An Incremental Mining Solution for Over Top K Queries over Dynamic Databases

Konda Sreenu
Assistant Professor
Vegesna Usha Praveena
M. Tech Student
vahiduddin sheriff
Assistant Professor
C. R. Reddy College of Engineering, Eluru, West Godavari Dt, AP, India

Abstract**-**Conventional frequent pattern mining algorithms usually considered the databases are static and focused on batch mining. In real-world applications, however, new records are usually inserted into databases or deleted from the database. When new records are added to databases, the original association rules may become invalid, or new implicitly valid rules may appear in the resulting updated databases. In these situations, conventional batch-mining algorithms must re-process the entire updated databases to find final association rules as a result it was not consider the earlier mining results. To overcome this limitation new algorithms are proposed to use the earlier mining results and to reduce the number of scans and the number of candidate itemsets. In this paper we are proposing a new incremental mining algorithm which uses the pre-large concept and the border itemset. The proposed algorithm can effectively handle all the cases arises in incremental mining algorithms, in which itemsets are small in an original database but large in newly inserted transactions can be considered for scanning if and only if it becomes a pre-border itemset., although it does need additional storage space to record the pre-large itemsets. The algorithm is scalable with respect to number of transactions and for different threshold values.

Keywords**-**Data Mining, Frequent patterns, incremental mining algorithm, pre-large itemsets, border-itemset.

I. INTRODUCTION

Mining of Frequent Patterns (FPM) is essential prerequisite to form the knowledge in Knowledge Discovery in Databases (KDD) process. In Association Rule Mining, patterns are referred with itemsets. Other kinds of patterns are used in various Data Mining functionalities such as Sequential Pattern Mining, Spatial Pattern Mining, Temporal Pattern Mining and Correlated Patterns Mining. The candidate-generate-test-paradigm of Apriori kind algorithms [1, 2, 3, 8, 19, 25, 26, 27], FP-tree based algorithms [13, 14, 15, 30] has motivated several researchers to contribute fast methods of extraction of frequent itemsets. Some research was also focused on kinds of frequent patterns that includes closed [23, 31, 32, 20], maximal [7, 10], utility [4], non-derivable patterns [9] and rare item sets [28] for their non-redundancy and compactness of association rules useful in decision making. Taking temporal information into account, researchers proposed various algorithms for mining temporal patterns [5, 6, 11, 17, 24, 38] including cyclic patterns [22] and calendar-based patterns [18], rare itemsets [28], frequent episodes [28], etc. The purpose of extracting frequent patterns over a transactional database is to understand the hidden knowledge in it and for better decisions to improve the business. In the earlier works of frequent pattern mining the researchers have proposed solutions for extracting frequent patterns over the entire database and the sets of patterns extracted were huge. Such a large number of patterns are not helpful for analysis purpose. Though early researchers have treated all the transactions uniformly in a database, but at a later stage, researchers have considered the time stamps of the transactions to get more insight into frequent patterns. It is also noticed that frequent patterns need not uniformly present over the entire transaction database, rather present at some part may be enough. This observation motivated researchers to design algorithms to do micro analysis over the database for better understanding of the hidden knowledge. Recently, the researchers are more focused on finding temporal knowledge that reveals the behaviour of frequent itemsets, such as extracting the patterns that are frequent over a specific period; extracting the specific timestamps, where the support of the frequent itemset before/after the timestamp increases/decreases drastically, etc. Wan and An [29] have introduced "Transitional Patterns" which represent patterns whose frequency dramatically changes over time in a transaction database. Primarily they focused on discovering time points at which positive (or negative) transitional patterns increases (or decreases) their frequency significantly over time. Their study gave scope to disclose the dynamic behaviour of the frequent patterns as they considered the time-stamps of transactions in the mining process. Applications of transitional patterns include tuning the marketing strategies in retail environments, analysing the time points at which drugs cause the side effects in diagnosis, restructuring web links based on frequency of visiting the web pages, finding the low and high end points of profit in the stock market, etc. TP-Mine Algorithm [29] runs in two stages and uses three scans for extracting the transitional patterns. At the first stage, it uses two scans and extracts the frequent patters (by using FP-Growth algorithm [15]) and in the second stage, an additional scan over the database extracts the transitional patterns with their significant frequency ascending and descending milestones in the given transaction database. There may exist many time points exhibit the behaviour of transitional patterns, the TP-Mine algorithm [29] records only those time points where the transitional ratio is maximum and each such time point gives the knowledge about the behaviour of the pattern over the entire database. Both the Apriori and the FP-tree mining approaches belong to batch mining. That is, they must process all the transactions in a batch way. In real-world applications, new transactions are usually inserted into databases or obsolete data is deleted from the databases regularly. With the advent of Data Warehouse it is now usual every x units of time (usually years) the data in the data warehouse is updated. For example a data warehouse constructed in 2010 may be updated in 2013, 2016 and 2017. Every addition of new transactions there might be possible an infrequent itemset in the old database may become infrequent in the new database or an itemset which is frequent in the old database may become infrequent in the updated database. The case-1 can be solved by reading only the updated part of the database. But for case-2 it is required to read the entire database Suppose ODB is a database and NDB is a new database of transactions to be appended to ODB to get UDB. It is acceptable to read NDB any number of times, but it is not acceptable to do more scans on ODB.

