RESEARCH ARTICLE OPEN ACCESS

Discovering Global and Local Temporal Frequent Patterns: An Optimized Approach

Sheel Shalini [1], Nayancy Kumari [2]

[1] Dept. of Comp. Sc. and Engineering, Birla Institute of Technology Mesra, Patna

[2] Dept. of Comp. Sc. and Engineering, Birla Institute of Technology Mesra, Ranchi

ABSTRACT

In a dynamic database, discovery and maintenance of frequent patterns and rules is a critical issue. To address this issue, several algorithms for incremental mining of temporal frequent patterns like ITARM, IndxTAR, etc. have been developed. However, these incremental algorithms suffer from performance bottlenecks due to larger candidate sets, excessive I/O and extensive computation, huge memory usage, etc. In this paper, an incremental temporal frequent pattern mining algorithm is proposed that discovers temporal associations among the items in reasonably lesser time in comparison to other similar algorithms. The proposed algorithm identifies changes in the behavior of data with time and discovers global as well as local frequent-patterns. It eliminates anomalies of earlier algorithms by applying a pruning technique in incremental mining. The proposed algorithm selects a few frequent 2-itemsets from an incremental database that have the probability of being frequent in the entire database that reduces the number of candidate 2-itemsets. Besides, pruning candidates at the time of candidate generation eliminates the chance of generating redundant supersets and powersets from infrequent itemsets. Consequently, search space is reduced to a great extent. The proposed algorithm considers negative border itemsets in mining incremental database; therefore, it utilizes two support measures for mining frequent and pre-frequent itemsets. Application of pruning technique in the candidate generation process is a novel approach that reduces candidates, making it better than existing incremental mining algorithms. The result of experiments also validates its better performance over other incremental mining algorithms.

Keywords **—** Temporal frequent pattern, Incremental mining, Candidate Pruning, Global and local patterns, Dense and sparse dataset

I. INTRODUCTION

Temporal association rule mining is a data mining technique that discovers association rules within a specified time period. Temporal association rules are of the form: $(X \rightarrow Y)^{TP}$ where, X and Y are items and TP is specified time period. The rules generated by mining a temporal database is valuable for identifying time-dependent associations between items [1]. and for making decisions. The abundance of temporal data has spurred researchers to uncover various types of time-variant patterns and regularities hidden within these databases. Temporal association rule mining [2-3] enhances strategies and decision-making capabilities by acquiring temporal knowledge [4-5]. Thus, it has gained importance in discovering co-occurrence relationships in different domains like Cancer treatment [5], Fault detection to enhance performance[6], anomaly detection[7], Outlier detection[8], crime detection [9], network intrusion detection [01], retail business [11], etc.

In real-life applications, databases often exhibit dynamic behavior. In such an evolving database, identification and maintenance of rules becomes an issue of paramount importance as it changes over time. Generally, new transactions are updated at a regular interval. With each newly added transaction, a deviation in the behavior of the database may occur. Due to which it is most likely that the patterns of associations are altered after the update of the database. Some new patterns emerge and some of the existing patterns become void in the updated database. Therefore, this research work incorporates an innovative idea that can be pragmatically implemented in an incremental database to identify global patterns and local patterns.

Many studies have been performed to suggest a method that needs minimum time and cost in mining dynamic database. Several algorithms for mining temporal association rules in an incremental database have been developed. Generally, incremental mining algorithms either follow level-wise approaches [12-14] or pattern-growth concept [15-17] to maintain the reliability of patterns and rules discovered. Cheung et al. [18] have introduced an incremental mining technique for the maintenance of patterns in an updated database. Subsequently, the concept of rule maintenance was extended by employing a sliding window filtering approach. However these algorithms are plagued by several limitations, such as their ability to only uncover global patterns, meaning patterns that are frequent during their maximum common exhibition period (MCP). Additionally, other issued include redundancy within rule sets, large candidate itemsets repeated scanning, and substantial requirements for internal memory space.

Frequent pattern mining plays a vital role in temporal association rule mining as frequent patterns identified within temporal databases serve as the basis for generating temporal association rules. The present paper has adopted an approach with intent to generate temporal frequent patterns. It eliminates redundancy in a frequent pattern set that was observed in previous mining techniques [19-20]. The proposed candidate generation based algorithm 'Incremental temporal frequent pattern mining (ITFPM)' works on the

sliding window concept and explores temporal frequent itemsets. It incorporates a negative border concept to identify frequent and pre-frequent itemsets. Experimental results indicate that the proposed method yields more precise (non-redundant) temporal frequent patterns compared to traditional approaches. Moreover, the inclusion of the pruning technique in the suggested method reduces the computational complexity more effectively than traditional approaches. The candidates generated by the ITFPM are significantly smaller than those generated by earlier state-of-the-art algorithms. Thus, the algorithm resolves the issue of performance bottleneck to a great extent. The proposed approach is more versatile and is capable of dealing with different datasets.

The major features of the proposed work are;

- The proposed algorithm discovers frequent patterns along with time period.
- The introduction of pruning technique in incremental mining, leading to the reduction of candidate set is a novel idea presented in the paper.
- Further, the algorithm reduces search space by verifying and selecting only those frequent 2-itemsets that has a high support count thus expected to be frequent in its MCP.
- Algorithm discovers global as well as local frequent-patterns (newly frequent-patterns). Hence, it is capable of mining temporal frequent itemsets from seasonal and publication databases also.

The rest of the paper is structured as follows: Section 2 renders related works. Section 3 delves into the proposed algorithm and its implementation details.. Further, Section 4 presents a comprehensive experimental analysis of the proposed algorithm on various datasets. Lastly, Section 5 provides concluding remarks.

