International Journal on Recent and Innovation Trends in Computing and Communication
Volume: 3 Issue: 6

ISSN: 2321-8169
4267 - 4274

Mining Frequent Itemsets for Evolving Database Involving Insertion

Mrs. Ashlesha A. Jagdale
Department of Computer Engineering
Alard College of engineering
Savitribai phule Pune University
Marunje, Pune 58, INDIA
ashlesha.jagdale@gmail.com

Prof. Sonali Patil
Department of Computer Engineering
Alard College of engineering
Savitribai phule Pune University
Marunje, Pune 58, INDIA
sonalipatil@live.com

Abstract**—** Mining frequent itemsets is one of the popular task in data mining. There are many applications like location-based services, sensor monitoring systems, and data integration in which the content of transaction is uncertain in nature. This initiates the requirements of uncertain data mining. The frequent itemsets mining in uncertain transaction databases semantically and computationally differs from techniques applied to standard certain databases. The goal of proposed model is to deal with the problem of extracting frequent itemsets from evolving databases using Possible World Semantics (PWS). As evolving databases contains exponential number of possible worlds mining process can be modeled as Poisson Binomial Distribution (PBD). In this proposed work apriori-based PFI mining algorithm and approximate incremental mining algorithm are developed. An approximate incremental mining algorithm can efficiently and accurately discover frequent itemsets. Also, focus is on the issue of maintaining mining results for uncertain databases.

Keywords- *Approximate incremental mining algorithm, Apriori-based PFI mining algorithm, Evolving Database, Frequent Itemsets.*

*****

I. INTRODUCTION

The databases used in many important applications are often uncertain [1]. The locations of users obtained through RFID and GPS systems are not precise due to measurement errors. Fig. 1 shows an online marketplace application, which carries probabilistic information. The purchase behavior details of customers jack and mary are given. The value with each item represents the chance that a customer may buy that item in the near future. Users browsing histories provides probability values. For example, if jack visited the marketplace 10 times in the previous week, out of which video products were clicked six times, the marketplace may conclude that jack has a 60 percent chance of buying videos. The mining of evolving data has recently attracted research attention. Many algorithms for finding frequent item sets (i.e., sets of attribute values that appear together frequently in tuples) for evolving databases are developed. These algorithms can be applied to two uncertainty models: attribute uncertainty and tuple uncertainty, every tuple is associated with a probability to indicate whether it exists. The frequent item sets discovered from evolving data are naturally probabilistic, in order to reflect the confidence placed on the mining results. Fig. 2 shows a probabilistic frequent item set (pfi) extracted from fig. 1. A pfi is a set of attribute values that occurs frequently with a sufficiently high probability. In fig. 2, the support probability mass function (s-pmf) for the pfi {video} is shown. This is the pmf for the number of tuples (or support count) that contain an item set Under PWS, a database induces a set of possible worlds, each giving a support count for a given item set. Hence, the support of a frequent item set is described by a pmf. In Fig. 2, consider all possible worlds where item set {video} occurs twice, the corresponding probability is 1/6. The important problem is maintaining mining results for changing or evolving databases. The type of evolving data that we address here is about the appending, or insertion of a batch of tuples to the database. Tuple insertion is common in the many applications. For example, a GPS system may have to handle location values due to the registration of a new user.

Customer	Purchase Items
Jack	(Video:1/2), (food:1)
Mary	(Clothing:1),(video:1/3), (book:2/3)

Figure 1: Illustrating an uncertain database.

To summarize, a model-based algorithm, this can reduce the amount of effort of scanning the database for mining threshold-based PFIs. We also develop incremental mining algorithms, for extracting approximate PFIs. Both these algorithms can support attribute and tuple uncertainty models.

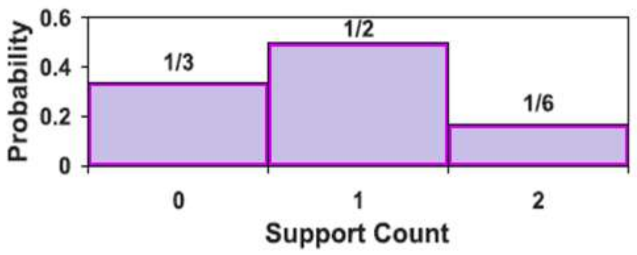


Figure 2: s-pmf of PFI {video}

II. RELATED WORK

Traditional mining algorithms work well for databases with precise values; it is not clear how they can be used to mine probabilistic data. Here algorithms are developed for extracting frequent item sets from evolving databases. Mining frequent itemsets is an important problem in data mining, and is the first step of deriving association rules. Hence, many efficient frequent itemset mining algorithms have been proposed.