A naive solution would be to rerun the algorithm from scratch every time new data arrives. This, however, is highly inefficient, as adding even a very small amount of new data will require running the association generation algorithm on all known transactions. Thus, the researchers introduced the new type of algorithms named them as “incrementalalgorithm", which allows generating the new associations in an incremental manner. Instead of processing all the records again, such an algorithm would only perform a small fraction of the work on each new set of data and thereby provide the results in a timely manner. If the changes do not produce any new frequent sets, then there is no access to the old data. Thus, that costly scanning of the entire database will only be performed when *new* frequent sets are obtained. In case the entire database is scanned, the number of passes over the database is should be small, and generally the support is required for a few candidates. Many incremental mining algorithms proposed in the literature [12].

One noticeable incremental mining algorithm was the Fast-Updated Algorithm (called FUP), which was proposed by Cheung et al. [34] for avoiding the shortcomings mentioned above with the conventional algorithms. Although the FUP algorithm could indeed improve mining performance for incrementally growing databases, original databases still needed to be scanned when necessary. Yonatan Aumann, et.al, [33] provide three variants of algorithms: one for the case of additions alone, one for additions and deletions, and one for the case where the analyst wishes to change the support threshold, without having to run the entire algorithm anew. These authors use the border itemsetconcept to reduce the number of candidate itemsets.

Hong et al. thus proposed the pre-large concept to further reduce the need for rescanning original database [35]. A pre-large itemset was defined based on two support thresholds. The upper support threshold was the same as that used in the conventional mining algorithms. The lower support threshold defined the lowest support ratio for an itemset to be treated as pre-large. An itemset with its support ratio below the lower threshold was thought of as a small itemset. The algorithm did not need to rescan the original database until a number of new transactions had been inserted. Since rescanning the database spent much computation time, the maintenance cost could thus be reduced in the pre-large-itemset algorithm.

Hong et al. also modified the FP-tree structure and designed the fast updated frequent pattern trees (FUFP-trees) to efficiently handle newly inserted transactions based on the FUP concept [35]. The FUFP-tree structure was similar to the FP-tree structure except that the links between parent nodes and their child nodes were bi-directional. Besides, the counts of the sorted frequent items were also kept in the Header Table of the FP-tree algorithm. In [34] the authors proposed the structure of prelarge tree for handling the deletion of records based on the concept of pre-large itemsets. A structure of prelarge tree is to keep not only frequent items but also pre-large items. Based on the pre-large itemsets, the proposed approach can effectively handle cases in which itemsets are small both in an original database and deleted records. The proposed algorithm does not require rescanning the original databases to construct the prelarge tree until a number of deleted records have been processed.

The prelarge concept helps in the incremental mining algorithms by reducing the number of scans on the database till some specified number of transactions is added or deleted. The prelarge concept used algorithms suffers with the maintenance of a large set of candidate itemsets. As the prelarge concept used algorithms uses a very less support threshold for an itemset to become frequent. It is motivated us to design an algorithm which uses the prelarge concept and maintain the number of candidates very less.

The objective of this paper is proposing a newsolution for extracting frequent patterns in dynamic databases by using the prelarge concept and border itemsetconcept. This paper is organized into five sections including the introduction section. Section two gives the detailed paper on incremental mining algorithms, section three presents the proposed algorithm and the pseudo code of the algorithm and the databases used for implementations, section four presents the results obtained by applying the proposed algorithm, and section five gives the conclusion of this paper.

II. RELATED WORK

The conventional frequent pattern mining algorithms usually considered the database size static and focused on batch mining. In real-world applications, however, new records are usually inserted into databases, and designing a mining algorithm that can maintain association rules as a database grows is thus critically important When new records are added to databases, the original association rules may become invalid, or new implicitly valid rules may appear in the resulting updated databases. In these situations, conventional batch-mining algorithms must re-process the entire updated databases to find final association rules. Two drawbacks may exist for conventional batch-mining algorithms in maintaining database knowledge:

(a) Nearly the same computation time as that spent in mining from the original database is needed to cope with each new transaction. If the original database is large, much computation time is wasted in maintaining association rules whenever new transactions are generated

(b) Information previously mined from the original database, such as large itemsets and association rules, provides no help in the maintenance process.

Cheung et al. [34] proposed an incremental mining algorithm, called FUP (Fast Update algorithm), for incrementally maintaining mined association rules and avoiding the shortcomings mentioned above. The FUP algorithm modifies the Apriority mining algorithm and adopts the pruning techniques used in the DHP (Direct Hashing and Pruning) algorithm. It first calculates large item sets mainly from newly inserted transactions, and compares them with the previous large item sets from the original database. According to the comparison results, FUP determines whether re-scanning the original database is needed, thus saving some time in maintaining the association rules. Although the FUP algorithm can indeed improve mining performance for incrementally growing databases, original databases still need to be scanned when necessary. A good rule maintenance algorithm should thus accomplish the following.

1. Evaluate large item sets in the original database and determine whether they are still large in the updated database;
2. Find out whether any small item sets in the original database may become large in the updated database;
3. Seek item sets that appear only in the newly inserted transactions and determine whether they are large in the updated database. These are accomplished by the FUP algorithm [34].

