Frequent Item Sets in Large Uncertain Databases are mined efficiently

Miss. Ifrah Kaladgi, Prof. Yoginath Kalshetty

Abstract- In recent year, mining frequent itemsets over uncertain databases and computing statistical information on probabilistic data under the Possible World Semantics (PWS) and maintaining the result for database that is evolving is technically challenging. So we develop an incremental mining algorithm which enables results to be refreshed that reduces the need of re-executing the whole mining algorithm on new database that is more expensive. But for Mining frequent patterns in transaction databases previous studies adopt an Apriori-like candidate set generation-and-test approach. However, candidate set generation is still costly, especially when there exist a large number of patterns. In this study, we propose a novel tree structure called frequent-pattern tree (FP-Tree) structure for storing crucial information about frequent patterns. Efficiency of mining is achieved by compressing a large database into smaller data structure and it avoids the cost of repeated database scans, it adopts a pattern-fragment growth method to avoid the costly generation of a large number of candidate sets. Our performance study shows that this method is efficient and scalable for mining both long and short frequent patterns, and is faster than incremental algorithm.

Keywords- **Frequent Itemsets, Incremental mining, FP-Tree**

I. INTRODUCTION

Database mining has recently attracted tremendous amount of attention in the database research because of its wide applicability in many areas. Mining frequent itemsets over uncertain databases has also attracted much attention.
To interpret uncertain databases, the Possible World Semantics (PWS) is often used [1]. Although PWS is intuitive and useful querying or mining under this notion is costly. This is because an uncertain database has an exponential number of possible worlds. Performing data mining under PWS can be technically challenging. To finding frequent itemsets over uncertain databases is the most essential issue [8]. An itemset is frequent if and only if the expected support of such itemset is no less than a specifi ed minimum expected support threshold (minsup) and also if and only if the frequent probability of such itemset is large than a given Probabilistic threshold (minprob) [6].

The frequent itemsets discovered from uncertain data are naturally probabilistic, in order to reflect the confidence placed on the mining results. A Probabilistic Frequent Item set (PFI) is a set of attribute values that occurs frequently with a sufficiently high probability [2]. In order to make database mining a feasible technology by designing efficient algorithms for mining different types of patterns and designing efficient algorithm to update, maintain and manage the rule discover. Apriori algorithm [7] was designed that runs number of iterations and compute the large itemsets of the same size in each iteration starting from the size one itemsets. In each iteration they first construct a set of candidate itemsets and then scan the database to count the number of transaction that contain each candidate set and when database is updated it re-run the association rule mining algorithm on the whole updated database. This approach is though simple but has some disadvantages. All the computation done initially at finding out old large itemsets are wasted and all large itemsets have to be computed again from scratch.

So we develop incremental mining algorithm as uncertain Fast Update algorithm (uFUP) [6] which maintains frequent itemsets result in an evolving database by reusing the information derived from old large itemsets. uFUP is which is derived from FUP algorithm[3] that work faster than Apriori and applied to updated database to find the new large itemsets and it is very effective in reducing the number of candidate sets.

It is costly to handle a number of candidate sets and accumulate and test their occurrence frequencies. 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. In order to avoid candidate generation-and-test and utilize and to reduce the cost in frequent-pattern mining.

In this paper we develop and integrate the following techniques in order to solve this problem. First compact data structure, called frequent-pattern tree, or FP-tree [4] in short is constructed, which is extended prefix-tree structure storing crucial, quantitative information about frequent patterns. To ensure that the tree structure is compact and informative, only frequent length-1 items will have nodes in the tree, and the tree nodes are arranged in such a way that more frequently occurring nodes will have better chances of node sharing
than less frequently occurring ones. Our experiments show that such a tree is compact, and it is sometimes orders of magnitude smaller than the original database. Subsequent frequent pattern mining will only need to work on the FP-tree instead of the whole data set.

