Negative Itemset Tree for Discovering Rare Patterns with Periodicity from Static Databases

Jyothi Upadhya K, Thanishka Bolisetty, B Dinesh Rao and Geetha M*

Abstract*—Periodic patterns assist in the discovery of crucial information in the fields, including fraud detection, telecommunications, retail marketing, research, and medicine. Numerous methods have been developed for the extraction of periodic frequent patterns. The capacity to find uncommon or unexpected combinations that are missed by periodic frequent pattern mining techniques is the main motivation for Periodic Rare Pattern Mining. If the rare patterns are distributed across the transaction dataset, they are periodic and important. A novel *PRPNegTidTreeMiner* algorithm is proposed to discover rare patterns with periodicity along with support threshold. An efficient tree structure, *NegTidTree,* is constructed to capture the complete negative representation of the given static database. *NegTidTree,* serves the dual purpose of finding support count as well as periodicity information. This tree accelerates the extraction of periodic rare patterns and prevents repeated database scanning. *PRPNegTidTreeMiner* uses two mining techniques to generate periodic rare patterns. The first mining method, *NegTidTreeMiner,* employs a top-down approach to find rare patterns with periodicity thresholds which avoids traversal of frequent patterns. In the second technique, *NegTidTreeMinerFLP*, the mining efficiency is improvised by avoiding the traversal of frequent as well as non-existing patterns. Experiments are carried out by varying support and periodicity components for a variety of datasets. The results show that *NegTidTreeMiner* performs better than *NegTidTreeMinerFLP* when the dataset is small and generates a huge number of periodic rare patterns. When the size of the database grows, *NegTidTreeMinerFLP* continuously outperforms *NegTidTreeMiner.

***Index Terms*—Periodic Rare Pattern Mining, Rare Pattern Mining, NegTidTree, Rare Patterns, Periodic Patterns, Tree data structure**

I. Introduction

Extracting and further associating the extracted information to discover the significant information from an enormous bundle of data generated by the various fields are challenging tasks. In this direction, there exist several algorithms to discover significant information in the area of Association Rule Mining(ARM). Pattern Mining is an important phase of ARM, as it deals with the explosive growth of the search space. Itemsets or patterns with support values greater than or equal to the user-defined support threshold value are called frequent patterns, whereas itemsets having support lesser than the threshold are termed rare patterns. Rare Pattern Mining(RPM) has been emerging as a new promising area to discover hidden unexpected or unusual activities. The RPM algorithms focus on discovering low-support significant association rules. The sample database shown in Table I consists of 7 transactions and 7 unique items. Let the maximum support called here as $maxSup$ be equal to 4. The itemsets having a support count lesser than the $maxSup$ are treated as rare patterns. In the sample database shown in Table I, {ABE}, {AD} are considered as some rare items according to the specified threshold.

TABLE I: Pos-Rep Transaction - TD

TID	Items
T1	A, B, C, F
T2	A, B, C, D, F
T3	C, E, F
T4	A, B, E
T5	A, B, F
T6	D, E, F, G
T7	E, F

TABLE II: Neg-Rep of TD - $\overline{TD}$ and Sorted items of $\overline{TD}$ after removal of {G}

TID	items of $\overline{TD}$	sorted items of $\overline{TD}$
T1	$\neg D$, $\neg E$, $\neg G$	$\neg D$, $\neg E$
T2	$\neg E$, $\neg G$	$\neg E$
T3	$\neg A$, $\neg B$,$\neg D$, $\neg G$	$\neg D$,$\neg A$, $\neg B$
T4	$\neg C$, $\neg D$, $\neg F$, $\neg G$	$\neg D$, $\neg C$, $\neg F$
T5	$\neg C$, $\neg D$, $\neg E$, $\neg G$	$\neg D$, $\neg C$, $\neg E$
T6	$\neg A$, $\neg B$, $\neg C$	$\neg C$, $\neg A$, $\neg B$
T7	$\neg A$, $\neg B$, $\neg C$, $\neg D$, $\neg G$	$\neg D$, $\neg C$, $\neg A$, $\neg B$

The algorithms designed in this area face two challenges. As rare itemsets are discarded in FPM, the frequent itemsets can not be discarded in RPM because there is a possibility that their supersets may become rare. This increases the volume of the search space requiring finding strategies to reduce the search space. As rare patterns are low support patterns, a huge set of spurious patterns is generated along with the significant information. Filtering unwanted patterns is another challenging task.

Rare pattern mining algorithms can be classified based upon the traversal techniques that are used for discovering rare itemsets: top-down or bottom-up fashion. Szathmary et al.[1] used bottom-up traversal technique in the *Apriori-Rare* and *MRG-Exp* to find the minimal rare itemsets (MRIs) which lie in the negative border of the frequent itemset zone. Further, the *ARIMA* algorithm[2] finds all rare itemsets from already discovered MRIs by removing zero generators. To compute the support at each level the database is scanned which degrades the performance. In order to handle the spurious rare itemsets discovered during mining of low support threshold value, Bhasker et al.[3], [4] used the *bond* measure with

Manuscript received September 30, 2022; revised April 05, 2023.

Jyothi Upadhya K is an Assistant Professor in the Department of Computer Science and Engineering, Manipal Institute of Technology, Manipal Academy of Higher Education, Manipal, Karnataka, India, 576104, email-id:jyothi.k@manipal.edu

Thanishka Bolisetty is an undergraduate student in the Department of Computer Science and Engineering, Manipal Institute of Technology, Manipal Academy of Higher Education, Manipal, Karnataka, India, 576104, email-id:thanishka.bolisetty99@gmail.com

B Dinesh Rao is a Professor at Manipal School of Information Sciences, Manipal Academy of Higher Education, Manipal, Karnataka, India, 576104, email-id:dinesh.rao@manipal.edu

Geetha M is a Professor in the Department of Computer Science and Engineering, Manipal Institute of Technology, Manipal Academy of Higher Education, Manipal, Karnataka, India, 576104, Correspondence mail-id: geetha.maiya@manipal.edu.

support threshold. *CORI* algorithm first transforms the input database into its vertical bitwise representation which aids in finding the disjunctive as well as conjunctive support of the itemsets. Using this metric the entire rare correlated patterns are extracted in a bottom-up fashion. Anindita et al.[5] developed *SSP*(Single Scan Pattern Tree) to mine both frequent and rare patterns from incremental databases in a bottom-up fashion. A compact tree is built by sorting the transactions in frequency descending order and then inserting them into the tree. During the tree construction, if any path of the tree deviates from the frequency descending order then the path is re-arranged. The binary count tree (BIN-Tree) proposed by Shwetha Rai et al.[6] is a compact tree structure proposed to store the static data in the main memory. An efficient mining technique is proposed to discover both rare and frequent itemsets.

In order to reduce the search space, most of the existing RPM algorithms that follow the bottom-up approach have found the solution by mining subsets of rare patterns. These algorithms concentrated on finding only rare 1-itemsets and their supersets by considering only those transactions containing at least one rare item. Tsang et al.[7] developed a compact tree structure called Rare Pattern Tree (RP-Tree) which is similar to the FP-Tree structure [8]. The model displays only those rare itemsets that fall in the range of *minFrepSup* and *minRareSup* threshold values. Anindita et al.[9] proposed Hyper-Linked Rare Pattern Mining, a memory-based queue data structure with the hyper-linked pattern that can mine subsets of rare patterns. Algorithms proposed in [10], [11], discovered minimal rare itemsets using the bottom-up approach which traverses through several frequent itemsets to reach the minimal rare itemsets in the lattice.

The RPM algorithms that follow the Apriori technique and top-down approach were able to generate all rare itemsets. *ARANIM,* the Apriori Rare and Non-Present Itemset Mining is an algorithm designed by Adda et al.[12] which extracts rare itemsets and non-present itemsets in the given database. It follows a top-down policy by first generating a k-itemset which is a combination of all the single items in the database. Then repeatedly the next level of itemsets is obtained as a result of generating the subsets of the k-itemset. As it starts from k-itemset which may not be present in the database, it may generate many non-existent patterns that are discarded at every level. Troiano et al.[13], [14] designed the Rarity algorithm, where a level-wise top-down strategy is employed for extracting the rare itemsets. The framework first identifies the longest itemset in the database and then its level-wise subsets are found. Different list structures are used to store candidate, frequent and rare itemsets generated at each level. This method avoids scanning the database multiple times. The memory requirements are higher in order to store the different lists. *NII-Miner* is a recent contribution of Liu et al. [15] which considers the dual perspective of the original database by the representation of negative items. The missing items of the original database are represented in the form of a Neg-Rep database and a Negative Itemset Tree(NI-tree) is constructed similarly to FP-Tree using the negative items. This is the first tree-based method to discover all rare itemsets using a top-down depth-first strategy.

Periodic Frequent Pattern Mining (PFPM), a generalization of FPM, deals with regularity (shape of occurrence or occurrence characteristics or periodicity). The research started with the work of Tanbeer et al.[16] where the importance of considering regularity is proposed. Instead of support count, the transaction ids are stored in the leaf nodes of a Regular-Pattern tree that aid in the calculation of the regularity of itemsets. Further, the work is extended to find regular patterns from stream data[17] and body sensor networks[18]. Rashid et al.[19] considered variance measure along with support threshold for finding temporal regularity and discovered regularly frequent patterns from static data. This research is continued further in the field of wireless sensor networks to find regularly occurring frequent sensor patterns[20]. Uday Kiran et al.[21], [22], [23] addressed the "rare item problem" by setting multiple support and periodicity thresholds to enumerate frequent along with rare regular patterns from a static database. Viger et al.[24] proposed an algorithm to mine the stable periodic patterns from the database. To assess the stability of the periodic behaviour of a pattern the novel measure called lability is defined. The model is extended further to uncover top-k stable periodic patterns from a static database [25]. Amphawan et al.[26] designed a single-pass algorithm to discover top-k regular frequent itemsets in the static database without setting the support threshold. The work is improvised by modifying the algorithm based on partition and estimation techniques. *TFRIM-DS,* is a single-pass algorithm designed by Amphawan et al.[27] that makes use of the sliding window technique to find top-k regular itemsets with the highest support in the stream data. *MHUIRA,* an efficient single-pass algorithm is the contribution of Amphawan [28] that incorporates regularity in mining high-utility itemsets. Further, HUIIs-Miner[29] is designed to discover itemsets with high profits even though these items are infrequently purchased. *PHM* designed by Viger et al.[30] use minimum and average periodicity measures to enumerate periodic high-utility itemsets from the static database. Patterns in the non-uniform temporal database will show varied periods and minimal repetitions. To deal with partial periodic patterns, Kiran et al. developed models [31], [32], [33] that allows the number of periods connected to a pattern to exceed the *maxPer* threshold by a user-defined count value. The applications of PFPM is widely focused on gene and medical data analysis[34], [35], mobility intention analysis[36], website user behaviour analysis[37] so on.