Although these algorithms are developed based on the apriori framework, they can be considered for supporting other algorithms for handling uncertain data. For uncertain databases, Aggarwal et al. and Chui et al.[2] developed efficient frequent pattern mining algorithms based on the expected support counts of the patterns. Bernecker et al, Sun et al [4], and Yiu et al. [5] found that the use of expected support may render important patterns missing. Hence, they proposed to compute the probability that a pattern is frequent, and introduced the notion of PFI. S. Nirmala Devi [6] and B. Prathema discussed dynamic-programming based solutions to retrieve PFIs from attribute-uncertain databases. However, these algorithms compute exact probabilities, and verify that an item set is a PFI in O (n2) time.

Dynamic programming is not used in model-based algorithms and model-based algorithms are able to verify a PFI much faster (in O (n) time). Approximate algorithms for deriving threshold-based PFIs from tuple-uncertain data streams were developed. While Zhang et al. [5] only considered the extraction of singletons; these solution discovers patterns with more than one item. Recently, Sun et al. developed an exact threshold-based PFI mining algorithm. However, it does not support attribute-uncertain data. Proposed work examines a model-based approach for mining PFIs. Here, study how these algorithms can be extended to support the mining of evolving data.

Incremental mining framework is inspired by Fast UPdate algorithm (FUP). ZIGZAG [7] also examines the efficient maintenance of maximal frequent itemsets for databases that are constantly changing. CATS Tree [8], was introduced to maintain frequent item sets in evolving databases. Another structure, called CanTree [9], arranges tree nodes in an order that is not affected by changes in item frequency. The data structure is used to support mining on a changing database. This work proposes novel incremental mining algorithms for approximate PFI discovery. These algorithms can support attribute and tuple uncertainty models.

III. IMPLEMENTATION DETAILS

A. Problem Definition

Let V be a set of items. [1] In the attribute uncertainty model, each attribute value carries some uncertain information. Here, adopt the following variant: a database D contains n tuples, or transactions. Each transaction, t_j is associated with a set of items taken from V. Each item v Є V exists in t_j with an existential probability Pr (v Є t_j) Є (0, 1], which denotes the chance that v belongs to t_j. In Fig. 1, for instance, the existential probability of video in t_{Jack} is Pr(videoJack) = 1/2.

In the tuple uncertainty model, each tuple or transaction is associated with a probability value. The following variants are assumed: each transaction t_j Є D is associated with a set of items and an existential probability Pr(t_j) Є (0,1]; which indicates that t_j exists in D with probability Pr(t_j). The number of possible worlds for tuple uncertainty model is exponentially large.

For Probabilistic Frequent Itemset, let I is subset of the V be a set of items, or an itemset. The support of I, denoted by s (I), is the number of transactions in which I appears in a transaction database. For precise databases, s (I) is a single value. Let S (wj;I) be the support count of I in possible world wj. Then, the probability that s (I) has a value of i, denoted by PrI (i), is:

$$Pr^I(i) = \Sigma_{wj \in W,\, S(wj,\, I)=i} Pr(w_j) \quad (1)$$

Hence, PrI (i) (i=1, n) form a probability mass function (pmf) of s(I), where n is the size of D. Pr^I is the support pmf (or s-pmf) of I. For evolving databases, the frequentness probability of I, denoted by Pr_{freq} (I), is the probability that an item set is frequent. Pr_{freq} (I) can be expressed as:

$$Pr_{freq}(I) = \Sigma\ i \geq msc\ (D)\ Pr^I(i) \quad (2)$$

In Fig. 2, if minsup= 1, then msc (D) =2. Thus, Pr_{freq} ({video}) = Pr {video} (1) + Pr {video} (2) = 2/3.

TABLE I SUMMARY OF NOTATIONS

Notation	Description
D	An Uncertain Database of n tuples
V	The set of items that appear in d
v	An item, where v Є V
t_j	jth tuple in D
W	The set of all possible worlds
w_j	A possible world w_j Є W
I	An itemset, where I is subset of V
minsup	A real value between (0,1]
msc(D)	The minimal support count in D
s(I)	The support count of I in D
minprob	A real value between (0,1]
d	Delta database with n' tuples
D^+	A new database with n^+ tuples, D^+ = D U d

Using frequentness probabilities, it is possible to determine whether an item set I is frequent. This proposed work, adopt the definition in I is a Threshold-based PFI if its frequentness probability is larger than some user-defined threshold. Given a real value minprob Є (0,1], I is a threshold-based PFI, if

$Pr_{freq}(I)>=minprob$. Call minprob the frequentness probability threshold. Here, would like to mention the following theorem,

Theorem 1 (Antimonotonicity) - Let S and I be two itemsets. If S is subset of I, then $Pr_{freq}(S) >= Pr_{freq}(I)$.

Incremental mining algorithms, which enable Probabilistic Frequent Itemset (PFI) results to be refreshed. This reduces the need of re-executing the whole mining algorithm on the new database, which is often more expensive and unnecessary. Table I summarizes the symbols used in this paper.