An FP-tree-based pattern-fragment growth mining method is developed [5], which starts from a frequent length-1 pattern, examines only its conditional-pattern base (a "sub-database" which consists of the set of frequent items co occurring with the suffix pattern), constructs its (conditional) FP-tree, and performs mining recursively with such a tree. The pattern growth is achieved via concatenation of the suffix pattern with the new ones generated from a conditional FP-tree. Since the frequent itemset in any transaction is always encoded in the corresponding path of the frequent-pattern trees, pattern growth ensures the completeness of the result. In this context, our method is not Apriori-like restricted generation-and-test but restricted test only. The major operations of mining are count accumulation and prefix path count adjustment, which are usually much less costly than candidate generation and pattern matching operations performed in most Apriori-like algorithms

ISSN: 2278 – 1323

The search technique employed in mining is a partitioning -based, divide-and conquers method rather than Apriori-like level-wise generation of the combinations of frequent itemsets. This dramatically reduces the size of conditional-pattern base generated at the subsequent level of search as well as the size of its corresponding conditional FP-tree. Moreover, it transforms the problem of finding long frequent patterns to looking for shorter ones and then concatenating the suffix. It employs the least frequent items as suffix, which offers good selectivity. All these techniques contribute to substantial reduction of search costs.

II. PROBABILISTIC FREQUENT ITEMSETS

Let k be a set of items. The support of k denoted by S(k) is the number of transactions in which k appears in a transaction database. In precise databases, S(k) is a single value. This is no longer true in uncertain databases, because in different possible worlds, S(k) can have different values. An item set k is said to be frequent in a database D if S(k)>= msc(D), where msc(D)=minsup * n is called the minimal support count of D. where minsup is value passed by user.

Using frequentness probabilities, we can determine whether an item set k is frequent. we adopt k is a Threshold-based PFI if its frequentness probability i.e Pr is larger than some user-defined threshold . Formally, k is a threshold-based PFI, if Pr(k) >= minprob .We call minprob the frequentness probability threshold. where minprob is value passed by user.

III. EXACT INCREMENATAL MINING

We now examine how to efficiently maintain a set of PFI's in an evolving database, where new items, are constantly appended to it. The framework of uFUP is similar to that of Apriori. It contains a number of iterations. The iteration start at the size-one itemsets, and at each iteration, all the large itemsets of the same size are found. Moreover the candidate sets at each iteration are generated based on the large itemsets found at the previous iterations

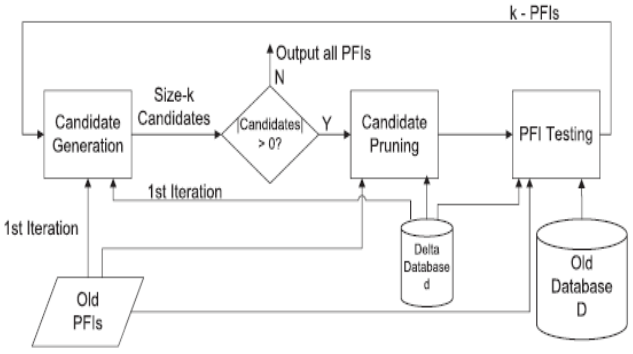

Fig 1: Working of Ufup

A. Candidate generation:

In the first iteration, size-1item sets that can be 1-PFIs are obtained, using the PFIs discovered from D, as well as the updated database d. In subsequent iterations, this phase produces size (k+1) candidate item sets, based on the k-PFIs found in the previous iteration. If no candidates are found, uFUP halts.

B. Candidate pruning:

The goal of this phase is to remove infrequent itemsets from a set of size-k candidates. Prune is used to remove itemsets from previous result. This is done by first checks if I is a frequent itemset in D . If this is false, and if cntd(I) does not exceed msc(d), then I cannot be a PFI in D, and I can be pruned. Notice that Prune does not test I on (D +d)D+, which can be expensive. Instead, it only computes cntd(I), which can be obtained by scanning d once. If n , the size of d, is small, then getting cntd(I) incurs a low cost. With the aid of d and the PFIs found from D, this phase filters the candidate item sets that must not be a PFI.

C. PFI testing:

The objective of this phase is to verify whether these candidates are really k-PFIs. In particular, the subroutine test is invoked to compute the spmfs of these itemsets on D+. Once this is obtained, we can easily verify whether these candidates are true k-PFIs. Although this approach can be used to compute the s-pmf of an itemset I, this can be expensive, especially if the size of D+ is large. However, if we know that I is a PFI in D, as well as its s-pmf in D, it is possible to derive the s-pmf of I in D+ without computing it from scratch. This involves the use of database D+, as well as the spmfs of PFIs on D.