Even though some itemsets in the database shown in Table I are rare patterns with respect to *maxSup,* the appearance of the patterns may be dominant only at a certain part of the database. For example, the patterns {AC},{BC},{AF},{BF},{EF} are some of the rare patterns with respect to the *maxSup* considered. The appearance of the itemsets {AF},{BF} and {EF} are throughout the database and called in this paper as periodic rare patterns. On the other hand, the existence of patterns {AC} and {BC} are only at the beginning of the database and the pattern {G} at the end of the database. The patterns {AC},{BC} and {G} are called here as non-periodic rare patterns. Although PFPM approaches can find regularity, the study is limited to frequent patterns. On the contrary, in most of the existing RPM approaches, there is no focus on the periodicity of rare itemsets. These approaches focus on the support threshold and do not give importance to the occurrence characteristics which may unhide significant information for useful decision-making.

Studying the periodic behaviour of rare patterns is also useful in many applications. For example, in a retail market, studying the regularity/irregularity of products purchased with the quantity helps to understand customer behaviour. For example, in a supermarket database groceries such as milk and bread are bought frequently in a month whereas rice and sugar are bought during the month's end. Even though Rice and sugar are rare items but it is periodically bought by customers. For a year-wise analysis, this rare and periodic item contributes to the profit of the supermarket. Alternately, items such as door mats and cushions that are bought yearly once are categorised as rare non-periodic items. This study, in turn, contributes to developing new market strategies, managing the inventory and layout of shelves etc. Similarly, studying the click sequences of rarely hit pages may help the web administrator to improvise the demand of the page. Studies involving the periodicity of rarely co-occurring genes in DNA sequence may produce unknown information to the scientists in gene data analysis. In the medical domain, unexpected responses to medications may be more crucial in some cases compared to frequent and known responses. Concentrating on periodic patterns of unexpected responses may further help the domain experts to study the occurrence details. Monitoring the occurrence characteristics of network packets/fraudulent transactions may help in controlling the network/fraudulent attacks. In traffic accidents, concentrating on the regularities of abnormal patterns helps to get the real cause of the accidents. The examples mentioned show the importance of considering the shape of the occurrence of rare patterns. Hence, periodicity plays a vital role in discovering significant rare patterns in a wide variety of applications. The focus of the study is to discover rare patterns along with their periodic behaviour. The only work done in this direction is the *MRCPPS* algorithm designed by Viger et al.[38] to mine rare correlated periodic patterns from multiple sequences. The standard deviation of periods is used as the periodicity measure. With the support threshold, the bond measure is used to filter the bundle of spurious patterns generated in the process of mining useful rare correlated periodic patterns.

The traditional PFPM algorithms focus on the discovery of periodic frequent patterns, hence following a bottom-up approach. This approach is not suitable for discovering the periodicity of rare patterns. *NIIMiner*[15] is a state-of-the-art algorithm that discovers rare patterns in a top-down manner. In this paper, *NIIMiner* is modified and a novel approach to the discovery of rare patterns with periodicity is proposed. The main contributions are:

- A new algorithm called *PRPNegTidTreeMiner* is proposed to discover the periodicity of rare patterns along with the support count from a static database. As for our knowledge, this is the first tree-based top-down approach that concentrates on the periodicity of rare patterns along with the support count.
- An efficient tree structure, *NegTidTree,* is constructed to capture the complete negative representation of the given static database. This tree helps in computing the support count with the periodicity measure. The root node of *NegTidTree* contains all periodic rare database items. *PRPNegTidTreeMiner,* traverses the items in the root node of the *NegTidTree* and extracts entire periodic rare itemsets from the given dataset.
- *PRPNegTidTreeMiner* uses *NegTidTreeMiner* to discover the periodic rare patterns of every item in the root node.
- Further the mining efficiency of *PRPNegTidTreeMiner* is improvised by designing a mining method called *NegTidTreeMinerFLP* which avoids traversal of non-existent itemsets.

The rest of the paper is organized in the following manner. Section II focuses on the basic definitions of the proposed algorithms. Various modules of *PRPNegTidTreeMiner* are presented in Section III. Experimental evaluation and result analysis are shown in section IV. The conclusion and future directions are highlighted in Section V.

II. Basic Terminologies

This section defines the basic terminologies and theorems that are used for discovering rare patterns and periodic patterns. Some of the notations are similar to the notations used in *NIIMiner*[15] and *MHUIRA*[28] algorithm. The terminologies are illustrated based on Example 1.

Let I = $\{i_1, i_2,....,i_m\}$ denote set of items. A transaction T=(tid, P) where *tid* represents unique transaction id and *P* is an n-itemset formed using *n* unique items of I. A transaction database TD consists of a set of tuples $\{T_1, T_2,....,T_z\}$ where each T_x is represented in the form of T.

Example 1. Consider a transaction database TD as given in Table I. Let the two thresholds *maxSup* and *maxPer* be set as 4. The further example refers to the transaction database TD shown in Table I.

A. Support of Patterns

Definition 1: For any item i∈I, where I=$\{i_1, i_2,....,i_m\}$, a positive pattern Ps=$\{ps_1, ps_2,..ps_i..,ps_q\}\subseteq$ I is called a Pos-Rep (Positive-Represented) pattern or simply pattern.

Definition 2: Negative item of i∈I is represented as ¬i. For the pattern Ps, its corresponding negative pattern can be represented as ¬Ps=$\{\neg ps_1, \neg ps_2,..\neg ps_i..,\neg ps_q\}$.

Definition 3: Given I=$\{i_1, i_2,...,i_m\}$ and Ps=$\{ps_1, ps_2,...,ps_q\} \subseteq$ I, the corresponding Neg-Rep (Negative-Represented) pattern is denoted by $\overline{Ps}$ = {¬i|{(i∈I) ∧ (i∉Ps)}} = I\Ps} which contains all the items of I that are missing in Ps.

Every transaction in TD is represented by its Neg-Rep to produce the database $\overline{TD}$. Neg-Rep transaction database of TD in Table I is shown in Table II.

Definition 4: The total count of transactions that contain all the items of Ps is denoted as the intersect support or absolute support of Ps in TD.

Ps.interSup = $|\{T\in TD \mid ps_1\in TD \wedge ps_2\in TD \wedge ...\wedge ps_q\in TD\}|$

In Example 1. Ps.intersup is equal to 3 for the pattern Ps={AF}.

Definition 5: A pattern Ps is a rare pattern if and only if (Ps.interSup <maxSup). In Example 1. The pattern Ps={AF} is a rare pattern since ({AF}).interSup <4).

Definition 6: In Neg-Rep database $\overline{TD}$, the total number of transactions in which at least one item of ¬Ps exists represents the joint support of ¬Ps. ¬Ps.jointSup = $|\{\overline{T}\in\overline{TD} \mid \neg ps_1\in\overline{T} \vee \neg ps_2\in\overline{T} \vee ...\vee \neg ps_l\in\overline{T}\}|$ [15]

Theorem 1: For a given transaction database *TD* and its Neg-Rep database $\overline{TD}$, the Ps.interSup = |TD| - $\neg Ps$.jointSup

Proof: If a pattern Ps is not present in a transaction T∈TD, then T $\not\supseteq$ Ps $\iff$ $\overline{T} \in \overline{TD} \cup \neg Ps \neq \emptyset$. By Definition 6 joint support of $\neg Ps$ can be obtained from $\overline{TD}$, $\neg Ps$.jointSup=$|\{\overline{T} \in \overline{TD} | \overline{T} \cup \neg Ps \neq \emptyset\}|$ = $|\{T \in TD | T \not\supseteq Ps\}|$. Ps.interSup = $|\{T \in TD \ | T \supseteq Ps\}|$.

Thus |TD| = Ps.interSup + $\neg Ps$.jointSup $\implies$ Ps.interSup = |TD| - $\neg Ps$.jointSup.

Considering pattern Ps={AF} in Example 1, it is found that $\neg Ps$.jointSup = 4 and therefore by Theorem 1 Ps.interSup = 7 - 4 = 3. Thus the support of pattern Ps in TD, which is a candidate itemset for the rare pattern can also be obtained using the joint support of the corresponding negative itemset present in $\overline{TD}$.

B. Periodicity of Patterns

Periodicity is the concept used to find the occurrence behaviour of patterns in the database. It is computed by observing the consecutive occurrences of patterns and measuring the gap between the consecutive occurrences.

Definition 7: If the pattern P occurs first in transaction T_f, this first period is called as $fPeriod$ and is calculated as fPeriod(P, T_f) = T_f - 0.

Definition 8: Any other intermediate period of pattern P is called $mPeriod$ and is measured by the gap between two consecutive transactions in which pattern P appears. If T_r and T_s are the two consecutive transactions in which itemset P appears, then mPeriod(P, T_r, T_s) = T_s - T_r.

Definition 9: The last period of pattern P is called as $lPeriod$ and is measured in terms of the gap between the last occurrence of pattern P and the last transaction in the database TD. If T_x is the last transaction in which pattern P occurs, then $lPeriod$ is calculated as lPeriod(P, T_x) = T_z - T_x.

Definition 10: The periods of pattern P is represented as P.Prd = {fPeriod(P, T_f), mPeriod(P, T_r, T_s), mPeriod(P, T_s, T_t),...., mPeriod(P, T_w, T_x), lPeriod(P, T_x)}. Then periodicity of itemset P denoted as P.maxPrd = max(fPeriod(P, T_f), mPeriod(P, T_r, T_s), mPeriod(P, T_s, T_t),...., mPeriod(P, T_w, T_x), lPeriod(P, T_x)).

For example the periods of pattern P={AF} in Example 1 is obtained as {AF}.Prd = {fPeriod({AF}, T_1), mPeriod({AF}, T_1, T_2), mPeriod({AF}, T_2, T_5), lPeriod({AF}, T_5)} = {1, 1, 3, 2}. Further {AF}.maxPrd is obtained as max(1, 1, 3, 2) = 3 accoding to Definition 10.

Problem Statement: Given the database TD, a support threshold *maxSup* and a periodicity threshold *maxPer,* the task of mining periodic rare pattern is to discover the entire set of patterns with (P.interSup <*maxSup*) $\wedge$ (P.maxPrd $\leq$ *maxPer*).

III. PRPNegTidTreeMiner(Periodic Rare Pattern Negative Transactionid Tree Miner): The proposed Algorithm

The proposed method *PRPNegTidTreeMiner* is able to discover an entire set of periodic rare patterns(PRPs) from the given static database. It discovers PRPs in two steps: (i) The given static database is converted into its Neg-Rep database and the corresponding *NegTidTree* is constructed (ii) The *NegTidTree* is mined in a top-down manner to find periodic rare patterns. Two user-defined threshold values *maxSup* and *maxPer* are accepted help to reduce the search space by discarding non-periodic patterns during the mining task.