B. *Mathematical Model*

Set Theory Analysis

A] Identify the set of data for Input
D= {d_1, d_2, d_3….}
Where 'D' is main set of data like d_1, d_2, d_3… d_n

B] Identify the probability
P= {p_1, p_2, p_3….}
Where 'P' is main set of probability like p_1, p_2, p_3… p_n

C] Identify the probability of FIS
PF= {pf_1, pf_2, pf_3….}
Where 'PF' is main set of probability Frequent Item Set like pf_1, pf_2, pf_3… pf_n

D] Identify the s-pmf
SF= {sf_1, sf_2, sf_3….}
Where 'SF' is main set of support probability mass function like sf_1, sf_2, sf_3… sf_n

E] Identify the Process of Generating Probability Frequent Item Set. Probability Frequent item sets Generation Steps uses minsup and minprob,
L is the main set of layer
L= {l1, l2, l3,…}
Incount is the main set of the input counts for layer
Incount= {Inc1, Inc2, Inc3, Inc4 …}
Outcount is the main set of the output counts for layer
Outcount = {Outc1, Outc2, Outc3, Outc4 …}
Check probability using following formula

$$minprob = 1 - F(msc(D) - 1, \mu_m).$$

DetectionRate is the main set of detection rate.
DetectionRate= {DR1, DR2, DR3…}
CandidateSet is the main set for candidateSet which helps for creating
CandidateSet= {cs1, cs2, cs3…}
Probabilistic frequent itemset is the main set for the detection of packets
Rule = {r1, r2, r3 …}

D] Identify the Process of Generate approximate incremental mining
In this algorithm dataset will update.
P= {Set of processes}
P = {P1, P2, P3,P4……}

If (dataset not updated) then
P1 = {e1, e2, e3,e4}
Where
{e1=i| i is to Generate Frequent Item Set}
{e2=j| j is to Check Probability}
If (dataset update) then
P1 = {e1, e2, e3}
Where
{e1=i| i is to Generate Frequent Item Set }
{e2=j| j is to Check Probability}
{e3=k|k is Update Result of updated Data Set}

G] Identify failure cases as FL
Failure occurs when –
FL= {F1, F2, F3…}
a) F1= {f| 'f' If no Data set is not formatted}
H] Identify success case SS:-
Success is defined as-
SS= {S1, S2, S3, S4}
b) S1={s| 's' If Data set is not formatted }
c) S2= {s| 's' if all operation done}

C. *Proposed System*

The proposed system architecture of mining evolving data is illustrated in Fig. 3. It consists of three phases as candidate generation, candidate pruning and PFI testing.

- **Candidate generation**- In the first iteration, size-1 item sets that can be 1-PFIs are obtained, using the PFIs discovered from D, as well as the delta database d. In subsequent iterations, this module produces size (k+1) candidate item sets, based on the k-PFIs found in the previous iteration. If no candidates are found, it halts.
- **Candidate pruning**- With the aid of d and the PFIs found from D, this module filters the candidate item sets that must not be a PFI.
- **PFI testing**- For item sets that cannot be pruned, they are tested to see whether they are the true PFIs. This involves the use of database D^+, as well as the s-pmfs of PFIs on D. A simple method of testing whether I is a threshold-based PFI, without computing its frequentness probability. Given the values of minsup and minprob, can test whether I is a threshold-based PFI, in three steps.

Step 1-Find a real number μ_m satisfying the equation:

$$\text{Minprob} = 1 - F(\text{msc}(D)-1, \mu_m) \qquad (3)$$

Above equation can be solved efficiently by employing numerical methods, Theorem 2.

Step 2-Use $\mu^I = \Sigma_{j=1\ldots n} p^I_j$ to compute μ^I. Notice that the database D has to be scanned once.

Step 3-If $\mu^I \geq \mu_m$, conclude that I is a PFI. Otherwise, I must not be a PFI. In order to verify whether I is a PFI, once μ_m is found, do not have to evaluate Pr_{freq} (I). Instead, compute μ^I in Step 2, which can be done in O (n) time.

Notice that in phases 1 and 2, only d and the PFIs of D are needed. Since these pieces of information are relatively small in size, they are usually not very expensive to evaluate. Phase 3 involves deriving the s-pmfs of item sets, with the use of D^+, and is thus more expensive than other phases. If phase 2 successfully removes a lot of candidates from consideration, the cost of executing phase 3 can be reduced.

The FP-Growth Algorithm is an alternative way to find frequent item sets without using candidate generations, thus improving performance. For so much it uses a divide-and-conquer strategy. The core of this method is the usage of a special data structure named frequent-pattern tree (FP-tree), which retains the item set association information.

