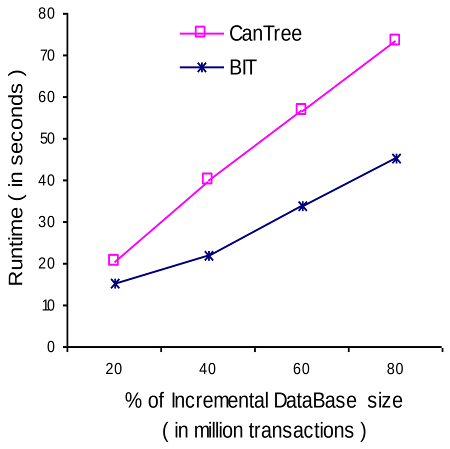
$$t = \sum_{i=1}^{n} \left[\left[\frac{m(m+1)}{2} * t_{cm} \right] + t_{ca} \right]$$

BIT algorithm gets transactions from the FP – tree T_2 unlike of CanTree which reads from database. In FP-Tree, multiple occurrences of each itemset are represented with a single branch, containing also the frequency of occurrence. Hence, in BIT algorithm multiple occurrences of an itemset are read and processed for merging only once. Therefore the value of ‘n’ is always much less than that of CanTree and hence the value of ‘t’. Further, as the database size increases the number of itemsets with high frequency also increases. Hence, BIT algorithm always takes much less time than the CanTree.

As the CanTree takes less time for FP-tree construction compared to AFPIM and CATS tree algorithms, we considered CanTree as the representative of sequential incremental FP-tree algorithms. We have implemented both CanTree and BIT algorithms and made comparative study of performance of the algorithms in terms of the execution time for tree construction. For CanTree, tree construction time is measured as the time required to read the transactions from incremental database and insert the items into the FP-tree constructed for original database. For BIT, tree construction time is measured as the time required for reading the itemset from the existing FP-tree of incremental database and inserting the itemsets into the FP-tree of original database.



(a)



(b)

Figure 4. Runtime: BIT Vs. CanTree

We tested the algorithm for their performance on duel processor machines with 2.8 GHz speed. We made multiple runs of the algorithms on synthetic databases of various sizes, ranging from 10 million transactions to 100 million transactions. Average itemset size of the transactions was 15 in the domain of 500 items. We tested the algorithms by measuring runtime against (i) varying size of databases keeping the original and incremental database size in fixed proportions (60: 40) and (ii) varying the proportion of original and incremental database keeping the total database size fixed. The results of the experiments are shown in the form of the graphs below in Figure 4 (a) & Figure 4 (b).

As can be observed from the graphs below, BIT algorithm takes much less time (almost half of the time required for CanTree) for the construction of FP-tree. As the size of the database increases (Figure 4 (a)), the runtime of BIT algorithm decreases. Further, the time difference between CanTree and BIT algorithm also increases as the database size increases. This is because, as the database size increases the frequency of occurrence of items also increases and hence CanTree requires more time to read transactions from incremental database. Whereas, in BIT algorithm as it reads itemsets from FP-tree and FP-tree contains only one representation for multiple occurrences of the itemsets, it reads only once.

In Figure 4(a), the runtime decreases as the percentage of the incremental database decreases (keeping the size of the original database fixed) for both CanTree and BIT. Here again, it can be observed that the difference in runtime of CanTree and BIT is more when the size of incremental database is more (i.e., percentage of incremental database) and it reduces as size reduces. As can be seen from the graph above in Figure 4, runtime of BIT algorithm reduces to nearly half of the runtime of sequential algorithms for large size databases.

5. CONCLUSION

BIT algorithm takes much less time to construct FP-tree by using previously generated FP-tree of incremental database. This is possible because BIT reads the incremental transactions from the FP-tree rather than database, where multiple occurrences of a transaction of the database are represented only once. As can be seen from the graph above in Figure 4, CanTree does more work to search for matching prefix as the database size increases. On the contrary BIT algorithm does less work as the database size increases. Because, as the database size increases the probability of recurrence of itemsets also increases and hence the difference in runtime between BIT algorithm and sequential incremental algorithms increases, i.e. BIT takes less time for tree construction.

6. REFERENCES

[1] Paul S. Bradley, J. E. Gehrke, Raghu Ramakrishnan and Ramakrishnan Srikant. “Philosophies and Advances in Scaling Mining Algorithms to Large Databases”. *Communications of the ACM,* August 2002

[2] R.J. Bayardo , “Efficient mining of long patterns from databases”. In Proc. SIGMOD 1998, pp. 85-93.

[3] 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.

[4] Agrawal R, Srikant R. “Fast Algorithms for Mining Association Rules”. In Proc. of *VLDB,* Sep 2- 15 1994, pp. 487-99.

[5] D W Cheung, J. Han, V.T. Ng, and C.Y. Wong,”Maintenance of discovered association rules in large databases: an incremental updating technique”. In Proc. of *ICDE 1996*, pp. 106–114.

[6] F. Bonchi and C. Lucchese, “ On closed constrained frequent pattern mining”. In Proc ICDM 2004,pp. 35-42.

[7] Lee, C-H., Lin, C-R., & Chen, M.S., “Sliding window filtering: an efficient method for incremental mining on a time-variant database”. In ELSEVIER-Information Systems,30(3), 2005, pp. 227-244.

[8] 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.

[9] Koh, J-L., & Shieh, S-F. “An Efficient Approach for Maintaining Association Rules Based on Adjusting FP-tree Structures”. Proceedings of the 2004 Database Systems for Advanced Applications, 2004, pp. 417-424.

[10] Cheung, W, & Zaïane, O. R.. “Incremental Mining of Frequent-patterns without Candidate Gneration or Support Constraint”. Proceedings of the 2003 International Database Engineering and Applications Symposium, 2003, pp. 111-116.

[11] Leung, C. K-S., Khan, Q. I., Li Z., & Hoque, T. “CanTree: A Tree Structure for Efficient Incremental Mining of Frequent Patterns”. Proceedings of the Fifth IEEE International Conference on Data Mining (ICDM’05), 2005.

[12] D. W. cheung , S.D. Lee, and B. kao, ”A general incremental technique for maintaining discovered association rules”. In Proc. DASFAA 1997, pp. 185-194.

[13] J. Han, J. Pei, and Y. Yin , “Mining Frequent Patterns without Candidate Generation”. In Proc. of SIGMOD 2000,pp.1-12