A. Construction and Updation process of SPList

The support and periodicity information related to all 1-itemsets in the database are maintained in a list called *SPList. SPList* maintains three fields related to an item *i:* support count - *Sup(i),* The previous transaction id in which the item *i* present - *preTid(i)* and *maxPrd* of item *i* is - *maxPrd(i).* During the first scan of the database, the support count and the maximum periodicity values are updated in the *SPList* as shown in Algorithm 1. For the database shown in Table I, the *SPList* creation and updation is as shown in Figure 1. Once the entire database is scanned, those 1-itemsets which are not present in the last transaction are updated based on maximum periodicity. It is observed from Figure 1(d), *maxPrd* of item {C} is updated because the *maxPrd* obtained by considering the last transaction is greater than its previous *maxPrd* value. Further, the non-periodic 1-itemsets are discarded and the remaining 1-itemsets are sorted in descending order by their support value. Figure 1(e) shows the final *SPList* after discarding the non-periodic 1-itemset {G} and sorting the remaining 1-itemsets in descending order with respect to the support count. This updated *SPList* is used in the process of *NegTidTree* creation.

Algorithm 1 Construction of *SPList*

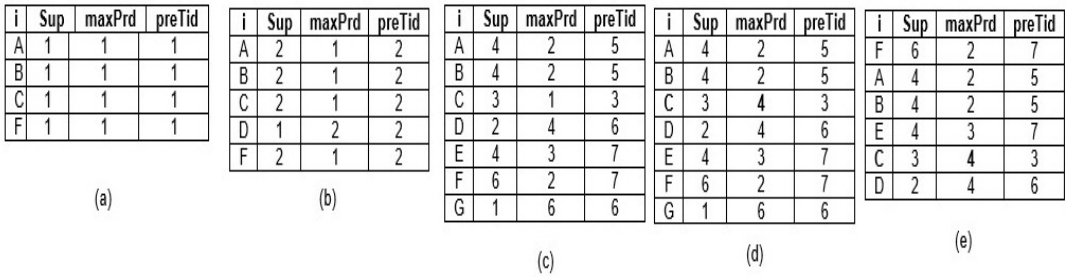
Input: *TD* - a transactional database, *maxPer* - maximum periodicity threshold

Output: *SPList* containing sorted list of items

1: **for each** transaction T ∈ TD with tid *tIDCur* **do**
2: **for each** item i ∈ T **do**
3: Increment Sup(i) by 1
4: Compute current period *curPrd* by subtracting the preTid(i) with *tIDCur.* If *curPrd* is larger than maxPrd(i) then replace maxPrd(i) by *curPrd*
5: preTid(i) ← *tIDCur*
6: **end for each**
7: **end for each**
8: Update Periodicity of all items in *SPList* with respect to |TD|
9: Remove each item i from *SPList* having maxPrd(i) >*maxPer* and Sort *SPList* in descending order of Support Count
10: **end procedure**

B. NegTidTree Structure

The *NegTidTree* is constructed to represent the transactions present in the Neg-Rep database $\overline{TD}$. The tree consists of a root node *NTRoot* and a set of item-prefix sub-trees represented as the children of *NTRoot.* Every path of the item-prefix sub-tree represents a unique transaction of $\overline{TD}$ where common paths are shared similar to NI-Tree[15]. The transactions in $\overline{TD}$ are arranged in ascending order with respect to the support count of items in TD. *ILabel* field of every child node *c* represents a negative item. As the root node *NTRoot* represents an itemset, it is differentiated from other nodes with the field *ISLabel.* Each node *c* keeps



(a)

i	Sup	maxPrd	preTid
A	1	1	1
B	1	1	1
C	1	1	1
F	1	1	1

(b)

i	Sup	maxPrd	preTid
A	2	1	2
B	2	1	2
C	2	1	2
D	1	2	2
F	2	1	2

(c)

i	Sup	maxPrd	preTid
A	4	2	5
B	4	2	5
C	3	1	3
D	2	4	6
E	4	3	7
F	6	2	7
G	1	6	6

(d)

i	Sup	maxPrd	preTid
A	4	2	5
B	4	2	5
C	3	**4**	3
D	2	4	6
E	4	3	7
F	6	2	7
G	1	6	6

(e)

i	Sup	maxPrd	preTid
F	6	2	7
A	4	2	5
B	4	2	5
E	4	3	7
C	3	**4**	3
D	2	4	6

Fig. 1: (a)After scanning tid = 1 (b) After scanning tid = 2 (c) After scanning tid = 7 (d) After updating *maxPrd* (e) Final sorted *SPList* after discarding non-periodic 1-itemsets

track of *childList* pointers which represent the children of the node *c*. The main purpose of *NegTidTree* is to discover the occurrence behaviour with the support count. So, it is insufficient to maintain only the support count information as in NI-Tree. Therefore, instead of maintaining the support count, transaction ids are considered and these serve the dual purpose of finding support count as well as periodicity information. The transaction-ids(tid) are stored only in the leaf nodes, which help in the computation of periodicity. Moreover, literature[16], [17], [18], [19], [20], [24] show that keeping the tid information only in the leaf node is memory efficient rather than keeping the support count in all the nodes. The nodes of *NegTidTree* except the root node *r* can be divided into two types called ordinary nodes and tail nodes. The ordinary nodes represent only the negative item information. Whereas the tail nodes represented in the form *ILabel* [t_1,t_2,....,t_n] serve the purpose of discovering occurrence information. If t_x = $\{\neg i_1, \neg i_2, ..., \neg i_{Tail}\}$, then *ILabel* field of tail node represents the negative item $\neg i_{Tail}$. The tid-list [t_1,t_2,....,t_n] denotes all the transactions in which $\neg i_{Tail}$ is the tail node and is represented as $TidSet(\neg i_{Tail})$.

Lemma 1: A *tail node* in a *NegTidTree* inherits the properties of an ordinary node and not vice versa.

Proof: An ordinary node in a *NegTidTree* maintains the negative item information. In addition, it maintains *childList* pointers. Whereas, a tail node pointer maintains all this information along with the additional information regarding the tid-list information. Therefore, the tail node inherits the properties of an ordinary node, but an ordinary node does not represent all the information represented by the tail node.

C. Construction process of NegTidTree

The construction process of *NegTidTree* is given in Algorithm 2. Initially, the root node of the *NegTidTree* named *NTRoot* is created. The *TidSet* and *ISLabel* fields of *NTRoot* are initially empty. During the second scan of the transaction database, the non-periodic 1 itemsets are removed and the itemsets are sorted in ascending order with respect to the original database. Further, the Neg-Rep $\overline{t_x}$ of every transaction t_x is produced using *SPList.* To incorporate a top-down strategy, the *ISLabel* field of the *NTRoot* is updated by considering the newly added items in the Neg-Rep $\overline{t_x}$. *NTRoot.ISLabel* maintains descending order with respect to the original database. After the removal of non-periodic item $\{G\}$, the remaining negative items sorted in ascending order with respect to the support count in TD are represented in Table II and inserted into the *NegTidTree.* The insertion process is as shown in Procedure *Constructing-Negative-TidSet-Tree.* Similar to RP-Tree[7], the path is shared if there is any common path of transaction $\overline{t_x}$ exists in *NegTidTree.* The remaining path(if any) is attached at the end of the shared path. If there is no common path of transaction $\overline{t_x}$ present in *NegTidTree* then it is inserted as a new path. The tid of $\overline{t_x}$ is added to the *TidSet* of the tail node of the inserted path. Figure 2 shows the construction process of *NegTidTree* for the Neg-Rep database $\overline{TD}$ shown in Table II. The sorted Neg-Rep transactions are inserted into the tree similar to RP-Tree. The final prefix shared *NegTidTree* is shown in Figure 2(d). As shown in this Figure, *ISLabel* field of the root node contains the sorted k-itemset $\{$FABECD$\}$.

Let for every transaction $\overline{t_x} \in \overline{TD}$, negItemSet($\overline{t_x}$) represent all negative items of $\overline{t_x}$.

Property 1: As *NegTidTree* is a prefix shared tree, it maintains negItemSet($\overline{t_x}$) for every transaction $\overline{t_x} \in \overline{TD}$ at best by a single path. Further, the tid *x* is stored only once in $\neg i_{Tail}$ representing the tail node of this path.

Lemma 2: Let $\overline{TD}$ be represented by *NegTidTree* with root *NTRoot.* Then the

$$\sum_{\neg i_{Tail} \in NTRoot} Ti \quad is\ equal\ to |TD|.$$

Proof: According to Property 1, each negItemSet($\overline{t_x}$) of every transaction $\overline{t_x} \in \overline{TD}$ represented in $NegTidTree$ at best by a single path. The tid of $\overline{t_x}$ is stored only once in the $\neg i_{Tail}$ where $\neg i_{Tail}$ represents the tail node of $\overline{t_x}$. Therefore

$$\sum_{\neg i_{Tail} \in NTRoot} Ti \quad is\ equal\ to\ |\overline{TD}|.$$

Every transaction $\overline{t_x}$ of $\overline{TD}$ is constructed by considering Neg-Rep of each transaction in TD. This results in |TD|= $|\overline{TD}|$. Therefore

$$\sum_{\neg i_{Tail} \in NTRoot} Ti \quad is\ equal\ to |TD|.$$

D. Mining process of NegTidTree

This section demonstrates *PRPNegTidTreeMiner,* a top-down divide and conquer method to mine the low support periodic rare itemsets using the *NegTidTree*. The root node of *NegTidTree* contains all periodic rare database items. *PRP-NegTidTreeMiner,* traverses the items in the root node of the *NegTidTree* and extracts entire periodic rare itemsets from the given dataset. *PRPNegTidTreeMiner* uses *NegTidTreeMiner* to discover the periodic rare patterns of every item in the root