The FUFP-tree construction algorithm is based on the FP-tree algorithm [15]. The links between parent nodes and their child nodes are, however, bi-directional. Bi-directional linking will help fasten the process of item deletion in the maintenance process. Besides, the counts of the sorted frequent items are also kept in the Header Table. An FUFP tree must be built in advance from the original database before new transactions come. When new transactions are added, the FUFP-tree maintenance algorithm will process them to maintain the FUFP tree. It first partitions items into four parts according to whether they are large or small in the original database and in the new transactions. Each part is then processed in its own way. The Header Table and the FUFP-tree are correspondingly updated whenever necessary. Consider an original database and some transactions to be deleted, the following cases may arise:

1. Case 1: An itemset is frequent both in original database and in deleted transactions.
2. Case 2: An itemset is frequent in an original database but not frequent in deleted transactions
3. Case 3: An itemset is not frequent in original database but is frequent in deleted transactions
4. Case 4: An itemset is not frequent in both original database and in deleted transactions.

Case 2 and 3 will not affect the final frequent itemsets. Itemsets in case 1 are frequent in both the original database and deleted transactions. Thus, some existing frequent itemsets may be removed from the after the database is updated. At last, itemsets in case 4 are infrequent in both original database and in deleted transactions. Some frequent item sets may thus be added it however, requires the original database to be rescanned for rebuilding the FUFP-tree of the final updated database. After that, the FP-growth algorithm must be used to mine all FIs [15].

In order to reduce the need for rescanning the original database, Lin et al. [35] proposed a pre-large tree structure and designed an algorithm to rebuild the pre-large tree based on the concept of pre-large item sets. The pre-large tree is similar to the FUFP-tree. When some transactions are deleted from the database, the pre-large-tree-based approach will process them to maintain the pre-large tree. Unlike the FUFP-tree-based approach, it partitions items into nine cases according to whether they are large or pre-large or small in the original database and in deleted transactions. The summary of the nine cases and their results is given in Table 1. The algorithm does not require the original database to be rescanned until a number of deleted transactions have been processed. When some transactions are deleted from the database, some nodes are removed from or inserted into the pre-large tree. After that, the FP-growth algorithm is applied for the pre-large tree of the entire database to mine all frequent item sets. So, the pre-large-tree-based approach does not utilize item sets which have been mined from the original database.

Hong et al. proposed the pre-large concept to reduce the need of rescanning original database [35] for maintaining association rules. A pre-large itemset is not truly large, but may be large with a high probability in the future. A pre-large itemset is not truly large, but may be large with a high probability in the future. Two support thresholds, a lower support threshold and an upper support threshold, are used to realize this concept. The upper support threshold is the same as that used in the conventional mining algorithms. The support ratio of an itemset must be larger than the upper support threshold in order to be considered large. On the other hand, the lower support threshold defines the lowest support ratio for an itemset to be treated as pre-large. An itemset with its support ratio below the lower threshold is thought of as a small itemset. Pre-large itemsets act like buffers and are used to reduce the movements of itemsets directly from large to small and vice-versa.

Considering an original database and some records to be deleted by the two support thresholds, itemsets may fall into one of the following nine cases illustrated in Figure 2.1. Cases 2, 3, 4, 7 and 8 will not affect the final association rules. Case 1 may remove some existing association rules, and cases 5, 6 and 9 may add some new association rules. If we retain all large and pre-large itemsets with their counts after each pass, then cases 1, 5 and 6 can be handled easily. Also, in the maintenance phase, the ratio of deleted records to old transactions is usually very small. This is more apparent when the database is growing larger. It has been formally shown that an itemset in case 9 cannot possibly be large for the entire updated database as long as the number of transactions is smaller than the number f shown below [35]:

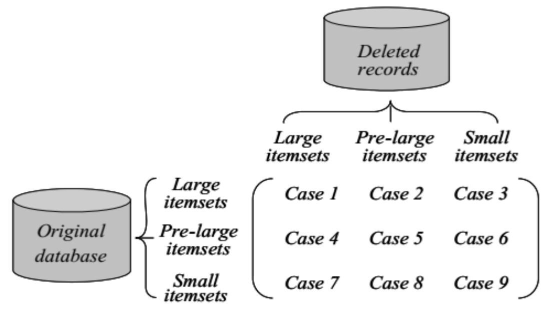

Fig 2.1: Nine cases arising from original database and the deleted records

$f = (S_u - S_l) * d / S_u$ ————

Where f is the safety number of deleted records, S_u is the upper threshold, S_l is the lower threshold, and d is the number of original transactions. The concept of pre-large item sets was proposed by Hong et al. [5]. It uses two thresholds, namely the upper threshold and the lower threshold, to set the pre-large item sets. The upper threshold is similar to min sup. The lower threshold defines the lowest support ratio for an itemset that is to be treated as pre-large. An itemset with a support ratio below the lower threshold is seen as small. Hong et al. [35] also proposed the pre-large-itemset algorithm. It is based on a safety number f of inserted transactions to reduce the need for rescanning the original database for the efficient maintenance of the large item sets. A summary of the nine cases and their results are given in Table 2.1.