In simple words, this algorithm works as follows: first it compresses the input database creating an FP-tree instance to represent frequent items. After this first step it divides the compressed database into a set of conditional databases, each one associated with one frequent pattern. Finally, each such database is mined separately. Using this strategy, the FP-Growth reduces the search costs looking for short patterns recursively and then concatenating them in the long frequent patterns, offering good selectivity.

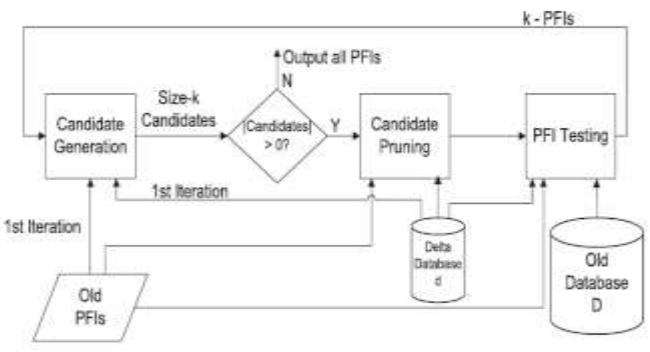


Figure 3: The System Architecture of mining uncertain data

D. *Apriori-based probabilistic frequent itemset Mining Algorithm*

The PFI testing techniques mentioned here are not associated with any specific threshold-based PFI mining algorithms. To incorporate these techniques to enhance the apriori algorithm, an important PFI mining algorithm is used. The Algorithm 1 uses the “bottom up” framework of the apriori: starting from k = 1, size-k PFIs are first generated. Using Theorem 1, size-(k+1) candidate item sets are derived from the k-PFIs, based on which the (k+1)-PFIs are found. The process goes on with larger k, until no larger candidate item sets can be discovered. The main difference of Algorithm 1 compared with that of traditional apriori is that all steps that require frequentness probability computation are replaced by PFI testing methods. Table 3 summarizes the symbols used in this paper.

Algorithm 1. Apriori-based probabilistic frequent itemset Mining Algorithm

```
Input: Uncertain database D, minsup, minprob
Output: All PFI: F={ $F_1$, $F_{2,\ldots\ldots,}$ $F_m$} // $F_k$ is set k-PFI
    1  begin
    2 $\mu_m$= MinExpSup (minsup, minprob, D);
    3 $C_1$.GenerateSingleItemCandidates (D);
    4 k=1; j=0;
    5 while $|C_k| \neq 0$ do
    6 foreach I Є $C_k$ do
    7 I.μ=0;
    8 while $(++j) \leq n$ and $|C_k| \neq 0$ do
    9 foreach I Є $C_k$ do
    10   I.μ = I.μ+ Pr(I is subset of $t_j$);
    11   if $I.\mu \geq \mu_m$ then
    12   $F_k$.push(I);
    13   $C_k$.remove(I);
    14   else if $j \geq n - \mu_m$ then
    15   if pruning(I, $\mu_m$, j, n) = = true then
    16   $C_k$.remove(I);
    17   $C_{k+1}$.GenerateCandidate($F_k$);
    18   18    k=k+1;   j=0
    19   return F;
    20   end
```

In particular, Algorithm 1 first computes μ_m (Line 2). For each candidate item set I generated on Lines 3 and 17 scan D and compute its μ^I_i (Line 10). If Lemma 1 is satisfied, then I put to the result (Lines 11-13). If Corollary 1 is satisfied, I pruned from the candidate item sets (Lines 14-16). This process goes on until no more candidates’ item sets are found.

Complexity- In Algorithm 1, each candidate item needs O (n) time to test whether it is a PFI. This is much faster than the Apriori, which verifies a PFI in $O(n^2)$ time. Moreover, since D is scanned once for all k-PFI candidates Ck, at most a total of n tuples is retrieved for each Ck.

E. *Approximate Incremental Mining Algorithm*

To efficiently maintain a set of PFIs in an evolving database, where new tuples, or transactions, are constantly appended to it, approximate incremental mining algorithm is used. Assume that every tuple has a timestamp attribute, which indicates the time that it is created. This timestamp is not used for mining; it is only used to differentiate new tuples from existing ones. Let D be the old database that contains n tuples, and d be a delta database of n' tuples, whose timestamps are larger than those of tuples in D. Let D^+ be a new database, which is a concatenation of the tuples in D and d, and D^+ has a size of $n^+ = n+n'$. Given the set of PFIs and their s-pmfs in D, goal is to discover PFIs on D^+, under the same minsup and minprob values used to mine the PFIs of D.

A simple way of obtaining PFIs from D+ is to simply rerun a PFI-mining algorithm on it. Hover, this approach is not very economical, since 1) running a PFI mining algorithm on a large database is not trivial; and 2) the same algorithm has to be frequently executed if a lot of update activities occur. If only a few tuples in d are appended to D, it may not be necessary to compute all PFIs on D^+ from scratch. This is because the PFIs found in D^+ should not be very different from those discovered in D. Based on this intuition, design an incremental mining algorithm that finds PFIs in D^+, without rerunning a complete PFI algorithm. This algorithm works well when the size of d is very small compared with that of D; nevertheless, it works with any size of d. Next discussed framework of solution, which discovers exact PFIs in D^+, based on the PFIs found in D.