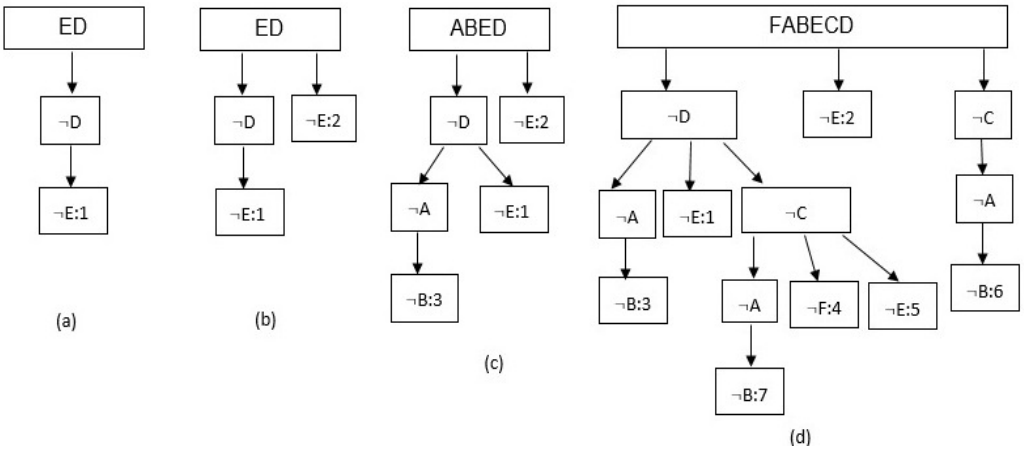


Fig. 2: (a)After inserting tid = 1 (b) After inserting tid = 2 (c) After inserting tid = 3 (d) After inserting tid = 7

Algorithm 2 Construction of *NegTidTree*

Input: *TD* - a transactional database, *maxSup* , *maxPer* - threshold values, *SPList*

Output: *NTRoot* - Root of *NegTidTree*

1: Create *NTRoot* as the root of *NegTidTree* and do the following initialization:
2: *NTRoot.*ISLabel $\leftarrow \{\emptyset\}$ and *NTRoot.*TidSet $\leftarrow \{\emptyset\}$
3: **for each** transaction $T \in$ TD with tid *tIDCur* **do**
4: Sort the items of T in descending order with respect to their Support
5: Create $NegTrans_T$ representing Neg-Rep of T using *SPList* and Discard non-periodic items from $NegTrans_T$
6: Add any missing item to *NTRoot.*ISLabel from $NegTrans_T$
7: Call **Constructing-Negative-Tidset-Tree** ($NegTrans_T$, NTRoot)
8: **end for each**
9: **Return** *NTRoot*
10: **Procedure Constructing-Negative-Tidset-Tree ([¬p|negP], curNode)**
11: Let ¬p represents the current negative item of $NegTrans_T$ and remaining negative items are represented by negP.
12: **if** curNode has a child *N* such that *curNode.*ILabel = ¬p **then** Select *N* as C
13: **else** Create a new node *N* as child of *curNode* with *N.*ILabel $\leftarrow$ ¬p
14: **end if**
15: Remove ¬p from [¬p|negP]
16: **if** negP is nonempty **then** call **Constructing-Negative-Tidset-Tree**(negP, N)
17: **else** Add *tIDCur* to TidSet(N)
18: **end if**
19: **end procedure**

node. Further the mining efficiency of *PRPNegTidTreeMiner* is improvised by designing a mining method called *NegTidTreeMinerFLP* which avoids unnecessary traversal of noise-itemsets.

The *PRPNegTidTreeMiner* deals with every item represented by *NTRoot.* Let symbol $\prec$ denote a "less frequent" item. In every iteration, *PRPNegTidTreeMiner* considers the next $\prec$ item with respect to the support count in the original dataset. In the database shown in Table I item D $\prec$ C $\prec$ A $\prec$ B $\prec$ E $\prec$ F. The proposed *PRPNegTidTreeMiner* uses a top-down depth-first strategy where the first iteration excludes item {D}. The procedure *NegTidTreeSubtraction* is invoked, which results in a new *NegTidTree* without {D}. *NegTidTreeMiner* method is called by passing the resultant *NegTidTree* and a NULL value as the initial item. A divide and conquer paradigm is used to generate all combinations of *PRPs* without {D}. In the next iteration, all those *PRPs* including item {D} and without item {C} are generated. This excluding and generation process is repeated by considering every item present in the *NTRoot.*

1) NegTidTreeMiner - A Top-down approach for mining *PRPs* : The mining process of every item in *NTRoot* is as shown in Algorithm 3. In every iteration, *NegTidTreeMiner* considers the next $\prec$ item *iN* with respect to the support count in the original dataset. The new *NegTidTree* is built by sending the current *NegTidTree* to the Procedure *NegTidTreeSubtraction.* The root node of the new *NegTidTree* is named as *newNTRoot.* The *ISLabel* field of *newNTRoot* is formed by removing the item *iN* from *ISLabel* of *NTRoot.* The nodes of the new *NegTidTree* are built by considering the first layer nodes of the current *NegTidTree* after removing the nodes with label *iN.* As shown in Procedure *TraverseSubTree,* the processed node *C* is attached to the new root node if it exists in the *ISLabel* field of the *newNTRoot.* During the process of attaching children, if a node *X* already exists matching the *ILabel* of *C* in the *newNTRoot* then only *TidSet* of *C* is added to the existing *TidSet* of the node *X.* Otherwise, *C* is added to *ChildList* of *newNTRoot.* If the processed node *C* is matching with label *iN* then, the node *C* is skipped and its *TidSet* is added to the *TidSet* of *newNTRoot.* Further, children of the node *C* are processed recursively as shown in Procedure *TraverseSubTree.* When this removal and attaching process is finished, the *TidSet* of *newNTRoot* represents the tid-list of the itemset represented by its *ISLabel* in the original database. Using *maxSup* and *maxPer* thresholds the *NegTidTreeMiner* extracts the periodic rare itemsets and stores it in *PRPList.* The removal process is continued in the path until a frequent itemset is obtained. Once a frequent itemset is found continuing

Algorithm 3 *NegTidTreeMiner* - Mining *PRPs*

Input: *NTRoot* - Root of *NegTidTree, maxSup, maxPer* - threshold values, *iP* - Previous 1-item considered
Output: *PRPList* - List of *PRPs*

1: **for each** item iN ∈ *NTRoot.*ISLabel, iP ≺ iN **do**
2: *newNTRoot* ← **NegTidTreeSubtraction** (NTRoot, iN)
3: currentCount ← Calculate Support from *newNTRoot.*TidSet
4: currentmaxPrd ← Calculate maximum periodicity from *newNTRoot.*TidSet
5: **if** (currentCount <maxSup) **then**
6: **if** (currentmaxPrd ≤ maxPer) **then**
7: PRPList ← PRPList ∪ *newNTRoot.*ISLabel
8: **end if**
9: NegTidTreeMiner(newNTRoot, maxSup, maxPer, iN, PRPList)
10: **end if**
11: **end for each**
12: **procedure NegTidTreeSubtraction(NTRoot,iN)**
13: newNTRoot.ISLabel ← *NTRoot.* ISLabel \ iN
14: newNTRoot.ChildList ← ∅
15: Call **TraverseSubTree**(NTRoot, newNTRoot, iN)
16: **return** newNTRoot
17: **Procedure TraverseSubTree(Q, P, iN)**
18: **for each** child C ∈ Q.ChildList **do**
19: **if** C.ILabel = iN **then**
20: add C.TidSet to P.TidSet
21: **TraverseSubTree**(C, P, iN)
22: **else** Add C to P.ChildList
23: **end if**
24: **end for each**
25: **end procedure**

in the path, further resulting in its subsets and according to the Anti-monotone property[39], the subsets are also frequent. The removal and the mining process consider all the items represented by *ISLabel* of the root node. For the current *NegTidTree* shown in Figure 2(d), the recursive node removal and the construction of the new *NegTidTree* process are shown in Figure 3. Item removal process starts from ≺ item {D}. At the beginning, *newNTRoot* is formed by removing the item {D} from the *ISLabel* field of the current *NegTidTree.* Further, the children of node {D} are attached to the *newNTRoot.* The resultant *NegTidTree* is as shown in Figure 3(b). It can be observed from Figure 3(b), as *TidSet* of {¬D} is empty, the *TidSet* of *newNTRoot* also results in ∅. This indicates itemset {FABEC} represented by *newNTRoot* is a non-existence itemset. This process continued recursively by considering the next ≺ item {C}, next {A} and further {B}. The resultant *NegTidTree* is shown in Figure 3(c), 3(d) and 3(e) respectively. After the removal of {C} and {A} the *newNTRoot* has resulted in non-existence itemsets {FABE} and {FBE} as shown in Figure 3(c) and 3(d) respectively. During the removal process of {B}, the *TidSet* of {¬B} is added to the *TidSet* of the *newNTRoot* as shown in Figure 3(d). The *TidSet* {3,6,7} of *newNTRoot* represents the tidlist of itemset {FE} in the original database. According to $maxSup$ and $maxPer$ thresholds {FE} is found to be a PRP and it is added to *PRPList.*This iteration has generated all the periodic rare itemsets without {D}. The current iteration completes and the next iteration starts by considering the next ≺ item {C}. Further, the recursive removal process continues which generates all itemsets with item {D} and without item {C}. This process is continued for all the remaining items in the *ISLabel* of the root node. Finally, *PRPList* contains all the resultant *PRPs.*

Theorem 2: Given a *NegTidTree* with root *NTRoot* representing an itemset *P,* then *P.*interSup is equal to |TidSet(NTRoot)|
Proof: Given a transaction t_x∈TD, if $\overline{t_x} \in \overline{TD} \cup \neg P \neq \emptyset$, it means at least $\neg iP_{Tail}$ the tail node of $\overline{t_x}$ must be present in the *NegTidTree.* Otherwise, all the nodes corresponding to the path of $\overline{t_x}$ will be already removed. By Definition 6, the number of such transactions is equal to $\neg P$.jointSup and it can be represented by

$$\sum_{\neg iP_{Tail} \in NTRoot} Ti$$

The *TidSet* of removed nodes are added to TidSet of *NTRoot.* According to Lemma 2,

$$\sum_{\neg i_{Tail} \in NTRoot} Ti \quad is\ equal\ to\ |TD|.$$

This results in

$$\sum_{\neg iP_{Tail} \in NTRoot} Ti \quad + \quad TidSet(NTRoot)\ is\ equal\ to\ |TD|.$$

$$If \sum_{\neg iP_{Tail} \in NTRoot} Ti$$

is $\neg P$.jointSup then according to Theorem 1, TidSet(NTRoot) represents *P.*interSup.