II. RELATED WORKS

To deal with the maintenance of rules in a dynamic database, initially, some Apriori based algorithms [21] were proposed [18-12]. FUP (stands for Fast Update) algorithm [18] was first among rule maintenance algorithms. The FUP algorithm uses the large itemsets from the original database to discover large itemsets in the updated database. The original database is scanned solely to identify new large itemsets. Furthermore, updated database is decreased in each cycle by removing infrequent items from transactions within updated database. However, FUP does not lower the need of multiple scanning of the original database. Another incremental mining algorithm called FUP2 [12], which is capable of updating association rule in case of insertion or deletion of transactions in the original database. It also utilizes the mined result from previous mining to reduce computation time and cost.

The concept of the sliding window was introduced by [22] in a publication database to deal with items having various exhibition period. The sliding window filtering (SWF) approach splits the database into small segment and explores frequent patterns progressively from one segment to another. SWF algorithm applies a cumulative filtering threshold to generate candidate 2-itemsets and implements a scan reduction technique for generating candidate k-itemsets thereby reducing CPU cost and memory usage. The SWF algorithm performs well on a sparse dataset. However, in dense databases, its performance declines due to higher associations, and there is always a chance of combinatorial explosion.

Many incremental algorithms [23-24] employed a sliding window filtering approach to discover frequent patterns. The New Fast Update Method (NFUP) algorithm [23], an extension of FUP[18], aims to minimize the database scanning. It achieves this by logically partitioning the incremental database based on predetermined time intervals and scanning these partitions in reverse order, thus minimizing the overall number of scans. It utilizes the frequent patterns generated in the original database and eliminates the need to scan the original database. Nevertheless, the NFUP algorithm follows the Apriori algorithm concept; hence it suffers from large candidate generation. TWo end AssocIation miNer (Twain) algorithm proposed in [24] divides the incremental database according to the pre-defined time period and finds maximum frequent with a more precise exhibition period.

Nevertheless, the algorithm compares the end time of MCP of items with current partition time, MCP of itemsets must be known in advance. ITARM algorithm [19] generates candidate 2-itemsets by combining the candidate 2-itemsets from previous mining and present partition. Scanning of the original database is conducted once to identify all large itemsets, regardless of whether there are any new frequent itemset in incremental database. If there is a non-trivial difference between original and incremental database, the algorithm may suffer from large candidate generation problem.

An algorithm proposed by Li et al. [25] called a three-way decision update pattern (TDUP), uses three support thresholds, one to find frequent itemsets and rest two to divide the itemsets into three sections. TDUP updates frequent itemsets real-time. It employs a synchronization mechanism for amending the variation caused by TDUP. This mechanism triggers at regular intervals to re-compute the frequent itemset offline.

To reduce execution time, FP-tree [26] like algorithms for incremental mining [27, 28,29] has been suggested by different researchers. Dafa-Alla et al. [28] have used the Apriori-TFP-tree [30] structure for developing an algorithm IMTAR to explore temporal association rules in publication databases. It creates T-tree for storing itemsets information and P-tree for stores partial support for itemsets. The algorithm builds the tree partition wise in one scan. Since the algorithm first creates the entire tree and then filters out infrequent itemsets; its memory requirement is very large.

To maintain the large itemsets against the incremental data and reduce the number of the scan, various algorithms [31] based on the concept of negative border [32], have been developed. Hong et al. [13] suggested two support thresholds and reduced database rescan by implementing a safety

threshold measure. Safety threshold is determined by lower support, upper support and original database size. Theoretically, an itemset from an incremental database cannot qualify as a large itemset in the entire database until the number of transactions in the incremental database surpasses the safety threshold. Further, this concept was extended in the Pre-FUFP maintenance algorithm [33] that maintains information from a previous scan on FUFP-tree. It keeps the results of previous mining to avoid recounting of large and pre-large itemsets of previous database that causes an increase in its memory requirement. Fouad et al. [20] have suggested an incremental algorithm called IndxTAR that explores frequent temporal itemsets in items' lifetime period. It is extended from the FP-tree [26] concept and used a data structure TIndex that contains tree-structure. The tree is constructed in one scan considering each item of the transactions and items temporal information and traverses k times to count the support of k-itemsets. It also utilizes the frequent itemsets of previous mining. Unfortunately, its memory requirement increases immensely with every update, and so the algorithm is not suitable for the large database with the number of partitions.

III. INCREMENTAL TEMPORAL FREQUENT PATTERN MINING

In this section, problem definition and terminologies associated with temporal frequent pattern mining are discussed.

A. Problem Definition and Basic Concept

In a growing database, rule maintenance is an issue of paramount importance due to the fact that with every newly added transaction, a deviation in behavior of database may occur. The paper incorporates an innovative idea that can pragmatically be implemented on incremental database to maintain global patterns (itemsets frequent in their MCP) and to explore local frequent patterns (itemsets frequent in a short time period). It scans the entire updated database just the once. Proposed algorithm ITFPM uses minimum support and probable support thresholds to recognize frequent and especially pre-frequent itemsets, and eliminate outliers (items with lower support) at the very beginning. Pre-frequent itemsets are negative border sets that have a greater probability of becoming frequent following a database update.

Definition 1 (Temporal frequent pattern mining): Frequent pattern mining [34] is discovering associations among the items in a database. Temporal frequent pattern mining, an extension of frequent pattern mining, is discovery of time-dependent associations [x] and represented as $(X \rightarrow Y)^{TP}$.

Definition 2 (Temporal Frequent Patterns): Set of items or Pattern that appears together frequently in a database in a specified time frame are called temporal frequent patterns. For instance, a collection of items, like "Cake", "Candles" and "Christmas Tree" that frequently occur together in a transaction dataset during December, can be called temporal frequent itemset or temporal frequent pattern for December. Temporal pattern mining can be conducted on a binary database where items are represented as present(true) or absent(false).