Table 2.1: Nine cases and their results

Case: Original - Deleted	Results
Case 1: Large - Large	Large or pre-large or small, determined from
Case 2: Large - Pre-Large	Always large.
Case 3: Large - Small	Always large.
Case 4: Pre-Large Large	Pre-large or small, determined from
Case 5: Pre-Large - Pre-Large	Large or pre-large or small, determined from
Case 6: Pre-Large - Small	Large or pre-large, determined from
Case 7: Small - Large	Always small.
Case 8: Small - Pre-Large	Always small.
Case 9: Small - Small	Pre-large or small, determined from existing information.

IT-tree-based approach [36] is one of the famous approaches for mining FIs in static transaction databases. It is based on equivalence classes, scans the database only once, uses the depth-first traversal technique to generate item sets and to compute the supports of the item sets fast by tidset intersections. This paper proposes an incremental algorithm for handling of deleted transactions based on the IT-tree structure and pre-large item sets. Like the pre-large-tree-based approach, when transactions are deleted from the database, the proposed approach will partition items into nine cases according to whether they are large or pre-large or small in the original database and in deleted transactions. The summary of the nine cases and their results is given in Table 1. Its main idea is to use the depth-first traversal technique to update the final supports of the item sets from their tidsets in deleted transactions. The supports of the item sets in deleted transactions are computed by tidset intersections. All FIs are mined using depth-first order traversal. The advantage of the IT-tree-based approach is to utilize item sets which have been mined from the original database. The proposed algorithm only processes deleted transactions for rebuilding the final IT-tree. Besides, the concept of pre-large item sets is used to reduce the need for rescanning the original database to save maintenance cost. The algorithm does not require the original database to be rescanned until many deleted transactions have been processed.

The Borders algorithm is based on the notion of *border sets*, introduced in [33]. A set *X* is a *border set* if all its proper subsets are frequent sets (i.e., sets with at least minimum support), but it itself is not a frequent set. Thus, the collection of *border sets* defines the borderline between the frequent sets and non-frequent sets, in the lattice of attribute sets. The Border algorithm maintains the count information for all frequent sets and all border sets in the current relation. When an increment, R_N, arrives, the increment alone is scanned to obtain its support for all (previous) frequent and border sets. From this information, we compute (with no extra data access), the support of all frequent and border sets in the combined relation. Additional scans of the entire relation are performed only if the support of some border set has reached the minimum support threshold (thus turning into a frequent set). This policy guarantees that the full relation is never scanned if there is no new frequent set. Furthermore, even when additional scans are required; monitoring the border sets minimizes the amount of counting work performed during these scans. The Borders algorithm also extended to handle the case of deletions as well as when the support threshold is changed.

III. INCREMENTAL DATA MINING ALGORITHM USING PRE-LARGE AND BORDER ITEMSETS

Although the FUP algorithm [34] focuses on the newly inserted transactions and thus saves much processing time by incrementally maintaining rules, it must still scan the original database to handle case 3 in which a candidate itemsets is large for new transactions but is not recorded in large itemsets already mined from the original database. This situation may occur frequently, especially when the number of new transactions is small. In an extreme situation, if only one new transaction is added each time, then all items in the transaction are large since their support rations are 100% for the new transaction. Thus, if case 3 could be efficiently handled, the maintenance time could be further reduced.

In [35] the authors proposed the concept of pre-large itemsets to solve the problem represented by case 3. *A pre-large itemset is not truly large, but promises to be large in the future.* A lower support threshold and an upper support threshold are used to realize this concept. The upper support threshold is the same as that used in the conventional mining algorithms. The support ratio of an itemset must be larger than the upper support threshold in order to be considered large. On the other hand, the lower support threshold defines the lowest support ratio for an itemset to be treated as pre-large. Pre-large itemsets act like buffers in the incremental mining process and are used to reduce the movements of itemsets directly from large to small and vice-versa. Considering an original database and transactions newly inserted using the two support thresholds, itemsets may thus fall into one of the following nine cases illustrated in Figure 3.1.

Cases 1, 5, 6, 8 and 9 above will not affect the final association rules. Cases 2 and 3 may remove existing association rules, and cases 4 and 7 may add new association rules. If we retain all large and pre-large itemsets with their counts after each pass, then cases 2, 3 and case 4 can be handled easily. Also, in the maintenance phase, the ratio of new transactions to old transactions is usually very small. This is more apparent when the database is growing larger. An itemset in case 7 cannot possibly be large for the entire updated database as long as the number of transactions is small compared to the number of transactions in the original database.