2) NegTidTreeMinerFLP - Mining *NegTidTree* by removing noise itemsets: The *NegTidTreeMiner* proposed in section III-D1 deals with every item represented by *NTRoot.* A divide and conquer paradigm is used to generate all *PRPs* by considering every item present in the *NTRoot.* From the experimentation carried out by considering the real-world datasets, it is observed that the number of non-existent itemsets is large. Before reaching the required *PRPs,* a bundle of non-existent itemsets has to be unnecessarily traversed. To reduce the number of non-existent itemsets traversed, a new mining approach called *NegTidTreeMinerFLP* is designed by improvising the mining method of *NegTidTreeMiner. NegTidTreeMinerFLP* is demonstrated in Algorithm 4. In *NegTidTreeMiner,* when the current node is a non-existent itemset then the remaining *NegTidTree* are generated recursively by removing all the items having greater support than the current item removed. Instead of this in *NegTidTreeMinerFLP,* when the current itemset is a noise itemset (itemsets having support count 0 or non-periodic itemsets), then the procedure *NegTidTreeMinerRemoveNoise* is invoked. *NegTidTreeMinerRemoveNoise* ensures the remaining itemsets in *NegTidTree* are generated by removing the items present only in the first layer having support greater than the current item removed. To reflect this the *NegTidTreeMiner* is improvised in *NegTidTreeMinerRemoveNoise* by modifying the line 1 as "for each item iN ∈ NTRoot.childList, iP ≺ iN". When huge datasets are taken

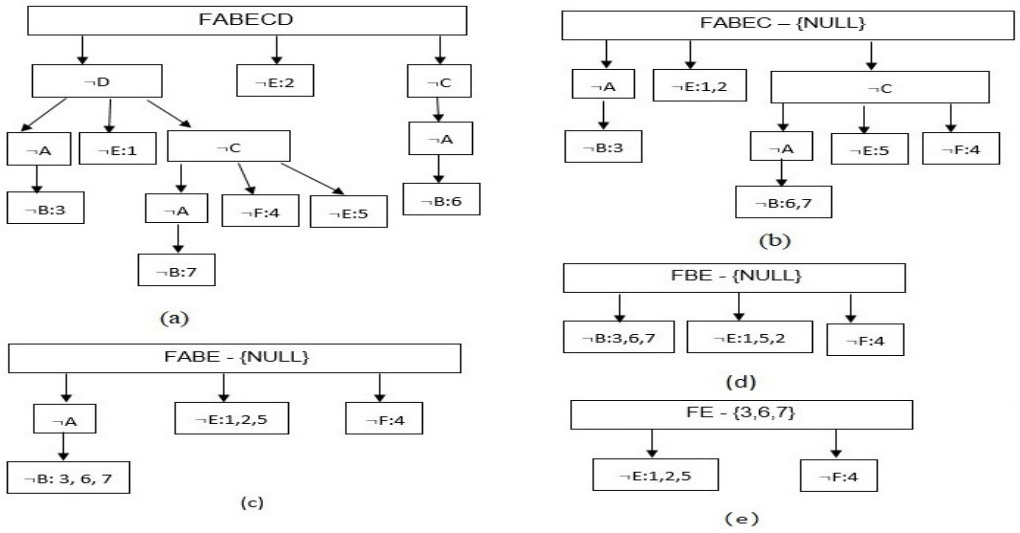


Fig. 3: The subsequent *NegTidTree* created during mining process for the final *NegTidTree* shown in 3(a) Resultant *NegTidTree* after removal of $\neg D$, $\neg C$, $\neg A$ and $\neg B$ is shown in (b),(c),(d) and (e) respectively

Algorithm 4 *NegTidTreeMinerFLP* - Mining *PRPs*

Input: *NTRoot* - Root of *NegTidTree, maxSup, maxPer* - threshold values, *iP* - Previous 1-item considered
Output: *PRPList* - List of *PRPs*

1: **for each** item iN ∈ *NTRoot.*ISLabel, iP ≺ iN **do**
2: *newNTRoot* ← **NegTidTreeSubtraction** (NTRoot, iN)
3: currentCount ← Calculate Support from *newNTRoot.*TidSet
4: currentmaxPrd ← Calculate maximum periodicity from *newNTRoot.*TidSet
5: **if** (currentCount <*maxSup*) **then**
6: **if** ((currentCount = 0) ∨ (currentmaxPrd >*maxPer*)) **then**
7: NegTidTreeMinerRemoveNoise(newNTRoot, maxSup, maxPer, iN, PRPList)
8: **else**
9: PRPList ← PRPList ∪ *newNTRoot.*ISLabel
10: NegTidTreeMinerFLP(newNTRoot, maxSup, maxPer, iN, PRPList)
11: **end if**
12: **end if**
13: **end for each**

into account, there is a possibility that the removal of items not present in the first layer may also generate *PRPs.* This is handled during the mining process, by temporarily storing the items that are missing in the first layer in an array. As soon as a *PRP* is generated, the remaining subsets are generated using the items in the temporary array. This avoids the non-existent itemset traversal and ensures the complete generation of *PRPs.*

IV. Results and Discussion

PRPNegTidTreeMiner is the first tree-based algorithm to employ a top-down strategy in the process of mining low-support periodic patterns. The proposed algorithm is implemented in the Java platform and is tested for various datasets by considering different threshold values. It accepts *maxSup* and *maxPer* threshold values from the user and discovers all the patterns that satisfy the support as well as periodicity threshold values. *PRPNegTidTreeMiner* proposed *NegTidTreeMiner* and an improvised mining method *NegTidTreeMinerFLP.* In the experiments it is referred to as *NTTMDFS* and *NTTMFLP* respectively. It is found that the number of *PRPs* discovered by both the mining methods are the same. The methods vary with respect to the time taken for execution and it is analyzed in this section. Both of these methods are executed in the system with configuration Intel(R) Core(TM) i5-7400 CPU@3.00GHz with 8GB RAM running Windows10 Enterprise.

A. Datasets

For the experimentation, four real datasets with a varying number of transactions are downloaded from the "frequent itemset mining dataset repository" (http://fimi.ua.ac.be/data/). *Chess* is a small dataset with total transactions of about 3k, *Mushrooms* is a dataset with total transactions of about 8k, *Pumsb* dataset is about 49k *and Connect* dataset is about 67k transactions. The descriptions of the dataset are given in Table III. The number of transactions is represented by |T|, the count of unique items by |I|, average transaction length by |L|. If the maximum transaction length |L| is very high then it will generate a huge set of rare itemsets because of which these algorithms are not able to complete the mining task. So the value considered for the experimentation is mentioned in the runtime comparison.

TABLE III: Description of datasets

Dataset	|T|	|I|	|L|
Chess	3,196	75	37
Mushrooms	8,124	119	23
Pumsb	49,046	2,113	74
Connect	67,557	129	43

B. Runtime comparison

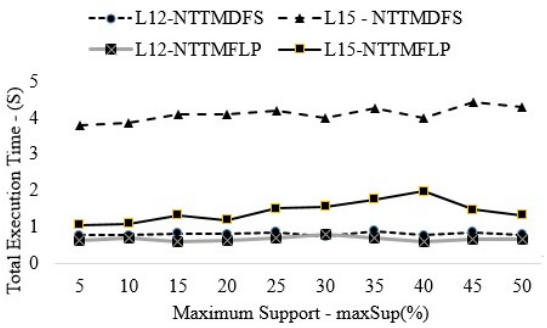
The runtime performance of the algorithms is obtained by considering different datasets for various maximum support

and periodicity threshold values. The total execution time taken by the algorithms for various thresholds is shown in Figures 4, 5, 6, 7 and 8. In these figures, the X-axis represents *maxSup* threshold values and the Y-axis represents the runtime in seconds. It has been found that *NegTidTreeMiner* is faster than *NegTidTreeMinerFLP* for smaller datasets which generate more *PRPs.* However, as the dataset size increases, *NegTidTreeMinerFLP* performs better. As the same *NegTidTree* is used for both methods, the major contribution to the performance improvement is the mining approach. The number of *PRPs* generated and periodic 1-itemsets extracted after the tree construction phase by various datasets for different thresholds are shown in Figures 9, 10 and 11 respectively.

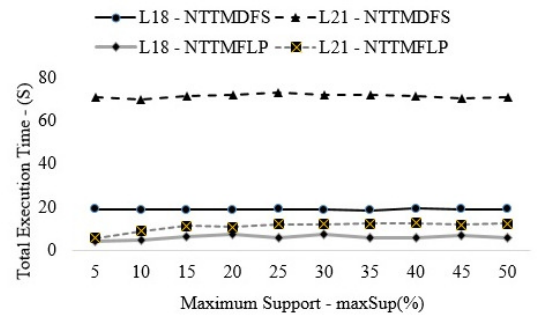
As the top-down approach is suitable for producing the low threshold periodic rare patterns, all the experiments are carried out by keeping the *maxSup* and *maxPer* threshold values in the low range. In all the experiments *maxSup* threshold value is varied in the range of 5% to 50%.

1) Results on varying dimensions of dataset: The total execution time taken by both algorithms for different dimensions of datasets is shown in Figures 4 and 5. The experiments are carried out by varying the *maxPer* threshold in the range of 10% to 30%. The runtime performance for *Mushrooms* dataset is shown by setting the value of *L* as 12, 15, 18, and 21 by keeping *maxPer* constant as 30%. Compared to *NegTidTreeMiner, NegTidTreeMinerFLP* has shown a performance improvement of 17%, 65%, 68% and 84% respectively as shown in Figures 4(a) and 4(b). Figure 4(c) and 4(d) depict similar observations for *Connect* dataset. The performance of *NegTidTreeMinerFLP* is improved by 11%, 14%, 50%, and 59% for the *L* values 6, 10, 12, and 15 respectively when *maxPer* is kept constant as 10%. The *L* value plays a major role in deciding the size of *NegTidTree.* As shown in Figure 11, when the *L* value is increased, the possibility of periodic 1-itemsets increases which results in an increase in the number of nodes in the tree and the number of non-existing itemsets. *NegTidTreeMinerFLP* avoids the non-existing item traversal showing a better performance compared to *NegTidTreeMiner.* It is also observed that when the value of *L* is increased beyond the values taken for experimentation *NegTidTreeMiner* takes a longer time for the execution.