The temporal frequent patterns can be stated mathematically as:

Given a database D with transactions (t1, t2, ..., tn), discovery of all patterns P that are present in at least a fraction σ of the transactions in a particular time interval T. The fraction σ denotes minimum frequency needed to classify a pattern frequent, and is referred to as the minimum support threshold. Support value of a pattern can be calculated as:

$$\text{Support}(P,T) = \frac{\text{frequency}(P)}{n} \quad (1)$$

Patterns with frequency or support value more or equal to minimum support thresholds are

$$\text{Support}(P,T) = \frac{\text{frequency}(P)}{n} \geq \sigma \quad (2)$$

Temporal frequent itemset or temporal frequent pattern are indeed synonymous terms, both referring to frequent combinations of items. To streamline the discussion, the attributes of the database are referred to as items and term temporal frequent itemset is consistently employed for convenience.

Let a set of items database be denoted as I, and each transaction t⊆ I. If σ and δ are minimum support threshold and minimum probable support (negative border) thresholds.

Definition 3 (Temporal Frequent Itemset): Temporal Frequent Itemset (TFI) is an itemset that is frequent in a time interval (s,n). An itemset $X^{s,n}$ can be expressed as TFI under the constraint:

a) $X \subseteq I$

b) From partition P_s to P_n, it satisfies

$$\frac{\cup_{p=s}^{n} |\{t|X \subseteq t\}|}{\cup_{p=s}^{n} |P_p|} \geq \sigma \quad (3)$$

i.e., a ratio of the sum of all transactions containing X and sum of all transactions in time interval (s,n) is not less than minimum support σ.

Definition 4(Pre-frequent Itemset): Pre-frequent Itemset (PFI) is an itemset that is expected to be frequent in a time interval (s, n). An itemset $X^{s,n}$ can be expressed as PFI under the constraint:

a) $X \subseteq I$

b) From partition P_s to P_n, it satisfies

$$\frac{\cup_{p=s}^{n} |\{t|X \subseteq t\}|}{\cup_{p=s}^{n} |P_p|} \geq \delta \quad (4)$$

i.e., a ratio of the sum of all transactions containing X and sum of all transactions in time interval (s,n) is not less than minimum probable support δ.

Hence, the relative σ and δ values in the updated database are

$$\sigma' = \sigma \times \cup_{p=s}^{n} |P_p| \quad (5)$$

$$\delta' = \delta \times \cup_{p=s}^{n} |P_p| \quad (6)$$

Since the anti-monotone property cannot hold on updating database, following properties are proposed and introduced in the algorithm to reduce candidate itemsets.

Property 1. In an updated database, subsets of a temporal frequent itemset remain frequent within the same time interval. For example, if an itemset X is frequent in the updated database in a time interval $[s,t]$, then $\forall x \subseteq X$ must be frequent in the same time interval $[s,t]$.

For $\forall x \in X, \quad Support(x) \geq Support(X)$

Property 2. . In an updated database, if an itemset contains at least one infrequent subset within a specified time interval, then the items itself is always infrequent in that same time interval. For example, if an itemset X is infrequent in the updated database in the time interval $[s,t]$, then $\exists\ x \subseteq X$ must be infrequent in the time interval $[s, t]$.

Property 3. If itemsets are frequent in different time duration t_a and t_b with no overlapping of time, then their superset cannot be frequent.

The prime objective of the algorithm is to minimize search space (candidate k-itemsets) and maintain maximal frequent itemsets. Unlike traditional incremental algorithms, the proposed method entails two major improvements toward reduction of search space:

- Frequent 2-itemsets and pre-frequent 2-itemsets from original database (*DB*) are carried over to the next partition (*db*) as candidate set for incremental mining. As a result, there is the least chance of new frequent 2-itemset in *db* that is not itemized in candidate set obtained from previous mining.

TABLE I

POSSIBLE STATES AND THEIR CONSEQUENCES

	Case: Original- New	**Results in the updated database**
1	Original=Frequent, New= Frequent	Updated=Frequent
2	Original=Frequent New=Pre-frequent	Updated ≥ Pre-frequent
3	Original=Frequent New= Infrequent	Cannot be predicted
4	Original=Pre-frequent New=Frequent	Updated ≥ Pre-frequent
5	Original=Pre-frequent New=Pre-frequent	Updated=Pre-frequent
6	Original=Pre-frequent New=Infrequent	Updated=Pre-frequent or Infrequent
7	Original=Infrequent New=Frequent	Updated ≥ Pre-frequent
8	Original=Infrequent New=Pre-frequent	Updated=Pre-frequent or Infrequent
9	Original=Infrequent New=Infrequent	Updated=Infrequent

- It adopts the property 1, 2 and 3, i.e., for an itemset $X \subseteq I$ to be frequent in time-interval $[t_s, t_e]$, all $Y \subseteq X$ and $Y \subseteq I$ is frequent on the same time-interval $[t_s, t_e]$. The algorithm filters outs infrequent itemsets in updated database at the very beginning.

Therefore, during the candidate generation process db, any itemset containing infrequent subset in updated database (UB) is immediately pruned (Property 2). The state of itemsets may change in the UB. Table I shows all possible outcomes after updating the database. The outcome of cases 1,2,4 and 5 is known since the supports of all such itemsets are already available from previous mining Case 3 shows frequent itemset in original database.

TABLE III

SYMBOLS USED IN THE ALGORITHM AND THEIR MEANING

Symbol	**Meaning**
N	Current partition number
DB , db, UB	Original, Incremental, Updated database
X,Y	Items
$X^{MCP(X)}$	Itemset with MCP
σ	Minimum support
δ	Probable support
C_2^{DB}, C_2^{db}, C_k^{UB}	Candidate 2-itemsets in DB, db and UB
F_2^{UB}	Frequent 2-itemsets in UB
LF_2^{UB}	Support of X in db
C^{PF}	Pre-frequent 2-itemsets in UB
$\|P_p\|$	No. of transactions in P_p
TI	Candidate temporal k-itemsets
SI	Candidate subsets of TI
$X.sup_{DB}$, $X.sup_{db}$	Support of X in DB and db
$X.sup_{UB}$	Support of X in UB
$X.sup_p$	Support of X in partition p
TFI	Temporal frequent itemset

This scenario represents the case of seasonal item when season comes to an end. Case 6, 8 and 9 is not interesting as itemsets cannot be frequent. Only in case 7, there is a little probability of an infrequent itemset to become frequent after updating. With every update the chances of support value to become frequent increases linearly (from Eq. (5) and Eq. (6)). In case, the transaction behavior in changes abruptly, the algorithm handles it and distinguishes itemsets that are expected to be frequent. Table II depicts symbols and its description used in the proposed algorithm.