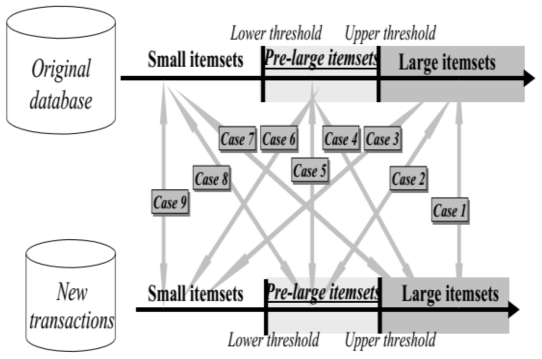

Fig 3.1 - The cases arises from adding new transactions

The algorithm proposed in [35] by using the pre-large itemset is described as follows: The large and pre-large itemsets with their counts in preceding runs are recorded for later use in maintenance. As new transactions are added, the proposed algorithm first scans them to generate candidate 1-itemsets (only for these transactions), and then compares these itemsets with the previously retained large and pre-large 1-itemsets. It partitions candidate 1-itemsets into three parts according to whether they are large or pre-large for the original database. If a candidate 1-itemset from the newly inserted transactions is also among the large or pre-large 1-itemsets from the original database, its new total count for the entire updated database can easily be calculated from its current count and previous count since all previous large and pre large itemsets with their counts have been retained. Whether an originally large or pre-large itemset is still large or pre-large after new transactions have been inserted is determined from its new support ratio, as derived from its total count over the total number of transactions. On the contrary, if a candidate 1-itemset from the newly inserted transactions does not exist among the large or pre-large 1-itemsets in the original database, then it is absolutely not large for the entire updated database as long as the number of newly inserted transactions is within the safety threshold. In this situation, no action is needed. When transactions are incrementally added and the total number of new transactions exceeds the safety threshold, the original database is re-scanned to find new pre-large itemsets in a way similar to that used by the FUP algorithm. The algorithm can thus find all large 1-itemsets for the entire updated database. After that, candidate 2-itemsets from the newly inserted transactions are formed and the same procedure is used to find all large 2-itemsets. This procedure is repeated until all large itemsets have been found.

3.1 Limitation of the incremental algorithm proposed in [37]:

Once the number of added transactions crosses the safety threshold value, the algorithm scans the database in level by level. At each level it is considering all the large or prelargeitemsets in the added database. The limitation of the algorithm in [37] is the number of candidate itemsets are very large at each level because of the number of transactions added to the database is very less and the threshold is used to qualify an itemset as a pre-large itemset is also very small. The number of scans is also more as it is level by level approach.

3.2 PROBLEM STATEMENT

Given a database D with n transactions, the set of large itemsets, the prelargeitemsetsand borderitemsets, in D, the lower and upper thresholds (S_l and S_u) and the set of transactions T. The algorithm must find the all the frequent itemsets in the database U (where U = D + T) by using less number of scans and with the less number of candidate itemsets used at each level compared to the algorithm proposed in [38].

3.3 NEW INCREMENTAL MINING ALGORITHM BASED ON PRE-BORDER ITEMSET

Pre-Border Itemset- Anyitemset X is called pre-border itemset if it is not a pre-large itemset and it is not a large itemset, but all its proper subsets are either large itemsets or pre-large itemsets.

The proposed new incremental mining algorithm

INPUT: A lower support threshold S_l**,** an upper support threshold Su, a set of large itemsets L, pre-large itemsets PL, and set of border itemsets B in the original database consisting of (d + c) transactions (where d is the number of transactions in the original database and c is the transactions added previously to the original database), and a set of new |T| transactions.

OUTPUT: A set of final frequent itemsets for the updated database U (U = D + T).

```
STEP 01:    Calculate the safety number f of new transactions by using the following formulae:
                $f = (S_u - S_l) * |T| / 1-S_u$.
STEP 02:    Scan T and find the count C(X, T), for all $X \epsilon L \cup PL \cup B$.
STEP 03:    For all $X \epsilon B \cup L \cup PL$ {
STEP 04:        C (X, DUT) = C( X, D) + C(X, T).
STEP 05:        S(X, DUT) = C(X, DUT) / |D| + |T|  }
STEP 06:    $PB = \{X \epsilon B \mid S(X, DUT) \geq S_l\}$
STEP 07:    $L = \{X \epsilon L \cup PL \cup PB \mid S(X, DUT) \geq S_u\}$
STEP 08:    $PL = \{X \epsilon L \cup PL \cup PB \mid S(X, DUT) < S_u$ AND $S(X, DUT) \geq S_l\}$
STEP 09:    $B = \{X \mid$ for all $x \epsilon X, X - \{x\} \epsilon L \cup PL\}$
STEP 10:    $m = \max\{i \mid PB(i) <> \Phi\}$
STEP 11:    $L_0 = \Phi, PL_0 = \Phi, i = 1$.
STEP 12:    while ($L_i <> \Phi$ or $PL_0 <> \Phi$  $i \leq m$ AND $c + |T| > f$ } {
STEP 13:        $C_{i+1} = \{ X \mid |X| = i+1$, for some x, $X - \{x\} \epsilon PB \cup L$ or $X-\{x\} \epsilon L(i) \cup L_i\}$
STEP 14:        Scan {DUT} and obtain C(X, DUT} for all $X \epsilon C_{i+1}$
STEP 15:        $L_{i+1} = \{X \mid X \epsilon C_{i+1}, S(X, DUT) \geq S_u\}$
STEP 16:        $PL_{i+1} = \{X \mid X \epsilon C_{i+1}, S(X, DUT) < S_u$ AND $S(X, DUT) \geq S_l\}$
STEP 17:        $L = L \cup L_{i+1}$, $PL = PL \cup PL_{i+1}$, $B = B \cup (C_{i+1} - L_{i+1} - PL_{i+1}\}$
STEP 18:        i = i + 1.  }
STEP 19:    If |T| + c > f then set c = 0; Otherwise set c = c + |T|. Update D (D= DUT).
```