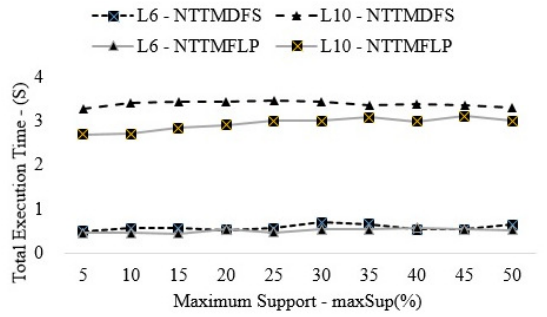
The *Pumsb* dataset has the same characteristics as the *Connect* dataset. A large number of periodic 1-itemsets are generated as shown in Figure 11 and *NegTidTreeMiner* takes longer time when compared to *NegTidTreeMinerFLP.* Therefore execution time of *NegTidTreeMinerFLP* is shown in all the figures. The time consumed by *NegTidTreeMinerFLP* is shown in Figure 5(c) for *L* values 6 and 9 respectively when *maxPer* is kept constant as 10% and 20%. Alternately, for a small *Chess* dataset, *NegTidTreeMiner* always performs better than *NegTidTreeMinerFLP.* For a smaller dataset it takes less time to build *NegTidTree* tree recursively and mine it further. The runtime performance is displayed in Figures 5(a) and (b) where the value of *L* is varied as 10, 12, and 15 while *maxPer* is kept constant at 10% and 30%. Even though *Chess* is a smaller dataset compared to *Mushrooms,* it can be observed that the number of periodic 1-itemsets generated is around 20 in both datasets. The number of *PRPs* extracted by *Mushrooms* dataset are around 1000 and 4000



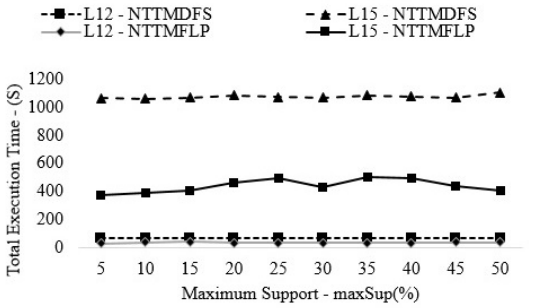
(a) Mushrooms,maxPer = 30%



(b) Mushrooms,maxPer = 30%



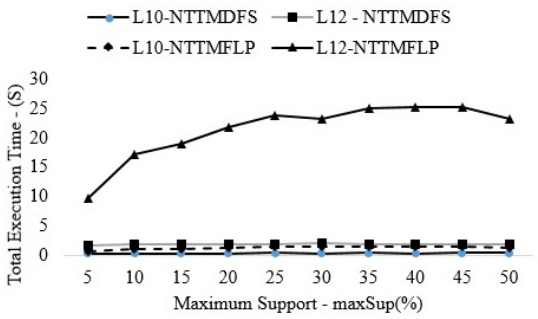
(c) Connect,maxPer = 10%



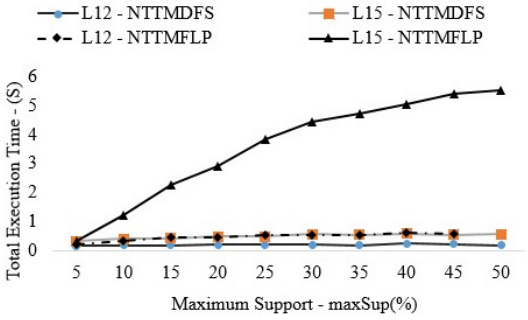
(d) Connect,maxPer = 10%

Fig. 4: Runtime comparison on varying dimensions of *Mushrooms* and *Connect* datasets

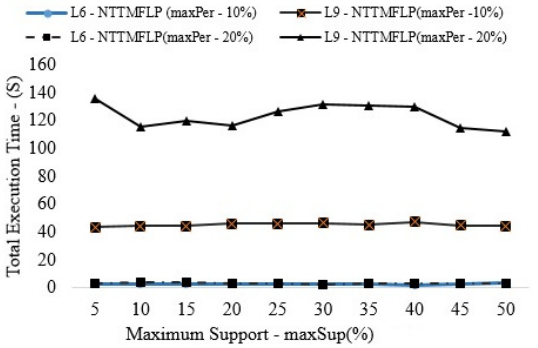
respectively for *maxPer* 30% and 40% and L value 12 as shown in Figure 9(c). Furthermore, for the same thresholds and L value *Chess* dataset produces more number of *PRPs* around 23,000 to 40,000 and 55,000 to 75,000 as shown in Figure 10(c). This shows that the non-existing itemsets are lesser in *Chess dataset,* as a result, *NegTidTreeMiner* performs well. In contrast, the *NegTidTreeMinerFLP* takes longer to verify the first layer children and generate any missing itemsets. *NegTidTreeMiner* has demonstrated a per-



(a) Chess,maxPer = 30%



(b) Chess,maxPer = 10%



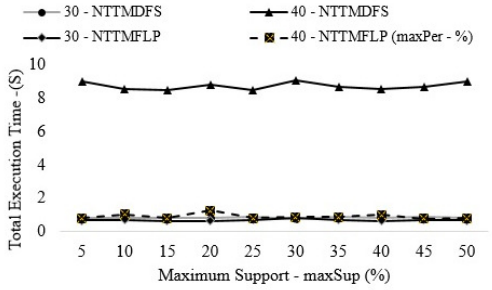
(c) Pumsb,maxPer = 10 and 20%

Fig. 5: Runtime comparison on varying dimensions of *Chess* and *Pumsb* datasets

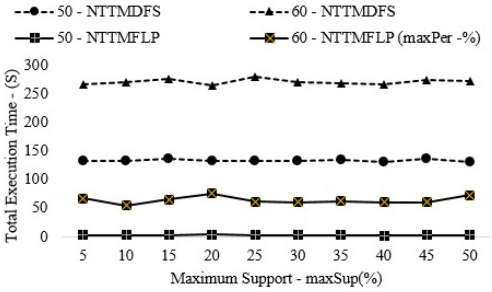
formance gain of 70% and 90% when *maxPer* is set to 30% compared to *NegTidTreeMinerFLP*. Similarly, *NegTidTreeMiner* has demonstrated speed improvements of 51% and 76% when *maxPer* is set to 10%.

Influence of varying dimensions of dataset: The increase in the value of *L* exhibits an increase in the number of periodic 1-itemsets as shown in Figure 11. Consequently, it generates more number of *PRPs* as shown in Figures 9(a),(b) and 10(a),(b). The increase in the nodes of *NegTidTree* results in repeated tree construction and recursive traversals. This leads to an increase in the mining time which can be observed from Figures 4 and 5.

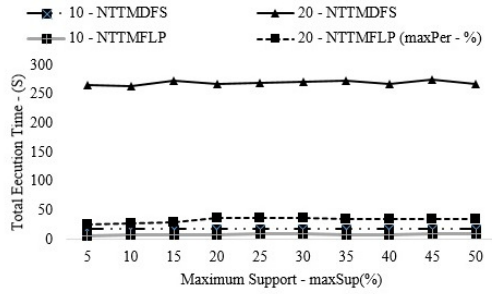
2) Results on varying maxPer threshold value: The runtime performance on varying the *maxPer* threshold value by keeping *L* value constant is shown in figures 6 and 7. The *maxPer* value is varied from 30% to 60% and for *Mushrooms* dataset. *NegTidTreeMinerFLP* shows a better performance of 17%, 89%, 98% and 76% respectively than



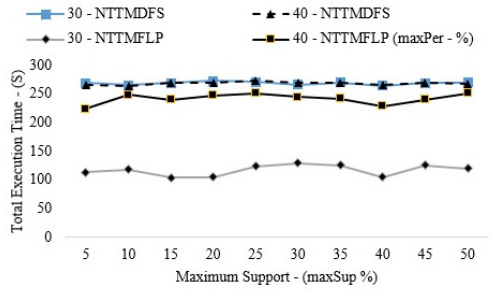
(a) Mushrooms,L = 12



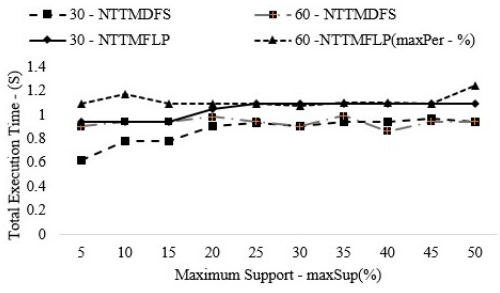
(b) Mushrooms,L = 12



(c) Connect,L = 11

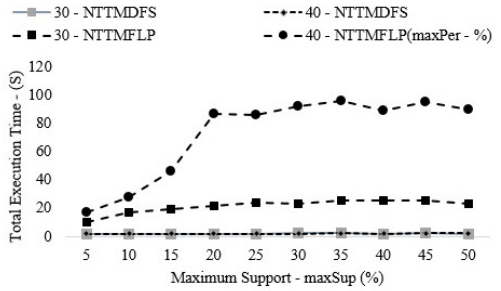


(d) Connect,L = 11

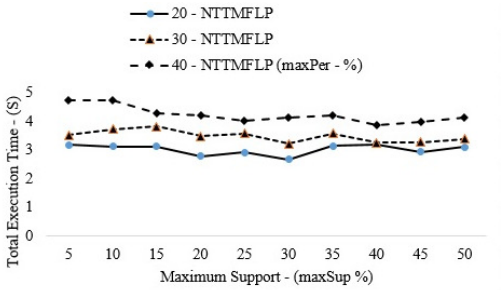


(e) Connect,L = 6

Fig. 6: Runtime comparison on varying periodicity thresholds for *Mushrooms* and *Connect* datasets



(a) Chess,L = 12

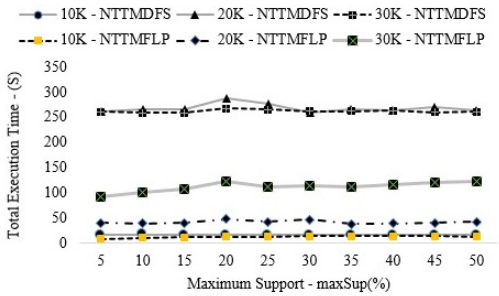


(b) Pumsb,L = 6

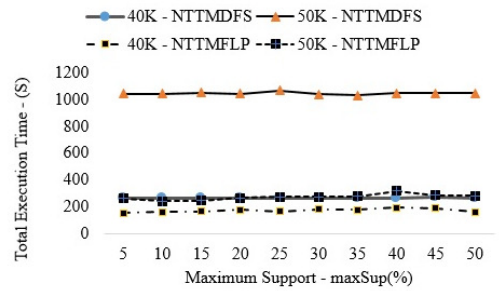
Fig. 7: Runtime comparison on varying periodicity thresholds for *Chess* and *Pumsb* datasets

NegTidTreeMiner as shown in Figures 6(a) and 6(b). For the *Connect* dataset, the *maxPer* threshold value is varied in the low range of 10%, 20%, 30%, and 40%. Since *Connect* is a large dataset, *NegTidTreeMiner* could not complete the execution for the threshold values greater than 40%. The performance of *NegTidTreeMinerFLP* was improved by 50%, 88%, 56% and 10% respectively as shown in Figures 6(c) and 6(d). The number of periodic 1-itemsets grows with the periodicity threshold value thereby increasing the nodes of the *NegTidTree and non-existing itemsets.* In comparison to *NegTidTreeMiner, NegTidTreeMinerFLP* performs better since it avoids traversing non-existent items. It should be noticed that when the *maxPer* threshold is kept between 20% and 30%, the performance improvement is greater. As *maxPer* threshold is increased further there is a decline in performance improvement. This is because the number of non-existent items decreases which increases the recursive tree construction and traversal. As a result, the mining time increases, and the performance is reduced. Similar results are obtained for *Pumsb* dataset. Figure 7(b) shows the result where *maxPer* threshold value is varied between 20% and 40% by keeping *L* value as constant 6.