Amongst all such itemsets that are infrequent in *DB* but frequent in incremental database (*db)*, the algorithm selects only frequent 2-itemsets X for candidate generation that satisfies following equation:

$$X.sup_{db} + \left(\delta \times \cup_{p=1}^{n-1} |P_p| - 1\right) \geq \sigma \times |UB| \qquad (7)$$

The present approach causes a reduction in frequent 2-itemsets. In addition, it reduces the candidate k-itemsets to be searched in UB.

Nevertheless, itemsets that are frequent in db and do not satisfy Eq. (7) are seasonal items or publication items in the itemset. In incremental mining, three possible categories (α, β and γ) of frequent itemsets appear.

α - In this category, itemsets are frequent in the UB. These are global itemsets.

β - In this category, itemsets frequent in the db, but not frequent in the DB. These itemsets contain seasonal or publication items.

γ - In this category, itemsets are frequent in the DB, and not frequent in db. These itemsets are seasonal, but no longer frequent.

The ITFPM explores frequent itemsets of categories: α and β.

B. The Algorithm: ITFPM

The algorithm ITFPM shown in Fig.1 and Fig.2 discovers TFI from the UB after db is appended in DB. All the 2-itemsets present in db is taken in the mining process. The reason is, in an updating database, there are few itemsets that are frequent in DB but not frequent in db. However, they contribute to increasing the support count in the entire database. The algorithm works on three primary tasks.

Initially, it discovers candidate 2-itemsets in db to merges with candidate 2-itemsets achieved in previous mining. The entire candidate 2-itemsets present, either in C_2^{db} or in both, are investigated by implementing Eq. (5) to confirm whether itemsets qualify frequent itemset criteria in UB. Here, itemsets that have no existence in db are removed even if they are frequent in DB. Such itemsets are type γ itemsets. The candidates that are frequent in UB are global itemsets.

Secondly, if itemsets fail to be frequent in UB though frequent in db are considered as β category itemsets, i.e. local frequent itemset. For local itemsets start time and end time are reset by the partition number in which they belong, i.e. it set to n. All frequent itemsets of category α and β are considered for candidate k-itemset generation. The candidate generation process utilizes pruning technique based on properties 1-3 and generates only a few candidate k-itemsets satisfying the above properties. Third, after the generation of pruned candidate k-itemsets, it counts the support of candidates in the updated database. The UB is scanned once in the incremental mining process for counting the support of generated candidate k-itemsets. Itemsets with support more than or equal to relative minimum support threshold along with their time interval are TFI in the updated database. Finally, the algorithm discovers all α and β (global and local frequent) itemsets.

Algorithm 1

Input: DB, db, C_2^{DB}, σ, δ, n
Output: TFI, C_2^{UB}

/* Find support of candidate 2-itemsets in a single pass using hash table*/
1. C_2^{db} = Count_support(db)
2. /* Merge candidate itemsets of DB and db */
3. For each itemset $X \in C_2^{db}$
4. If itemset $X \in C_2^{DB}$ then
5. $X.sup = X.sup_{DB} + X.sup_{db}$
6. $C_2 = C_2 \cup \{X^{MCP(X)}\}$
7. Remove itemset X from C_2^{db}
8. End for
/* Find probable frequent and frequent 2-itemsets in UB */
9. Set $C^{PF} = \emptyset$, $F_2^{UB} = \emptyset$ /* initialize pre-frequent set and frequent set*/
10. For each itemset $X \in C_2$
11. If $X.sup \geq \delta \times \cup_{p=s}^{n} |P_p|$then
12. $C^{PF} = C^{PF} \cup \{X^{MCP(X)}\}$
13. If $X.sup \geq \sigma \times \cup_{p=s}^{n} |P_p|$ then
14. $F_2^{UB} = F_2^{UB} \cup \{X^{MCP(X)}\}$
15. End for
/* Filter frequent 2-itemsets in n^{th} segment */
16. Set $LF_2^{UB} = \emptyset$ /* initialize local frequent set
17. For each itemset $X \in C_2^{db}$
18. If $X.sup_{db} \geq \sigma \times |P_n|$
19. If $X.sup_{db} + (\delta \times \cup_{p=s}^{n-1} |P_p| - 1) \geq \sigma \times \cup_{p=s}^{n} |P_p|$ then
20. $C_2^{UB} = C_2^{UB} \cup \{X^{MCP(X)}\}$
21. Else
22. Set $X^{MCP(X)} = X^{n,n}$
23. $LF_2^{UB} = LF_2^{UB} \cup \{X^{MCP(X)}\}$
24. $C^{PF} = C^{PF} \cup \{X^{MCP(X)}\}$
25. End for
/*Generate candidate temporal itemsets TIs and SIs*/
26. TI= Cand_gen($C_2^{UB} \cup F_2^{UB} \cup LF_2^{UB}$)
27. /* generate candidate subsets SIs of TIs and F_2^{UB} *and* SF_2^{UB}*/
28. For each itemset $X^{MCP(X)} \in TI \cup F_2^{UB} \cup SF_2^{UB}$
29. $ST = \{S^{MCP(X)} \mid S \subseteq TI \cup F_2^{UB} \cup SF_2^{UB}\}$
30. $SI = SI \cup ST$
31. End for
32. * Count support in each partition of updated database */
33. For p=n to 1
34. For each $X \in (TI \cup SI)$
35. If p in MCP(X)
36. If $X.sup_p ! = 0$
37. $X.sup_{UB} = X.sup_{UB} + X.sup_p$
38. Else
39. $X^{MCP(X)} = X^{p+1,n}$
40. End for
41. End for
42. /*Find frequent itemset */
43. For each itemset X∈ $(TI \cup SI)$
44. If $X.sup_{UB} \geq \sigma \times \cup_{p=s}^{n} |P_p|$ then
45. $TFI = TFI \cup \{X^{MCP(X)}\}$
46. End for
47. /* Set candidate 2-itemset for next segment*/
48. $C_2^{UB} = C^{PF}$