As discussed before, model-based algorithm enables PFIs to be accurately and quickly discovered. Now investigate how to extend it to retrieve PFIs from evolving data. This extension is also called as approximate uncertain Fast UPdate algorithm (uFUPapp). Algorithm 2 describes the details. In Line 3, the candidates in C^+_1 are generated (Phase 1). In Lines 5-7, the parameter values used for pruning are computed. In the kth iteration (Lines 8-15), some candidates in the set C^+_k are pruned (Phase 2; Line 9), while the remaining ones are tested (Phase 3; Line 11). In Line 14, size-(k+1) candidates are generated by using the k-PFIs found. When no more candidates are left (Line 8), the algorithm outputs F^+, which contains PFIs of different sizes (Line 16).

Since the model-based approach supports both tuple and attribute uncertainty, the uFUPapp algorithm, which adopts the model-based approach, can also be used in both data models. Also remark that uFUPapp is generally faster, since less time is needed to test approximate PFIs than exact PFIs. uFUPapp is highly efficient and accurate.

Algorithm 2. Approximate uncertain Fast UPdate algorithm (uFUPapp)

Input: Uncertain database D, d, F^D, minsup, minprob

Output: Approximate PFIs in D: $F^+ =\{ F_1^+, F_2^+,\ldots, F_m^+\}$

```
1  begin
2  $F^+ = Ø$
3  $C_1^+$.GenerateSingleton (d, $F_1^D$);
4  k=1;
5  $\mu_m(D^+) = $ MinExpSup(minsup,minprob,$D^+$);
6  $\mu_m(D) = $ MinExpSup(minsup,minprob,D);
7  $\mu_m^- = \mu_m(D^+) - \mu_m(D)$
8  while $|C_k| \neq 0$ do
9   $C_k^+$.prune( d , $F_k^D$, $\mu_m^-$ );
10 if $C_k^+ \neq 0$ then
11 $Fk^+ \leftarrow C_k^+$.Test(D, d, $F_k^D$, $\mu_m(D^+)$ );
12 else
13   break;
14 $Ck_{+1}^+$.GenerateCandidate($F_k^+$);
15 k=k+1;
16 return $F^+ =\{ F_1^+, F_2^+,\ldots, F_{k-1}^+\}$;
17 end
```

Phase 1: Candidate Generation

Considered two cases of generating size-k candidate item sets in this phase: 1) k = 1 and 2) k > 1.

Case 1: k = 1. Invoke Generate Singleton, in Line 3 of Algorithm 2. This subroutine simply returns the union of all single items in d and the 1-PFIs of D (i.e., F_1^D), as the set of size-1 candidate item sets (C_1^+). To understand why Generate Singleton covers all possible size-1 candidates, first notice that if an item set is a 1-PFI in D, it should naturally be considered as a candidate itemset in D+. Then claim that it suffices to include all single items of d to C_1^+ , using the Lemma 3.

Case 2: k > 1. Uses typical Apriori-gen method to generate size-k candidates from (k-1)PFIs. Subroutine GenerateCandidate (Line 10 in Algorithm 2) performs the following: for any two (k -1) PFIs, I and I', if there is only one item that differentiates I from I', a candidate item set I U I' is produced. Using Lemma 1 (antimonotonicity), can easily show that GenerateCandidate produces all size-k candidates.

Next, examine how some of the candidates generated in this phase can be pruned.

Phase 2: Candidate Pruning

Let μ^I (DB) be the expected value of random variable X^I in DB, where DB is any of the databases D, d, or D^+. Also, let μ_m (DB) be a real value that satisfies (3) in DB. The theorem 3 is used by Phase 2. In Algorithm 2, lines 5-7 compute the value of μ_m^- . Then, in Line 9, subroutine Prune uses Theorem 3 to remove candidates that are not PFIs in D, and whose μ^I (d) values do not exceed μ_m^- . Since Prune needs to scan d once to obtain μ^I (d), the cost of pruning an item set is O (n').

Phase 3: PFI Testing

The objective of this phase is to verify whether an item set in Ck+ is a true k-PFI. Subroutine Test (Line 11, Algorithm 2) is invoked to perform this task: for each item set I, it first computes μI (D+). If this value is not less than μm(D+), I is judged to be a PFI of D+.

A simple way of computing μI (D^+) is to scan the tuples in D+ once. This can be costly, if many candidates need to be tested. Suppose known the μI (D) value of an item set I, which is a PFI of D. First evaluate μI (d), by scanning d once.