Where T is a new data added to old data D. We assume that for each border or frequent set X in old data D, the count c(X, D) is already known from the previous stage. (We may assume starting from the empty set, with Φ as the only frequent set). The algorithm starts by scanning the new relation D and updating the counters of all large itemsets, pre-large itemsets and pre-border item sets (line 2-5). The new support of a set X is its count divided by the new total size. Note that since the size of the relation is now larger, some previous large or pre-large sets may not be large or pre-large any longer. Thus, the new large or prelarge and pre-border sets are determined (lines 7–9). Beforehand, the set of promoted pre-borders is determined (in step 6). A pre-border set X (of D) is said to be promoted pre-border if after the increment T its support reaches the minimum support threshold S_l, (and hence became large of pre-large item sets). Next, the candidates are generated (lines 13). Their count is obtained by scanning the entire database (line 14). Based on the count, the new large or pre-large sets and pre-border sets are determined (line 17). Candidate generation and counting works in a sequence of rounds, where in round i candidates of size i are generated and checked. The candidates of size i+1, denoted by C_{i+1}, are generated based on the new sets of size i (L_i), the promoted pre-borders of size i, and the old large or prelarge sets of size i. The notation PB(i) denotes the promoted pre-borders of size i. Similarly, L(i), denotes the large item sets of size i. The procedure is based on the fact, that a set need be considered as a candidate only if it has a subset that is a promoted pre-border.

3.4 AN EXAMPLE

In this section, an example is given to illustrate the proposed new incremental data mining algorithm. Suppose a database with eight transactions such as the one shown in Table 3.1 is to be mined. The database has two features, transaction identification (TID) and transaction description(Items). For *Sl*=30% and *Su*=50%, the sets of large itemsetsand pre-large itemsets for the given data are shown in Tables 3.2 and 3.3, respectively.

Table 3.1 - An original database with TID and Items

TID	Items
100	**ACD**
200	**BCE**
300	**ABCE**
400	**ABE**
500	**ABE**
600	**ACD**
700	**BCDE**
800	**BCE**

Table 3.2 -The Large itemsets for the original database

Large Itemsets					
1-item	Count	2 items	Count	3	Count
A	**5**	**BC**	**4**	**BCE**	**4**
B	**6**	**BE**	**6**		
C	**6**	**CE**	**4**		
E	**6**				

Table 3.3 -The Pre-Large itemsets for the original database

Pre- Large Itemsets					
1-	Count	2 items	Count	3 items	Count
D	**3**	**AB**	**3**	**ABE**	**3**
		AC	**3**		
		AE	**3**		
		CD	**3**		

Table 3.4 -The Pre-Border itemsets for the original database

Pre- Border Itemsets					
1-item	Count	2 items	Count	3	Count
		AD	**2**	**ABC**	**1**
		BD	**1**		
		DE	**1**		

Using a conventional mining algorithm such asApriori algorithm, all large itemsets with counts larger than or equal to 4 (8∗50%) are found, as shown in Table 3.2 and the all pre-large itemsets (which are not large itemsets) with their support count larger than or equal to 3 (8*30%) are found and shown in table 3.3. Table 3.4 shows the all the border itemsets in the given original database shown in table 3.1.

Table 3.5 - An incremental database with TID and Items

TID	Items
900	**ABCD**
1000	**BCD**
1100	**ABCDE**
1200	**BCD**

Assume the four new transactions shown in Table 3.5 are inserted after the initial data set is processed. The proposed incremental mining algorithm proceeds as follows. The variable *c* is initially set at 0.

STEP 1: The safety number *f* for new transactions is calculated as:

$f = (S_u - S_l) * d / 1\text{-}S_u = ((0.5 - 0.3) * 8 / (1-0.5) = 3.$

STEP2: Find the count of large itemsets, pre-large itemsets and border itemsets in T.

Table 3.6 -Count of Large itemsets in T

Large Itemsets					
1-item	Count	2 items	Count	3	Count
A	**2**	**BC**	**4**	**BCE**	**1**
B	**3**	**BE**	**1**		
C	**4**	**CE**	**1**		
E	**1**				

Table 3.7 - Count of Pre-Large itemsets in T

Pre- Large Itemsets					
1-	Count	2	Count	3	Count
D	**4**	**AB**	**2**	**ABE**	**1**
		AC	**2**		
		AE	**1**		
		CD	**4**		

Table 3.8 - Count of Pre-Border itemsets in T

Pre- Border Itemsets					
1-	Count	2 items	Count	3	Count
		AD	**2**	**ABC**	**2**
		BD	**4**		
		DE	**1**		

Table 3.9 - Promoted Pre-Border itemsets

Promoted Pre- Border Itemsets					
1-	Count	2 items	Count	3	Count
		AD	**4**		
		BD	**5**		

After updating all the sets the resultants large, pre-large and pre-border set are like the following:
L = {A, B, C, D, E, BC, BE, CD}
PL = {AB, AC, AD, BD, AE, CE, ABE, BCE}
Pre-Border = {DE, ABC, ABD, ACD, ACE, BCD}
As the value of f is less than the number of transactions T so the algorithm runs steps 12 to 19 and these steps runs twice. First time it verifies whether DE become large or pre-large and second time it verifies the remaining three length itemsets become large or pre-large and only ACE and BCD will move from border itemsets to pre-large itemsets. At the end the set of large, prelarge and pre-border itemsets are shown in the tables 3.10 to 3.11.