Fig. 1 Steps of proposed incremental algorithm

Algorithm 2: Cand_gen()

Input: C_2^{UB} , F_2^{UB},LF_2^{UB}
Output: TI

1. Set $C_2^{UB} = C_2^{UB} \cup F_2^{UB} \cup LF_2^{UB}$, k =3, $TI = C_2^{UB}$, $SI = \emptyset$
2. While ($C_{k-1}^{UB} \neq \emptyset$) do
3. For each pair X and Y with common $k-2$ itemset $\in C_{k-1}^{UB}$
4. If $X[k-1] \times Y[k-1]) \in F_2^{UB} \cup SF_2^{UB}$ then
5. $C_k^{UB} = \{X \times Y\}^{MCP(X \cup Y)}$
6. End for
7. $TI = TI \cup C_k^{UB}$
8. $k = k+1$
9. End do

Fig. 2 Steps of candidate generation and pruning

C. Illustrative Example

To visualize operation of the algorithm, a transaction database, shown in Table III, is considered. It models a publication database where items have different exhibition times. This section aims to show what itemsets arrive at the next partition after mining; and how it influences the performance of successive mining. The frequent itemsets obtained after processing of each incremental database are either global itemsets or local. Let us consider σ 40% and δ at 20%. The step-by-step change in the state of itemsets after incremental mining of partition is described here:

TABLE IIIII
A TRANSACTIONAL DATABASE

Partition	Month	TID	Itemset
P1	Oct-17	00101	A B
		00102	A B D
		00103	B C D
		00104	B C
		00105	B D
P2	Nov-17	00106	B C E
		00107	A B D E
		00108	A D E
		00109	B C D
		00110	A C E
P3	Dec-17	00111	A D
		00112	A B C
		00113	A B D E
		00114	D E
		00115	A B F

Mining of P1: Initially, transactions in partition *P1* have items {A, B, C, D}. After scanning *P1* first time, {AB, BC, BD} are found frequent 2-itemsets and {AD, CD} are found pre-frequent 2-itemsets for given threshold values. These itemsets {AB, BC, BD, AD, CD} are carried forwarded for mining of next partition *P2*. Fig.3 shows frequent, and pre-frequent 2-itemsets discovered in *P1*. Pre-frequent 2-itemsets are depicted by shaded area throughout in rest of the paper. Frequent 2-itemsets are used to generate candidate k-itemsets. Here, {BCD} is the only candidate k-itemset generated by {AB, BC, BD}. But, since {CD}$\subseteq${BCD} is not in frequent itemset list, hence, owing the property 1, {BCD} cannot be frequent thus it is pruned.

Partition	Itemset	Starting	Support
P1	AB	1	2
	AD	1	1
	BC	1	2
	BD	1	3
	CD	1	1

Fig. 3 Candidate 2-itemsets generated in P1

Since the support of frequent 2-itemsets is already counted, therefore, candidate k-itemsets in the database *P1* are only supersets of frequent 2-itemsets.

***Mining of P2*:** Table III shows a new item E in the transactions of *P2*. When E is combined with other items that appeared in P1, their common exhibition period is {2, 2}, the minimum support value for these itemsets to become frequent depends on the size of *P2*, i.e. $\sigma \times |P2|$. For the rest of the itemsets, σ' and δ' are calculated by Eq. (3) and Eq. (4).Candidate 2-itemsets obtained from previous mining {AB, BC, BD, AD, CD} when merged with candidate itemsets of *P2*, {AE, BC, BD, BE, CE, DE} are obtained as frequent and {AB, AD, CD} as pre-frequent itemsets. The frequent and pre-frequent itemsets are further carry forwarded for mining of partition *P3* (Fig. 4(a)). Here, {BC, BD} are global frequent itemsets while {AE, BE, CE, DE} are local frequent itemsets. Further, AD is frequent in *P2*. However, it is pre-frequent itemset in *P1*, and its cumulative support is 3. It is again a pre-frequent itemset since $\delta' \leq 3 \leq .\sigma'$.

Partition	Itemset	Starting	σ'	δ'	Support
P1+P2+P3	AB	1	6	3	3+3
	AD	1	6	3	3+2
	AE	2	4	2	3+1
	AF	3	2	1	1
	BC	1	6	3	4+1
	BD	1	6	3	5+1
	BE	2	4	2	2+1
	BF	3	2	1	1
	DE	2	4	2	2+2

(a) Search itemsets -$BC^{2,2}$, $BD^{2,2}$, $BCE^{2,2}$, $BDE^{2,2}$
Itemset pruned –BCD

Partition	Itemset	Starting	σ'	δ'	Support
P1+P2	AB	1	4	2	2+1
	AD	1	4	2	1+2
	AE	2	2	1	3
	BC	1	4	2	2+2
	BD	1	4	2	3+2
	BE	2	2	1	2
	CD	1	4	2	1+1
	CE	2	2	1	2
	DE	2	2	1	2

(b) Search itemsets-Empty search space
Itemset pruned –ABE
Fig. 4 Candidates generated in (a) *P2*and (b) *P3*

It also lists itemsets that are to be searched in the entire updated database, and are pruned at some point in the k-itemset generation process.