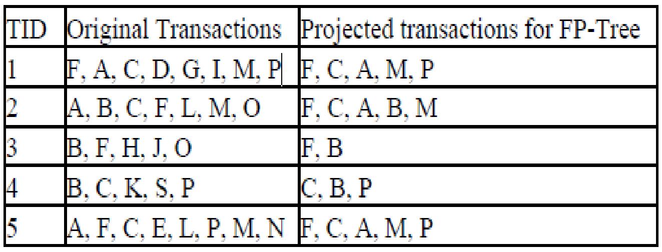
The value of µI (D^+) can be then obtained by adding these two values together. If d is small, scanning tuples in d is fast, and so computing µI ($D^{+)}$ can be more efficient. In uFUPapp, it saves the µI (D) values of all the PFIs discovered in D, so that they can later be used to derive PFIs for D^+.

F. *CATS Tree Algorithm*

CATS tree is a prefix tree. In CATS tree path from the root to the leaves represent set of transactions. After constructing CATS tree from database, it enables frequent pattern mining with different minimum supports without rebuild the whole tree structure

An extension [7] of FP-Tree is CATS Tree. Table 2 illustrates the construction of a CATS Tree. Initially, the CATS Tree is empty. Transaction 1 (F, A, C, D, G, I, M, P) is added as it is. Transaction 2 (A, B, C, F, L, M, and O) is added, As shown in Fig 4, common items, F, A, C, are extracted from Transaction 2 and are merged with the existing tree. Item D is not contained in Transaction 2, common items could be found underneath node D. Item M is common. However, Transaction 2 cannot be merged directly at node M because it would violate the structure of CATS tree that the frequency of a parent node must be greater than the sum of its children's frequencies. As shown in Fig 4 Node M of CATS Tree is swapped in front of node D. And it is merged with the transaction. After that, there is no more common item. The remaining portion of Transaction 2 is added to node M.

TABLE 2: SAMPLE DATABASE



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

Transaction 3 (B, F, H, J, O) is added in Fig. 4. Item F of Transaction 3 is merged. Since the frequency of node A is the same as that of node F, the search for other possible merge nodes continues along the branch. It passes through node A, C, and M and finally, reaches node B. Even though Transaction 3 also contains an item B, but the frequency of node B is smaller than that of node M, the remaining of the transaction is inserted as a new branch at node F.

When Transaction 4 (B, C, K, S, P) is added, there is no common item. Transaction 5 (A, F, C, E, L, P, M, N) is added; In Figure.5, F, A, C, and M are merged. The search for common items continues along the path. Item P is common in both the tree path and Transaction 5. This triggers swapping of node P to the front of [7] node D. Item P is merged, there is no more common item. The remainders of Transaction 5 are inserted as a new branch at node P.

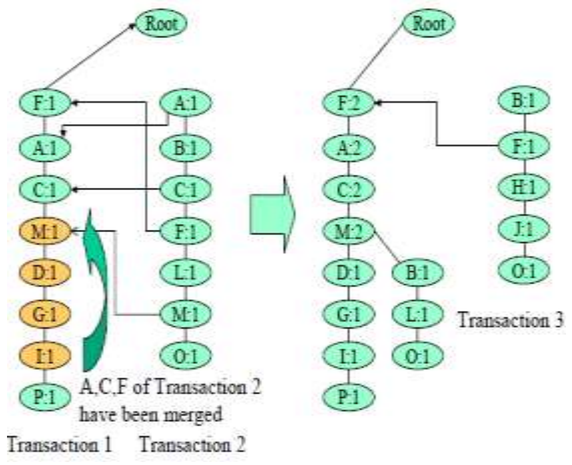


Figure 4: Insertion of Transaction 1, 2 & 3

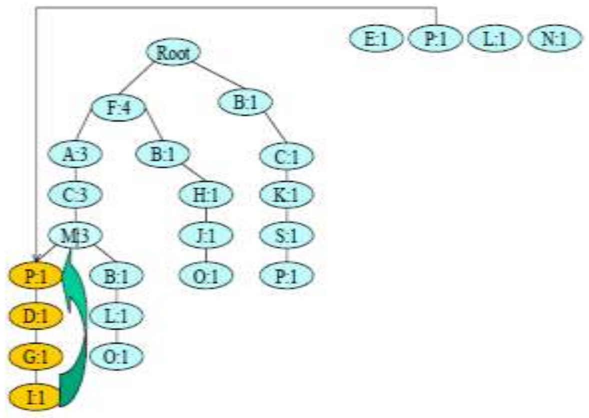


Figure 5: Insertion of Transaction 5

Construction of CATS Tree [7] requires only a single data scan. Thus, it is not optimal since there is no preliminary analysis before this single data scans. At the root level new transactions are added. At each level, items of the transaction are compared with those of children nodes. If the same items exist in both the new transaction and that of the children nodes, the transaction is merged with the node at the highest frequency. The frequency of the node is incremented. The remainder of the transaction is added to the merged nodes and the process is repeated recursively until all common items are found. Any remaining items of the transaction are added as a new branch to the last merged node. CATS Tree Builder has to consider not only the immediate items of that level, but also all possible descendants. Once the frequency of the new transaction is added, the frequency of a descendant node can become larger than that of its ancestor. If the frequency becomes larger, the descendant has to swap in front of its previous ancestor to maintain the structural integrity of CATS Tree.