Table 3.10 -The Large itemsets for the updated database

Large Itemsets					
1-item	Count	2 items	Count	3	Count
A	**7**	**BC**	**8**		
B	**9**	**BE**	**7**		
C	**10**	**CD**	**7**		
D	**7**				
E	**7**				

Table 3.11 -The Pre-Large itemsets for the updated database

Pre- Large Itemsets					
1-	Count	2 items	Count	3 items	Count
		AB	**5**	**ACE**	**4**
		AC	**5**	**BCD**	**5**
		AE	**4**		
		AD	**4**		
		BD	**5**		

Table 3.12 -The Pre-Border itemsets for the updated database

Pre- Border Itemsets					
1-item	Count	2 items	Count	3	Count
		DE	**2**	**ABC**	**3**
				ABD	**2**
				ACD	**2**

In this section, we have proposed designed a novel, efficient, incremental mining algorithm based on pre-large itemset and border itemset.. Using two user-specified upper and low support thresholds, the pre-large itemsets act as a gap to avoid small itemsets becoming large in the updated database when transactions are inserted. Our proposed algorithm also retains the following features of the FUP algorithm

1. It avoids re-computing large itemsets that have already been discovered.
2. It focuses on newly inserted transactions, thus greatly reducing the number of candidate itemsets.
3. It uses a simple check to further filter the candidate itemsets in inserted transactions.

Moreover, the proposed algorithm can effectively handle different cases; in which itemsets are small in an original database but large in newly inserted transactions can be considered for scanning if and only if it becomes a pre-border itemset, although it does need additional storage space to record the pre-large itemsets. Note that the FUP algorithm needs to rescan databases to handle such cases. The proposed algorithm does not require rescanning of the original databases until a number of new transactions determined from the two support thresholds and the size of the database have been processed. If the size of the database grows larger, then the number of new transactions allowed before rescanning will be larger too. Therefore, as the database grows, our proposed approach becomes increasingly efficient. This characteristic is especially useful for real-world applications.

IV.RESULTS

All the datasets used in our experiments are synthetic datasets generated by using a random function. In each of the experiments we fixed an initial database, consisting of 100 transactions. We then added an *increment set*, again by randomly generating the transactions, and measured the performance of the algorithms when producing the frequent set for the combined data set.In our first experiment, we measured the performance of the algorithms with varying increment sizes. The support threshold was set to 8%. We varied the increment size from 25 records (25% the size of the original set) to 50 records (50%), in increments of 25.The average times are depicted in graphs (each point is averaged over ten experiments).The new incremental algorithm exhibits improved performance throughout the entire range. Theperformance gain ranges from over 30-fold for the small increments (25%), to 3–4 fold for very large (25-50%) increments.The main advantage of the algorithm is due to the small number of times the entiredatabase (old and new) must be scanned. Figure 7 depicts the average number of fulldatabase passes each of the algorithms required for the different increment sizes.
The first experiment we conducted on the synthetic database of size 100 and incremental size increases from 25, 30, 35, 40, 45, and 50, the number of items we considered in this data set is 15. The following are the statistics we obtained by applying this dataset on our proposed algorithm.