Mining of P3: Here, item F appears the first time in P3; therefore MCP of itemsets containing F is {3, 3}.Similar to mining of P2, candidate set obtained from P2 {AE, BC, BD, BE, CE, DE, AB, AD, CD} when merged with candidate set of P3, {AB, AE, BD, DE} become frequent itemsets and {AD, AF, BF, BC, BE} as pre-frequent itemsets. Here also, AB and AD frequent in db, are not required to be searched in entire database P1+P2+P3 for counting their support, as it is forwarded from previous mining, hence its support is known in advance. Itemset CE was frequent in*P1+P2*, but neither frequent nor pre-frequent in *P1+P2+P3*. It falls in category γ. Supersets of itemsets in γ are not frequent in the updated database. All frequent itemsets in updated database *P1+P2+P3* fall in category α.

IV. EXPERIMENTAL RESULTS AND DISCUSSION

To judge the performance of the proposed algorithm and to compare it with state-of-the-art algorithms, experiments have been performed on different datasets. Those experiments have been performed on Intel(R) Core(TM) i3-3217U machine in Windows 8.1 platform by simulating in Java. The parameters that have been considered for comparison are the number of candidate set generated and memory utilization, execution time and scalability.

A. Description of Dataset

Experiments have been performed on datasets from available data repository [35] shown in Table IV.

TABLE IVV
DESCRIPTION OF DATASETS

Dataset	DB	db	Items	Average Transaction
Retail	85,162	3,000	16,470	13
Mushroom	6,124	2,000	119	23
T10D100	90,000	10,000	1,000	10
T40D100	90,000	10,000	1,000	10

These databases exhibit varying sizes and also differ in the average number of items present in transactions; as a result, they are characterized as sparse and dense. Here, Retail and T10D100 are sparse datasets and Mushroom and T40D100 are dense datasets. For incremental mining, each database is partitioned into two sub-databases as original, and other as the incremental. The algorithms ITFPM, ITARM and IndxTAR have been applied on the same datasets to assess their comparative performance.

Experiments have also been performed to set value for δ for differentσ. It is observed that the median of σ is most suitable for δ and shows consistent performance, hence, taken throughout the experiment.

B. Candidates Generation and Memory Utilization

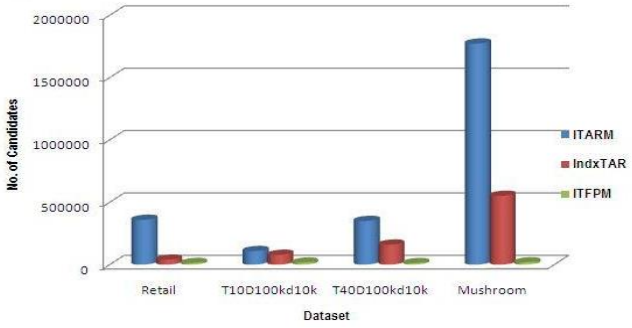
The performance of an algorithm depends on the number of candidates generated by the algorithm. Therefore, a number of candidates generated by each algorithm in different datasets is recorded. It is observed from the experiments that the

TABLE V
NO. OF CANDIDATES GENERATED

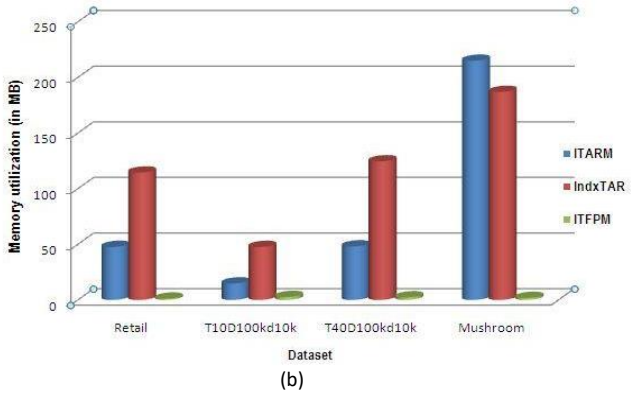
	No. of candidates generated		
Data	IT	Ind	I
Retai	3	39	1
T10D	1	78	1
T40D	3	16	1
Mus	1	55	1

TABLE VI
MEMORY UTILIZED

	Memory utilized (in MB)		
Datas	I	In	I
Retail	4	11	1
T10D	1	48	2
T40D	4	12	2
Mush	2	18	1



(a)



(b)

Fig. 5 (a) Candidates generated (b) memory utilized in Retail, T10D10k, T40D10k, and Mushroom dataset.

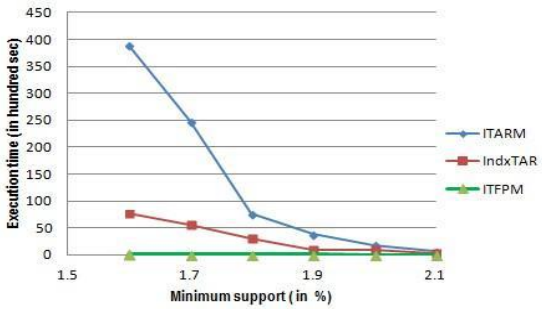
number of candidates generated in dense datasets is very large compared to sparse datasets. Table V, Table VI and Fig.5 depict the disparity in the size of candidates set and memory utilized.

In both dense and sparse datasets, algorithm ITFPM has generated lesser candidates than ITARM and IndxTAR. ITARM considers all frequent itemsets of updated as well as incremental database.

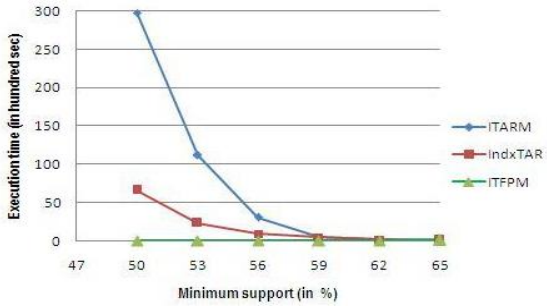
Because of the level-wise approach, it generates a lesser number of itemsets than ITARM. Nevertheless, ITFPM generates a number of candidates that is close to the count of frequent itemsets. The process of candidate set generation takes a number of cycles in IndxTAR, while ITARM and ITFPM generate the candidate set twice only. Fig.5 (a) shows the maximum number of candidates generated by different algorithms. Since IndxTAR keeps individual database separately under the root node, its memory Utilization to store transactions is very high as compared to ITARM, but memory utilization of ITARM is more than ITFPM. A comparative graph is shown in Fig. 5(b) depicts the memory utilization.