IV .FREQUENT PATTERN TREE METHODOLOGY

FP-Tree is used for finding frequent itemsets in less time without candidate generation it goes through following process

A. Pre-processing:

A compact data structure can be designed based on the following observations:

1) Since only the frequent items will play a role in the frequent-pattern mining, it is necessary to perform one scan of transaction database DB to identify the set of frequent items .

2) If the set of frequent items of each transaction can be stored in some compact structure, it may be possible to avoid repeatedly scanning the original transaction database.

3) If multiple transactions share a set of frequent items, it may be possible to merge the shared sets with the number of occurrences registered as count. It is easy to check whether two sets are identical if the frequent items in all of the transactions are listed according to a fixed order.

4) If two transactions share a common prefix, according to some sorted order of frequent items, the shared parts can be merged using one prefix structure as long as the count is registered properly. If the frequent items are sorted in their frequency descending order, there are better chances that more prefix strings can be shared.

B. *Building FP-Tree*:

A frequent-pattern tree (FP-tree) is a tree structure defined below.

1) It consists of one root labeled as "null", a set of item-prefix subtrees as the children of the root, and a frequent-item-header table.

2) Each node in the item-prefix subtree consists of three fields: item-name, count, and node-link, where item-name registers which item this node represents, count registers the number of transactions represented by the portion of the path reaching this

ISSN: 2278 – 1323

node, and node-link links to the next node in the FP-tree carrying the same item-name, or null if there is none.
3) Each entry in the frequent-item-header table consists of two fields, (1) item-name and (2) head of node-link (a pointer pointing to the first node in the FP-tree carrying the item-name).

Algorithm 1 (FP-tree construction)
Input: A transaction database *DB* and a minimum support threshold X.
Output: FP-tree, the frequent-pattern tree of *DB*.
Method: The FP-tree is constructed as follows.
1) Scan the transaction database DB once. Collect F, the set of frequent items, and the support of each frequent item. Then sort F in support-descending order as FList, the list of frequent items.
2) Create the root of an FP-tree T , and label it as "null". For each transaction Trans in DB do the following.

Select the frequent items in Trans and sort them according to the order of F List. Let the sorted frequent-item list in Trans be [p | P], where p is the first element and P is the remaining list. Call insert tree([p | P], T).

The function insert tree([p | P], T) is performed as follows. If T has a child N such that N.item-name = p.item-name, then increment N's count by 1; else create a new node N, with its count initialized to 1, its parent link linked to T , and its node-link linked to the nodes with the same item-name via the node-link structure. If P is nonempty, call insert tree(P, N) recursively.

Below is the example for construction of FP-TREE where support value=3.

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

Fig 2: A Transaction Database

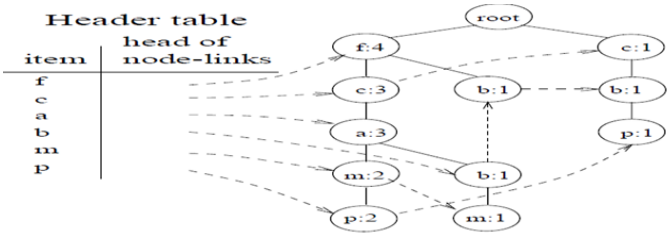

Fig 3: FP-TREE constructed for fig 2

C. *Mining Frequent Pattern using FP-Tree*:
For any frequent item i, all the possible patterns containing only frequent items and i can be obtained by following i 's node-links, starting from i 's head in the FP-tree header. All the patterns containing frequent items that a node i participates can be collected by starting at i 's node-link head and following its node-links. We examine the mining process by starting from the bottom of the node-link header table and we construct Conditional Base Pattern.

From the result of conditional Base pattern we constructed conditional FP-Tree which takes only item that satisfy the pass support threshold value and from this result we construct frequent patterns

V. RESULT

The Experimental result was done on "accidents dataset" comes from the Frequent Item Set Mining (FIMI) Data Set Repository [9][10]. Our Experiment was carried out on Windows 8 Operating System, on machine with Intel Core i3 with 4 GB memory. The result generated was related to execution time require to generate frequent pattern by both algorithm i.e by UFUP and FP-Tree.

The scalability of FP-growth and UFUP for pass support threshold value is shown in Figur4.