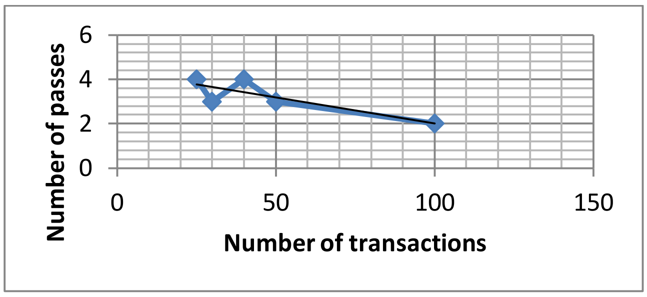

Fig 4.1 - #passes Vs # transactions on synthetic dataset -1

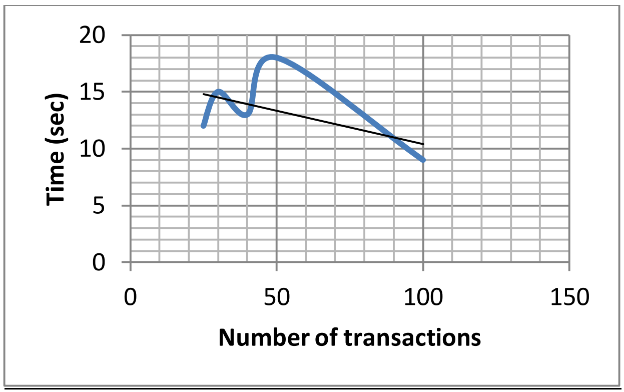

Fig 4.2 - Running time Vs # transactions on synthetic dataset -1

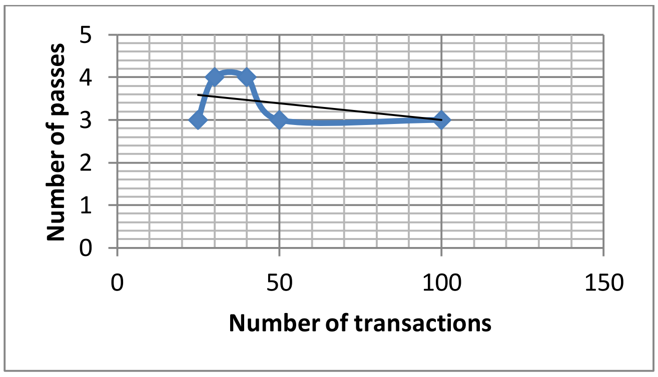

Fig 4.3 - #passes Vs # transactions on synthetic dataset -2

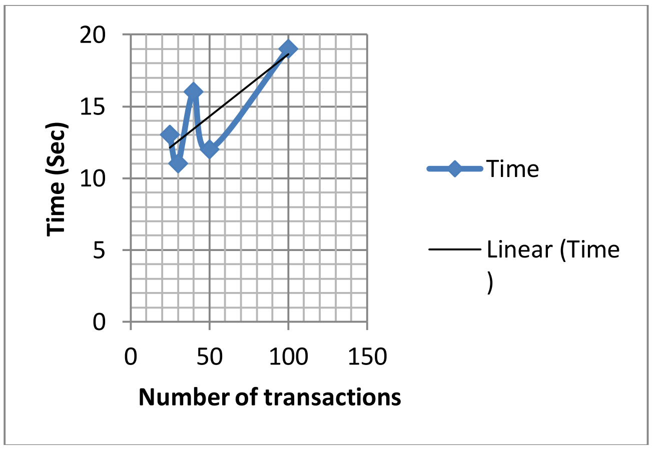

Fig 4.4 - Running time Vs # transactions on synthetic dataset -2

The graphs 4.1 to 4.4 show the algorithms scalable if the number of transactions and the increments are also increases. The running time of the algorithms are also shows the time it takes for executing the program is an acceptable one.

V. CONCLUSION

In this paper, we have presented the detailed literature survey on different incremental mining algorithms and also we studied the different cases need to be consider in designing the incremental mining algorithms. We proposed a novel, efficient, incremental mining algorithm based on pre-large itemset and border itemset. It uses the following thresholds.

1. Upper threshold
2. Lower threshold.

The pre-large itemsets act as a gap to avoid small itemsets becoming large in the updated database when transactions are inserted. Our proposed algorithm also retains the following features of the FUP algorithm. It avoids re-computing large itemsets that have already been discovered. It focuses on newly inserted transactions, thus greatly reducing the number of candidateitemsets. It uses a simple check to further filter the candidate itemsets in inserted transactions. Moreover, the proposed algorithm can effectively handle the different cases, in which itemsetsare small in an original database but large in newly inserted transactions can be considered for scanning if and only if it becomes a pre-border itemset; although it does need additional storage space to record the pre-large itemsets. Note that the FUP algorithm needs to rescan databases to handle such cases. The proposed algorithm does not require rescanning of the original databases until a number of new transactions determined from the two support thresholds and the size of the database have been processed. If the size of the database grows larger, then the number of new transactions allowed before rescanning will be larger too. Therefore, as the database grows, our proposed approach becomes increasingly efficient. This characteristic is especially useful for real-world applications.

VI. REFERENCES

[1]. Agrawal A. Imielinski T., and Swami A., "Mining Association Rules between Sets of Items in Large Databases," *In SIGMOD '93 Proceedings of the International Conference on Management of Data*, Washington, D.C., pp. 207-216, 1993.

[2]. Agrawal R., and Srikant R., "Fast Algorithms for Mining Association rules," *In VLDB'94 Proceedings of the International conference on Very Large Databases*, Santiago de Chile, Chile, pp. 487-499, 1994.

[3]. Agrawal R., and Srikant R., "Mining sequential patterns," *In ICDE'95 Proceedings of the 11th International Conference on Data Engineering*, Taipei, Taiwan, pp 3-14, March 1995.

[4]. Ahmed F., Tanbeer K., Jeong J., and Lee K., "Efficient Tree Structures for High Utility Pattern Mining in Incremental Databases", *IEEE transactionsons on Knowledge and Data Engineering*, vol.21, no.12, 2009.

[5]. Ale M., and Rossi H., "An approach to discovering temporal association rules," *In SAC'00 Proceedings of the 2000 ACM*

Symposium on Applied Computing, pp. 294-300, Como, Italy, 2000.

[6]. Bashar S., and Masseglia F., "Discovering the behaviours: time is an essential element of the context", *Knowledge and Informations systems*, Vol 28, no. 3, 311-331, August 2011.

[7]. Brigis T., Swinnen G., Vanhoof K., and Wets G., "Using Association Rules for Product Assortment Decessions: A Case Study," *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, San Diego, CA, USA, pp. 254-260, 1999.

[8]. Burdick D., Calimlim, M., and Gehrke J. "MAFIA: A Maximal Frequent Itemset Algorithm,"*IEEE transactionson Knowledge and Data Engineering*, Vol. 17, No. 11, 2005.

[9]. Calders T. and Goethals B., "Depth-first non-derivable itemset mining," *In SDM 2005 Proceedings of the fifth International Conference on Data Mining*, California, USA, pp. 250-261, 2005.

[10]. Chen F., and Li M., "A Hybrid method for discovering maximal frequent itemsets," *In FSKD 2008 Proceedings of the Fifth International Conference on Fuzzy Systems and Knowledge Discovery*, Jinan, Shandong, China, pp. 546-550, 2008.