Algorithm: **CATS Tree Builder**
Input: set of transactions
Output: CATS Tree

```
PROCEDURE CATSTreeBuilder(input set S)
for all transactions $t \in S$
for all $i \in t$
i.(frequency in header)++;
root.add ($t$);
PROCEDURE add(transaction $t$)
if (*this*.children $\cap\, t \neq \varnothing$)
child node. Merge($t$);
else if (*this*. Descendant $\cap\, t \neq \varnothing$))
swap descendant node and split child
Node if necessary;
Descendant. Merge($t$);
else
this. children $\leftarrow t$;
Reposition the merged node if necessary;
PROCEDURE merge(transaction $t$)
this. Frequency++;
remove *this*.item from $t$;
18. node.add($t$);
```

IV. Result

The data set contains any user defined numerical data, Attributes in one tuple are separated by comma the default value of minsup is 20 percent. To test the incremental mining algorithm, original database is used as the old database D, and the new added tuples are considered as the delta database d.

By comparing the performance of apriori algorithm, approximate incremental mining algorithm and CATS tree algorithm. Notice that CATS tree is faster than approximate incremental mining algorithm and approximate incremental mining algorithm is faster than apriori algorithm. Fig. 6 shows the Frequent Itemsets vs. Execution time.

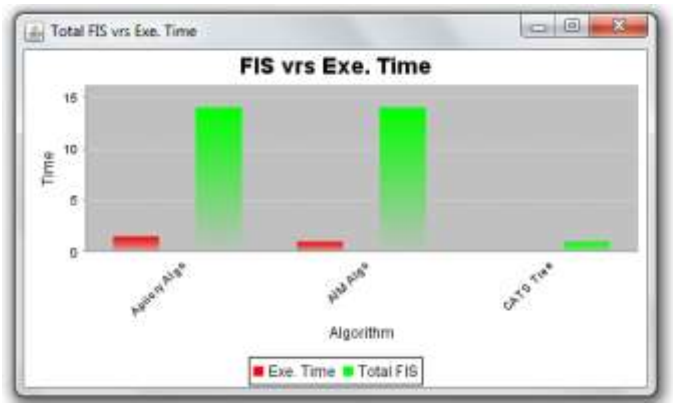


Figure 6: Frequent Itemsets vs Execution time

Fig. 7 shows the execution time of apriori, approximate incremental mining and CATS tree algorithm.

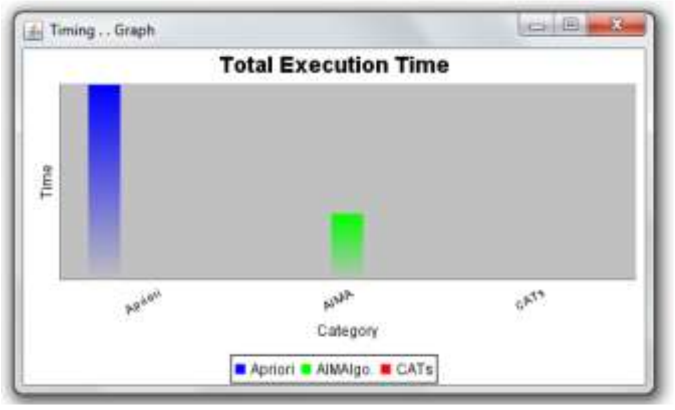


Figure 7: Efficiency of apriori, approximate incremental mining and CATS tree algorithm

V. Conclusion

Now a day's many applications contain uncertain data that is changing data. By considering high demands of user on uncertain data, mining uncertain databases becomes a popular task in data mining. In this proposed work apriori-based PFI mining algorithm and approximate incremental mining algorithm are developed.

The model-based approach to extract threshold-based PFIs from large evolving databases is used. Its main idea is to approximate the s-pmf of a PFI by some common probability model, so that a PFI can be verified quickly. Approximate incremental mining algorithm for retrieving PFIs from evolving databases is highly efficient and accurate than exact incremental mining algorithm. Apriori based PFI mining and approximate incremental mining algorithms support both attribute and tuple uncertain data model.

Since an uncertain database contains an exponential number of possible worlds, this issue is technically challenging. Also remark that approximate incremental mining algorithm is generally faster, since less time is needed to test approximate PFIs than exact PFIs.