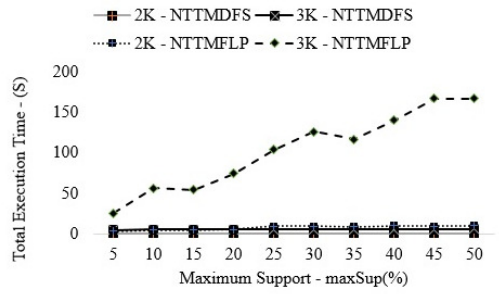
On the contrary, *NegTidTree* is built faster recursively for *Chess* dataset and it takes lesser mining time. Figure 10(c) shows the *PRPs* generated by *Chess* dataset is more which implies the non-existing itemsets are lesser. As non-existing itemsets are lesser, *NegTidTreeMinerFLP* takes longer to verify the first layer children and generate any missing itemsets resulting in prolonged execution time. The runtime performance is displayed in Figure 7 for the *Chess* dataset, where the value of *maxPer* is varied as 30% and 40% while *L* value is kept constant at 12. *NegTidTreeMiner* has demonstrated a performance gain of 91% and 97% when *maxPer* is set to 30% and 40% respectively compared to *NegTidTreeMinerFLP.* Similar observations are depicted in Figure 6(e)



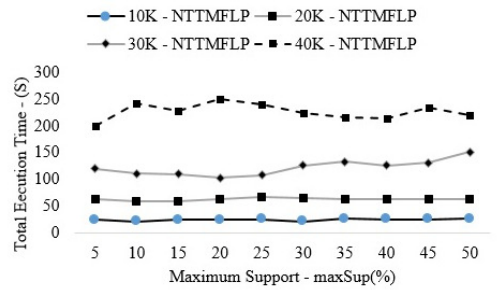
(a) Connect,L =12, maxPer = 20%



(b) Connect,L =12, maxPer = 20%



(c) Chess,L = 15, maxPer = 20%

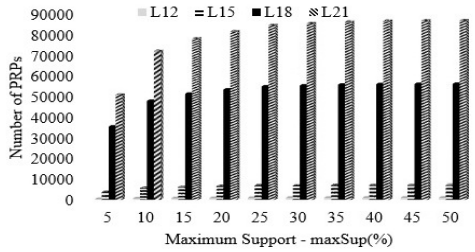


(d) Pumsb,L = 9, maxPer = 30%

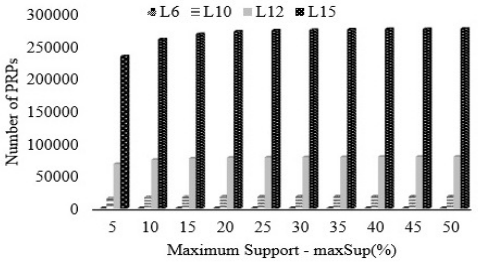
Fig. 8: Runtime comparison on a varying number of tansactions

where a small set of *Connect* dataset is constructed by taking a *L* value constant as 6, the number of transactions is restricted to 5K and *maxPer* is varied as 30% and 60%. In both the cases *NegTidTreeMiner* shows a performance gain of 17% and 16% compared to *NegTidTreeMinerFLP* when *maxPer* is set to 30% and 60% respectively. However, the improvement is lesser because the number of *PRPs* generated is very less compared to *Chess* as displayed in Figure 9(f).

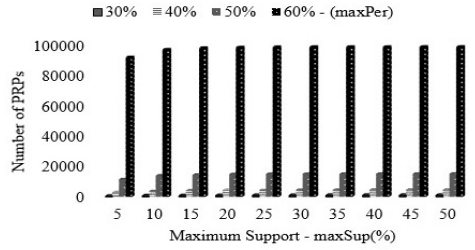
Influence of varying *maxPer* threshold: The increase in *maxPer* threshold value increases the number of periodic 1-itemsets and *PRPs* as shown in Figure 11 and Figures 9(c),(d),(e),(f) and 10(c),(d). This increases the number of recursive tree construction and traversal which influences the mining performance.



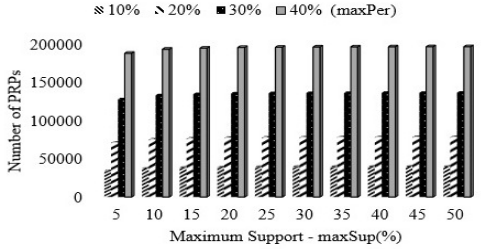
(a) Mushrooms,maxPer = 30%



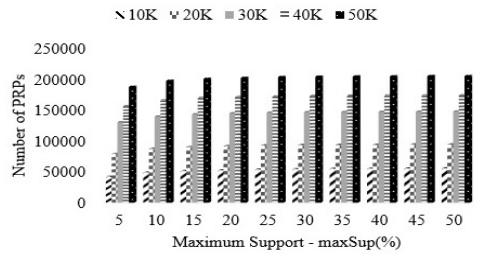
(b) Connect,maxPer = 10%



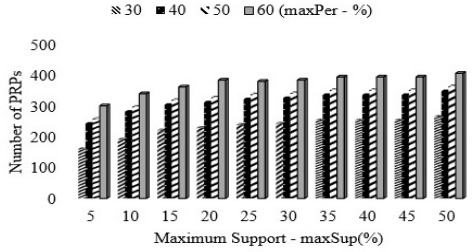
(c) Mushroom,L = 12



(d) Connect,L = 11

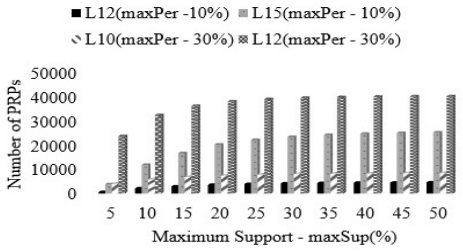


(e) Connect,L = 12, *maxPer = 20%*

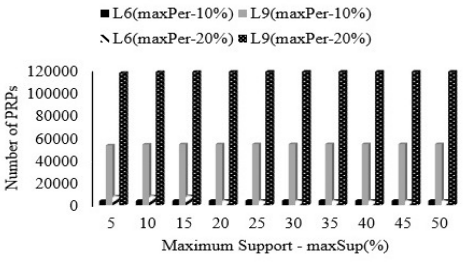


(f) Connect,L = 6, *maxPer = 30 to 60%*

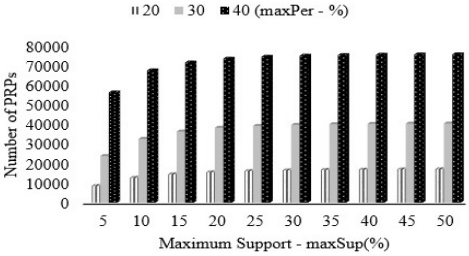
Fig. 9: Number of PRPs generated for *Mushroom* and *Connect* datasets



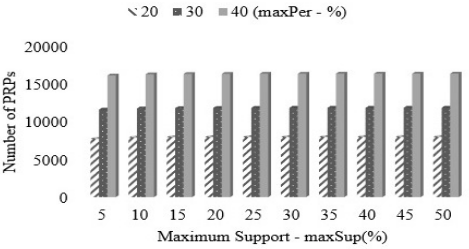
(a) Chess,maxPer = 10 and 30%



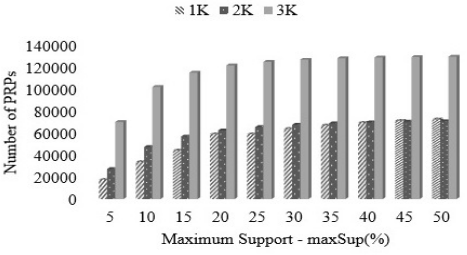
(b) Pumsb,maxPer = 10 and 20%



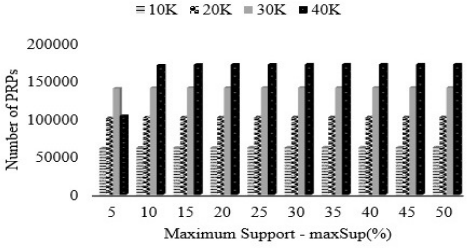
(c) Chess,L = 12



(d) Pumsb,L = 6



(e) Chess,L = 15, *maxPer = 20%*



(f) Pumsb,L = 9, *maxPer = 30 %*

Fig. 10: Number of PRPs generated for *Chess* and *Pumsb* datasets

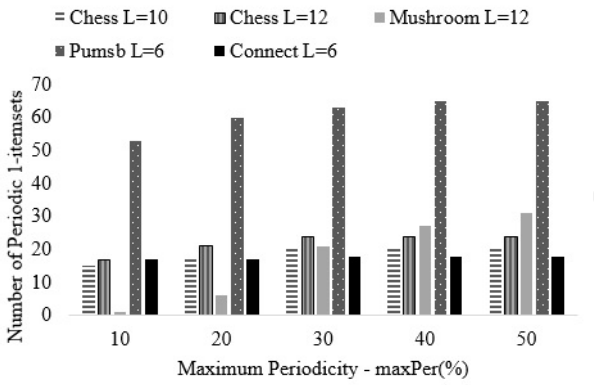


Fig. 11: Number of periodic 1-itemsets generated by various datasets for different maximum periodicity

3) Results on varying the number of transactions: The runtime performance of the algorithms on *Connect* dataset by varying the number of transactions is shown in figures 8(a) and (b) where *L* value is set to 12 and *maxPer* is set to 20%. The number of transactions varied from 10k to 50k in steps of 10k. It can be observed that *NegTidTreeMinerFLP* performed 21%, 84%, 57%, 35%, and 74% better than *NegTidTreeMiner.* Due to variations in the number of *PRPs* and non-existing itemsets generated, *NegTidTreeMinerFLP* performance is also varied. Similar characteristics can also be seen in the *Pumsb* dataset as shown in Figure 8(d). Here, while keeping the *L* value constant at 9 and *maxPer* value constant at 30%, the number of transactions varied from 10,000 to 40,000. For *Chess* dataset, the number of transactions is varied from 2k to 3k by setting the *L* value constant as 15 and *maxPer* kept constant as 20%. It can be observed from Figure 8(c) the performance of *NegTidTreeMiner* was improved by 87%, and 92% respectively when compared to *NegTidTreeMinerFLP.*

Influence of varying number of transactions: The increase in the number of transactions shows an increase in the number of *PRPs* generated which is evident in Figure 9(e), 10(e) and (f). As the number of transactions increases, periodic 1-itemsets are increased. For instance, in the case of *Pumsb,* periodic 1-itemset is increased from 78 to 94 when the number of transactions varied from 10k to 40k. This increases the number of recursive tree construction and traversal which influences the mining performance.