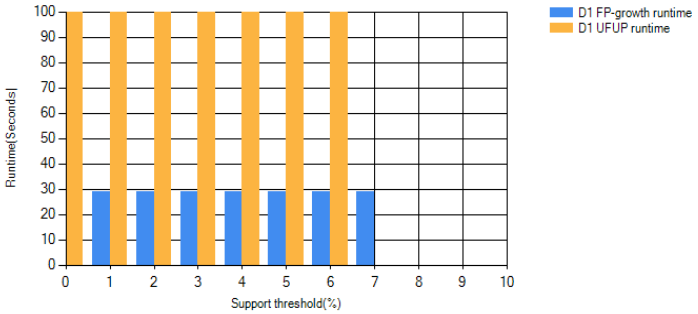

Figure 4: Scalability with threshold

FP-growth scales much better than UFUP. This is because as the support threshold goes down, the number as well as the length of frequent itemsets increases dramatically. The candidate sets that UFUP must handle become extremely large, and the pattern matching with a lot of candidates by searching through the transactions becomes very expensive. Overall, FP-growth is about an order of magnitude faster than UFUP in large databases, and this gap grows wider when the minimum support threshold reduces.
.

VI. CONCLUSION

In this paper we have proposed a novel tree structure i.e frequent pattern tree (FP-tree), for storing compressed and crucial information about frequent patterns. There are several advantages of FP-Tree over other approaches: (1) It constructs a highly compact FP-tree, which saves the costly database scans in the subsequent mining processes. (2) It applies a pattern growth method which avoids costly candidate generation and test by successively concatenating frequent 1-itemset found in the (conditional) FP-trees. Our performance study shows that the method mines both short and long patterns efficiently in large databases outperforming the current candidate pattern generation-based algorithms.

REFERENCE

[1] L. Sun, R. Cheng, D.W. Cheung, and J. Cheng, "Mining uncertain Data with Probabilistic Guarantees," Proc. 16th ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining, 2010.
[2] T. Bernecker, H. Kriegel, M. Renz, F. Verhein, and A.

ISSN: 2278 – 1323

Zuefle," Probabilistic Frequent Itemset Mining in Uncertain Databases," Proc. 15th ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining (KDD), 2009.
[3] D.W. Cheung, Jiawei Han, V.T. Ng and C.Y. Wong, "Maintenance of Discovered Association Rules in Large Databases: An Incremental Updating Technique" IEEE Data Engineering 1996 proceeding of twelfth international conference.
[4] Christian Borgelt , "An Implementation of the FP-growth Algorithm", ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining (KDD), 2005.
[5] J. Han, J. Pei, and Y. Yin, "Mining Frequent Patterns without Candidate Generation," Proc. ACM SIGMOD Management of data,2000.
[6] L Wang, D. W Cheung, R. Cheng, Sau Dan Lee and X.S Yang, "Efficient Mining of Frequent Item Sets on Large Uncertain Databases" IEEE Data Engineering VOL 24, NO 12 December 2012.
[7] R. Agrawal, T. Imieli_nski, and A. Swami, "Mining Association Rules between Sets of Items in Large Databases," Proc. ACM SIGMOD Int'l Conf. Management of Data, 1993.
[8] C.K. Chui, B. Kao, and E. Hung, "Mining Frequent Itemsets from Uncertain Data," Proc. 11th Pacific-Asia Conf. Advances in Knowledge Discovery and Data Mining (PAKDD), 2007.
[9] Karolien Geurts "Traffic Accident Data Set", "accident.pdf" Research Group Data and Modelling Limburgs Universitair Centrum Universitair , Belgium
[10] Karolien Geurts , Geert Wets , Tom Brijs and Koen Vanhoof "Profiling High Frequency Accident Locations Using Association Rules" Proceedings of the 82th Annual Meeting of the Transportation Research Board, Washington, January 12-16, USA, 18pp, 2003.

Miss. Ifrah Kaladgi Received her B.E degree in Computer Science and Engineering from Solapur University in 2011 respectively. Currently she is pursuing M.E degree in Computer Science and Engineering at Solapur University. Her research interest is Data Mining.

Prof. Yoginath Kalshetty Received the B.E and M.E degrees in Computer Science and Engineering from Mumbai University and Shivaji University in 2003 and 2011 respectively. Currently he is pursuing PhD. His research interest is Algorithms.

ISSN: 2278 – 1323