Acknowledgment

I would like to express my sincere thanks of gratitude to my esteemed guide Prof. Sonali Patil for her guidance and suggestions during this work. I am really thankful to her.

I am thankful to Prof. Priyadarshani Kalokhe, HOD of computer Dept for providing me the necessary lab facility and software. I am thankful to all who have directly or indirectly guided and helped me in delivery of this work.

Finally I would also like to thank friends who helped me a lot in finishing this work within the limited time.

References

[1] Liang Wang, David Wai-Lok Cheung, Reynold Cheng, Member, IEEE, Sau Dan Lee, and Xuan S. Yang,"Efficient mining of frequent item sets on large uncertain databases," *IEEE*

Transactions On Knowledge And Data Engineering, vol. 24, no. 12, pp. 5–6, Dec. 2012.

[2] C. Aggarwal and P. Yu,"A survey of uncertain data algorithms and applicatoins," *IEEE Trans Knowledge and Data Eng.*, vol. 21, no. 5, pp. 609-623, May. 2009.

[3] M. Yiu, N. Mamoulis, X. Dai, Y. Tao, and M. Vaitis, "Efficient evaluation of probabilistic advanced spatial queries on existential uncertain data," *IEEE Trans Knowledge and Data Eng.*,vol. 21, no. 9, pp. 108-122, Jan. 2009.

[4] L. Sun, R. Cheng, D.W. Cheung, and J. Cheng, "Mining uncertain data with probabilistic guarantees," Proc. 16th ACM SIGKDD International Conference Knowledge Discovery and Data Mining, 2010.

[5] Q. Zhang, F. Li, and K. Yi, "Finding frequent itemsets in probabilistic data," Proc. ACM SIGMOD International Conference Management of Data, 2008.

[6] S. Nirmala Devi, B. Prathema,"A Model Based Approach for Extracting Frequent Item Sets on Large Uncertain Databases," Sree Sowdambika College of Engineering, Virudhunagar, India, pp. 2-3.

[7] William Cheung and Osmar R. Zaïane,"Incremental Mining of Frequent Patterns Without Candidate Generation or Support Constraint", University of Alberta, *Edmonton, Canada* ,pp.5-6.

[8] A. Veloso, W. Meira Jr., M. de Carvalho, B. Poˆ ssas, S. Parthasarathy, and M.J. Zaki, "Mining Frequent Itemsets in Evolving Databases," Proc. Second SIAM International Conference Data Mining, 2002.

[9] W. Cheung and O.R. Zaiane, "Incremental Mining of Frequent Patterns without Candidate Generation or Support Constraint," Proc. Seventh Int'l Database Eng. and Applications Symp. (IDEAS), 2003.

[10] C.K.S. Leung, Q.I. Khan, and T. Hoque, "Cantree: A Tree Structure for Efficient Incremental Mining of Frequent Patterns," Proc. IEEE Fifth International Conf. Data Mining (ICDM), 2005.

[11] H. Cheng, P. Yu, and J. Han, "Approximate Frequent Itemset Mining in the Presence of Random Noise," Proc. Soft Computing for Knowledge Discovery and Data Mining, pp. 363-389, 2008.

[12] R. Cheng, D. Kalashnikov, and S. Prabhakar, "Evaluating Probabilistic Queries over Imprecise Data," Proc. ACM SIGMOD International Conf. Management of Data, 2003.

[13] 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.

[14] T. Jayram et al., "Avatar Information Extraction System," IEEE Data Eng. Bull., vol. 29, no. 1, pp. 40-48, Mar. 2006.

[15] S. Tsang, B. Kao, K.Y. Yip, W.-S. Ho, and S.D. Lee., "Decision Trees for Uncertain Data," Proc. IEEE Int'l Conf. Data Eng. (ICDE), 2009.

[16] M. Yiu, N. Mamoulis, X. Dai, Y. Tao, and M. Vaitis, "Efficient Evaluation of Probabilistic Advanced Spatial Queries on Existentially Uncertain Data," IEEE Trans Knowledge and Data Eng., vol. 21, no. 9, pp. 108-122, Jan. 2009.

[17] O. Benjelloun, A.D. Sarma, A. Halevy, and J. Widom, "ULDBs: Databases with Uncertainty and Lineage," Proc. 32nd International Conference Very Large Data Bases (VLDB), 2006.

[18] C.-K. Chui et al, "Mining frequent itemsets from uncertain data," In *Proc. PAKDD*, pp. 47‑58, 2007.

[19] C.K.-S. Leung et al. "A tree-based approach for frequent pattern mining from uncertain data," In *Proc. PAKDD*, pp. 653‑661, 2008.

[20] C.K.-S. Leung et al, "Mining uncertain data for frequent itemsets that satisfy aggregate constraints," In Proc. ACM SAC, pp. 1034‑1038, 2010.