4) Influence of varying maxSup threshold value: An increase in *maxSup* threshold value increases the mining time slowly as observed in Figures 4, 5, 6, 7, and 8. Even though the number of rare 1-itemsets increases, the number of periodic 1-itemsets remains the same. The periodic 1-itemsets created for the *Connect* dataset is 24 when *maxPer* is set to 30%, *L* value to 12, and *maxSup* is varied. As a result, the nodes in *NegTidTree* remain the same. There is a small increase in the number of *PRPs* generated which influences the mining time slowly as shown in Figures 9 and 10.

V. Conclusion

PRPNegTidTreeMiner is the first top-down tree-based strategy proposed in this paper to discover rare periodic patterns. A novel tree structure, *NegTidTree,* is presented which serves the dual purpose of finding support count as well as periodicity information. The two distinct mining techniques *NegTidTreeMiner* and *NegTidTreeMinerFLP* proposed, are able to mine periodic rare patterns for various *maxSup* and *maxPer* low threshold values. The study demonstrates that as the *maxPer or L* value is raised, the quantity of periodic rare itemsets increases. The findings exhibit that *NegTidTreeMiner* performs better when compared to *NegTidTreeMinerFLP* for smaller datasets which produces more number of *PRPs.* As the database size increases, *NegTidTreeMinerFLP* consistently outperforms *NegTidTreeMiner.* For low thresholds, as the number of non-existing itemsets rises, the performance of *NegTidTreeMinerFLP* improves. In spite of an impressive performance, the algorithm has its limitation based on the size of the neg-rep database for sparse datasets. The limitation can be overcome by a combined strategy using both top-down and bottom-up tree traversal methods to discover *PRPs*. The proposed work can be extended to mine *PRPs* from the stream data using *NegTidTree.*

References

[1] L. Szathmary, A. Napoli, and P. Valtchev, "Towards rare itemset mining," in *19th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2007)*, vol. 1. IEEE, 2007, pp. 305–312.

[2] L. Szathmary, P. Valtchev, and A. Napoli, "Generating rare association rules using the minimal rare itemsets family," *International journal of software and informatics*, vol. 4, pp. 219–238, 2010.

[3] S. Bouasker and S. Ben Yahia, "Key correlation mining by simultaneous monotone and anti-monotone constraints checking," in *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, 2015, pp. 851–856.

[4] S. Bouasker, T. Hamrouni, and S. B. Yahia, "New exact concise representation of rare correlated patterns: Application to intrusion detection," in *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer, 2012, pp. 61–72.

[5] A. Borah and B. Nath, "Identifying risk factors for adverse diseases using dynamic rare association rule mining," *Expert systems with applications*, vol. 113, pp. 233–263, 2018.

[6] P. K. Shwetha Rai, Geetha M. and G. B., "Binary count tree: An efficient and compact structure for mining rare and frequent itemsets," *Engineered Science*, vol. 17, pp. 185–194, 2022.

[7] S. Tsang, Y. S. Koh, and G. Dobbie, "Rp-tree: rare pattern tree mining," in *International Conference on Data Warehousing and Knowledge Discovery*. Springer, 2011, pp. 277–288.

[8] J. Han, J. Pei, and Y. Yin, "Mining frequent patterns without candidate generation," in *ACM sigmod record*, vol. 29. ACM, 2000, pp. 1–12.

[9] A. Borah and B. Nath, *Mining Rare Patterns Using Hyper-Linked Data Structure*, 12 2017, pp. 467–472.

[10] L. Szathmary, P. Valtchev, A. Napoli, and R. Godin, "Efficient vertical mining of minimal rare itemsets." in *CLA*. Citeseer, 2012, pp. 269–280.

[11] A. Gupta, A. Mittal, and A. Bhattacharya, "Minimally infrequent itemset mining using pattern-growth paradigm and residual trees," *arXiv preprint arXiv:1207.4958*, 2012.

[12] M. Adda, L. Wu, S. White, and Y. Feng, "Pattern detection with rare item-set mining," *arXiv preprint arXiv:1209.3089*, 2012.

[13] L. Troiano, G. Scibelli, and C. Birtolo, "A fast algorithm for mining rare itemsets," in *2009 Ninth International Conference on Intelligent Systems Design and Applications*. IEEE, 2009, pp. 1149–1155.

[14] L. Troiano and G. Scibelli, "A time-efficient breadth-first level-wise lattice-traversal algorithm to discover rare itemsets," *Data Mining and Knowledge Discovery*, vol. 28, no. 3, pp. 773–807, 2014.

[15] Y. Lu, F. Richter, and T. Seidl, "Efficient infrequent pattern mining using negative itemset tree," in *Complex Pattern Mining*. Springer, 2020, pp. 1–16.

[16] S. K. Tanbeer, C. F. Ahmed, B.-S. Jeong, and Y.-K. Lee, "Mining regular patterns in transactional databases," *IEICE TRANSACTIONS on Information and Systems*, vol. 91, no. 11, pp. 2568–2577, 2008.

[17] S. K. Tanbeer, C. F. Ahmed, and B.-S. Jeong, "Mining regular patterns in data streams," in *International Conference on Database Systems for Advanced Applications*. Springer, 2010, pp. 399–413.

[18] S. K. Tanbeer, M. M. Hassan, A. Almogren, M. Zuair, and B.-S. Jeong, "Scalable regular pattern mining in evolving body sensor data," *Future Generation Computer Systems*, vol. 75, pp. 172–186, 2017.

[19] M. M. Rashid, M. R. Karim, B.-S. Jeong, and H.-J. Choi, "Efficient mining regularly frequent patterns in transactional databases," in *International Conference on Database Systems for Advanced Applications*. Springer, 2012, pp. 258–271.

[20] M. M. Rashid, I. Gondal, and J. Kamruzzaman, "Regularly frequent patterns mining from sensor data stream," in *International Conference on Neural Information Processing*. Springer, 2013, pp. 417–424.

[21] R. U. Kiran, M. Kitsuregawa, and P. K. Reddy, "Efficient discovery of periodic-frequent patterns in very large databases," *Journal of Systems and Software*, vol. 112, pp. 110–121, 2016.

[22] R. U. Kiran and M. Kitsuregawa, "Novel techniques to reduce search space in periodic-frequent pattern mining," in *International Conference on Database Systems for Advanced Applications*. Springer, 2014, pp. 377–391.

[23] J. Venkatesh, R. U. Kiran, P. K. Reddy, and M. Kitsuregawa, "Discovering periodic-correlated patterns in temporal databases," in *Transactions on Large-Scale Data-and Knowledge-Centered Systems XXXVIII*. Springer, 2018, pp. 146–172.

[24] P. Fournier-Viger, P. Yang, J. C.-W. Lin, and R. U. Kiran, "Discovering stable periodic-frequent patterns in transactional data," in *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*. Springer, 2019, pp. 230–244.

[25] P. Fournier-Viger, Y. Wang, P. Yang, J. C.-W. Lin, U. Yun, and R. U. Kiran, "Tspin: Mining top-k stable periodic patterns," *Applied Intelligence*, vol. 52, no. 6, p. 6917–6938, apr 2022. [Online]. Available: https://doi.org/10.1007/s10489-020-02181-6

[26] K. Amphawan, P. Lenca, and A. Surarerks, "Efficient mining top-k regular-frequent itemset using compressed tidsets," in *New Frontiers in Applied Data Mining*. Springer Berlin Heidelberg, 2012, pp. 124–135.

[27] T. Mesama and K. Amphawan, "Mining top-k frequent-regular itemsets from data streams based on sliding window technique," in *2018 5th International Conference on Advanced Informatics: Concept Theory and Applications (ICAICTA)*, 2018, pp. 224–230.

[28] K. Amphawan, P. Lenca, A. Jitpattanakul, and A. Surarerks, "Mining high utility itemsets with regular occurrence." *Journal of ICT Research & Applications*, vol. 10, no. 2, 2016.

[29] S. Laoviboon and K. Amphawan, "Mining high-utility itemsets with irregular occurrence," in *2017 9th International Conference on Knowledge and Smart Technology (KST)*. IEEE, 2017, pp. 89–94.

[30] P. Fournier-Viger, J. C.-W. Lin, Q.-H. Duong, and T.-L. Dam, "Phm: mining periodic high-utility itemsets," in *Industrial conference on data mining*. Springer, 2016, pp. 64–79.

[31] R. U. Kiran, P. Veena, P. Ravikumar, C. Saideep, K. Zettsu, H. Shang, M. Toyoda, M. Kitsuregawa, and P. K. Reddy, "Efficient discovery of partial periodic patterns in large temporal databases," *Electronics*, vol. 11, no. 10, 2022. [Online]. Available: https://www.mdpi.com/2079-9292/11/10/1523

[32] P. Veena, L. Palla, B. chithra, and U. Rage, *Discovering Partial Periodic Patterns in Temporal Databases*, 10 2021, pp. 69–79.

[33] R. U. Kiran, J. Venkatesh, M. Toyoda, M. Kitsuregawa, and P. K. Reddy, "Discovering partial periodic-frequent patterns in a transactional database," *Journal of Systems and Software*, vol. 125, pp. 170–182, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121216302382

[34] A. R. M. Earl F Glynn, Jie Chen, "Detecting periodic patterns in unevenly spaced gene expression time series using lomb-scargle periodograms," *Bioinformatics*, vol. 22, pp. 310–316, 2006.

[35] c. k. khaleel m, dash g. and khan m., "medical data mining for discovering periodically frequent diseases from transactional databases," in *computational intelligence in data mining*, vol. 1. Springer, 2015, pp. 87–96.

[36] F. Yi, L. Yin, H. Wen, H. Zhu, L. Sun, and G. Li, "Mining human periodic behaviors using mobility intention and relative entropy," in *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer, 2018, pp. 488–499.

[37] A. C. M. Fong, B. Zhou, S. C. Hui, G. Y. Hong, and T. A. Do, "Web content recommender system based on consumer behavior modeling," *IEEE Transactions on Consumer Electronics*, vol. 57, no. 2, pp. 962–969, 2011.

[38] P. Fournier-Viger, P. Yang, Z. Li, J. C.-W. Lin, and R. U. Kiran, "Discovering rare correlated periodic patterns in multiple sequences," *Data & Knowledge Engineering*, vol. 126, p. 101733, 2020.

[39] T. I. Rakesh Agrawal and A. Swami, "Mining association rules between sets of items in large databases," in *ACM SIGMOD Record*, vol. 22, 1993, pp. 207–216.