C. Execution Time Performance

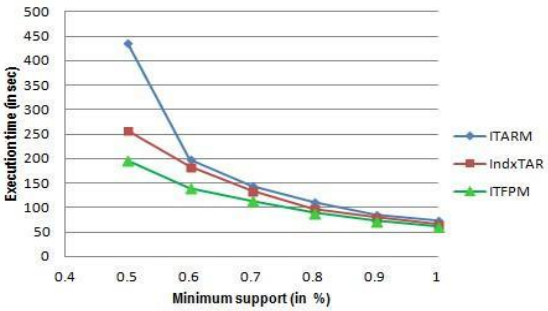
Execution times of algorithms under various minimum support thresholds have been recorded. Based on the characteristics of the datasets, varying range of minimum support is usual for each datasets. The four panels of Fig.6 shows two major behavior in the results of the algorithms: Initially, the execution time of ITARM demonstrate a sharp increase with decreasing minimum support. However, IndxTAR shows a gradual increase while the execution time of ITFPM remains relatively constant across all minimum support values. Secondly, ITARM takes a longer time to mine dense datasets compared to sparse datasets, as it needs to search through a large number of candidates. Similarly, IndxTAR is also time-consuming relative to ITFPM algorithm because it appends tree with incremental data, maintains TIndex data structure and conducts Apriori like operation. But, ITFPM performs well in sparse and dense datasets both. The performance difference of the algorithms is always better throughout, and the runtime of ITFPM is lesser than ITARM



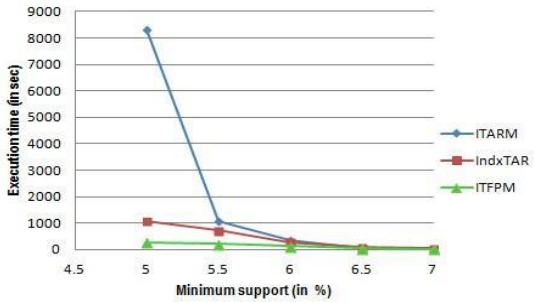
(a)



(b)



(c)



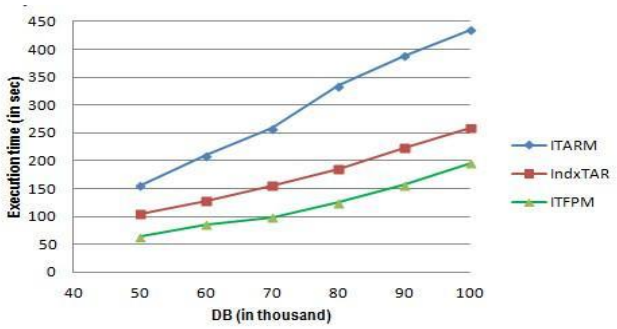
(d)

Fig. 6 Execution time in dataset (a) Retail, (b) Mushroom, (c) T10D100k and (d) T40D100k

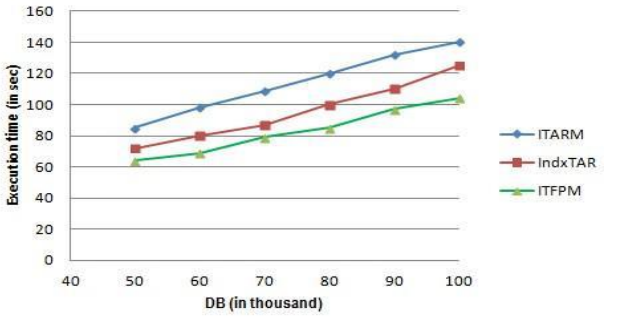
and IndxTAR. Hence ITFPM outperformed ITARM and IndxTAR.

D. Scalability

Experiments with different minimum support for different DB sizes have been performed on T10D100k dataset to analyze the scalability of the algorithm. Fig.7 shows the results obtained on conducting experiments on progressive DB for minimum support of 0.5% and 0.7% respectively. At 0.5%support threshold shown in Fig.7 (a), ITARM shows a fast increase in execution time compared to IndxTAR and proposed algorithm. At 0.7% support threshold shown in Fig.7 (b), there is a continuous increase in execution time. However, it is observed that the proposed algorithm is more stable in comparison to other algorithms.



(a)



(b)

Fig. **7** Scalability of the algorithm for minimum support of 0.5% and 0.7% respectively

V. CONCLUSION

The algorithm introduced in this paper explores and maintains global temporal frequent patterns and local temporal frequent patterns in the latest time window that can further be employed to derive temporal association rules. The present research work has a major focus on reducing the risk of combinatorial explosion, which is a major issue in the candidate generation technique when implemented in incremental mining. The threshold values implemented in the algorithm ITFPM are used to optimize search space. Moreover, the incorporation of pruning during candidate generation is an innovative idea in this pursuit which further reduces the candidate itemsets in turn and makes the proposed approach more versatile and accurate. These actions primarily affect the execution time in discovering temporal frequent itemsets to a great extent. The algorithm is capable of handling publication and seasonal database and discovers global as well as local frequent-patterns.

The algorithm can be implemented in future for discovering global and temporal patterns in different domains like Retail, Financial Analysis, etc.

REFERENCES

[1] A. Segura-Delgado, M. J. Gacto, R. Alcalá and J. Alcalá-Fdez, "Temporal association rule mining: An overview considering the time variable as an integral or implied component," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 10(4), pp. e1367,2020.

[2] M.H. Dunham, *Data Mining Introductory and advanced Topics*, Pearson, 6th ed., India, 2006.

[3] J.M. Ale and G.H. Rossi, An approach to discovering temporal association rules, *ACM Symposium on Applied computing*, 2000.

[4] J. F. Roddick and M. Spiliopoulou Survey of temporal knowledge discovery paradigms and methods. *Knowledge and Data Engineering*, 1, pp. :750-767, 2002.

[5] K. Verma and O.P. Vyas, Efficient calendar based temporal association rule. ACM SIGMOD Record, 2005

[6] Piscitelli MS, Mazzarelli DM, Capozzoli A. Enhancing operational performance of AHUs through an advanced fault detection and diagnosis process based on temporal association and decision rules. Energy and Buildings, 2020; 226:110369.

[7] Böhmer K, Rinderle-Ma S. Mining association rules for anomaly detection in dynamic process runtime behavior and explaining the root cause to users. *Information Systems*, 2020;90:101438.

[8] Radhakrishna V, Kumar PV, Janaki V. A Survey on Temporal Databases and Data mining. Eng. MIS 2015.

[9] Ng V, Chan S, Lau D, Ying C. M. Incremental mining for temporal association rules for crime pattern discoveries. Australasian Database, 2007.

[10] Vidhu A, Shibili T. Network Intrusion Detection Using Temporal Association Rules. International Journal of Science Engineering and Research, 2014; 4:378.

[11] Guidotti R, Gabrielli L, Monreale A, Pedreschi D, Giannotti F. Discovering temporal regularities in retail customers' shopping behavior. EPJ Data Science. 2018; 7: 6.

[12] Cheung DW, Lee SD, Kao B. A General Incremental Technique for Maintaining Discovered Association Rules. Database Syst. Adv. Appl. 1997;

[13] Hong TP, Wang CY, Tao YH. A new incremental data mining algorithm using pre-large itemsets. Inteligent Data Analysis. 2001; 111–129.
[14] Lee WJ, Lee SJ. An Efficient Mining Method for Incremental Updation in Large Databases. Intelligent Data Eng. Auto. Learning Liu J., Cheung Y., Yin H. (eds). Springer, Heidelberg, 2003; 630–637.
[15] Ezeife CI, Su Y. Mining Incremental Association Rules with Generalized FP-Tree. Adv. Artificial Intelligence. Cohen R., Spencer B. (eds), Springer, Heidelberg. 2002; 147-160.
[16] Li X, Deng ZH, Tang S. A Fast Algorithm for Maintenance of Association Rules in Incremental Databases. Adv. Data Mining Appl., Li X. Zaïane OR. Li Z. (eds), Springer. Heidelberg. 2006; 56-63.
[17] Lin CW, Hong TP, Lu WH, Chien BC. Incremental Mining with Prelarge Trees. New Front. Appl. Artif. Intell., Nguyen NT, Borzemski L, Grzech A., Ali M. (eds). Springer. Heidelberg. 2008; 169-178.
[18] Cheung DW, Han J, Ng VT, Wong CY. Maintenance of discovered association rules in large databases: An incremental updating approach. Data Engineering. 1996.
[19] Gharib TF, Hamed N, Taha M, Abraham A. An efficient algorithm for incremental mining of temporal association rules. Data Knowledge Engineering. 2010; 69: 800-815.
[20] Fouad MM, Mostafa MGM. IndxTAR: An Efficient Algorithm for Indexed Mining of Incremental Temporal Association Rules. Internation Journal of Computer Information Syst. Ind. Manage. Appl, 2017; 9: 103-113.
[21] Agrawal R, Srikant R. Fast Algorithms for Mining Association Rules. International conference on Very Large Data Bases. 1994; 487-499.
[22] Lee CH, Lin CR, Chen MS. Sliding-Window Filtering: An Efficient Algorithm for Incremental Mining. Information and Knowledge Management. 2001.
[23] Chang C, Li YC, Lee JS. An Efficient Algorithm for Incremental Mining of Association Rules. Res. Issues Data Eng. , 2005.
[24] Huang JW, Dai BR, Chen MS. Twain: Two-end association miner with precise frequent exhibition periods. ACM Transaction: Knowledge Discovery Data. 2007; 1: 8.
[25] Li Y, Zhang ZH, Chen WB, Min F. TDUP: an approach to incremental mining of frequent itemsets with three-way-decision pattern updating. Int. J. Mach. Learn. Cybern. 2015; 8:441-453.
[26] Han J, Pei J, Yin Y. Mining frequent patterns without candidate generation. Manage. Data, 2000.
[27] Lin CM, Hsieh YL, Yin KC, Hung MC, Yang DL. ADMiner: An Incremental Data Mining Approach Using a Compressed FP-tree. Journal of Software. 2013; 8: 2095-2193.
[28] Dafa-Alla AFA, Shon HS, Saeed KEK, Piao M, et.al.. IMTAR: Incremental Mining of General Temporal Association Rules. *Journal of Information Processessing System*.2010; 6: 163-176.
[29] Wang L, Meng J, Xu P, Peng K. Mining temporal association rules with frequent itemsets tree. Application of Soft Computing. 2018;62: 817–82.
[30] Conen F, Leng P, Ahmed S. Data Structure for Association Rules Mining: T-tree and P-tree. Knowledge Data Engineering. 2004; 16: 774-778.
[31] Shalini S, Lal K. Mining Changes in Temporal Patterns in Latest Time Window for Knowledge Discovery. Journal of Information & Knowledge Management. 2019;18(03): 1950028.
[32] Toivonen H. Sampling Large Databases for Association Rules. Very Large Data Bases, 1996.
[33] Lin CW, Hong TP, Lu WH. The Pre-FUFP algorithm for incremental mining. Expert System Application. 2009;36:9498-9505.
[34] C. C. Aggarwal and J. Han, Frequent Pattern Mining, Springer International, Switzerland, 2014.
[35] Frequent Itemset Mining Dataset Repository: Available online at: http://fimi.ua.ac.be/data/ (Accessed on 6th April 2023)