











Towards Efficient Discovery of Stable Periodic Patterns in Big Columnar Temporal Databases

Hong N. Dao , Penugonda Ravikumar , P. Likitha ,
Bathala Venus Vikranth Raj , R. Uday Kiran  , Yutaka Watanobe ,
and Incheon Paik 

The University of Aizu, Aizuwakamatsu, Fukushima, Japan
{udayrage,watanobe,paik}@u-aizu.ac.jp

Abstract. Extracting stable periodic-frequent patterns in very large temporal databases is a key task in big data analytics. Existing studies have mainly concentrated on discovering these patterns only in row temporal databases, and completely ignored the existence of these patterns in columnar databases, which are widely becoming popular for storing big data. In this paper we propose an efficient algorithm, Stable Periodic-frequent Pattern-Equivalence CLass Transformation (SPP-ECLAT), to find the desired patterns in a columnar temporal database. Empirical results demonstrate that the SPP-ECLAT algorithm is much faster and consumes significantly less memory than the state-of-the-art SPP-growth algorithm on sparse and dense databases.

Keywords: Columnar databases · Periodic patterns · Pattern mining

1 Introduction

Databases are broadly classified into two types based on the layout of data recording on a storage device, namely *row databases* and *columnar databases*¹. Row databases store data as records, maintaining the complete data associated with a record in a storage device next to each other. These databases are primarily based on ACID² properties and are designed to read and write rows fast. MySQL and Postgres are two examples of horizontal databases. Columnar databases organize data into fields and store the complete data corresponding with a field in the same storage device. These databases are primarily based on

¹ Row and columnar databases are also referred to as horizontal and vertical databases, respectively.

² ACID stands for Atomicity, Consistency, Isolation, and Duration.

H. N. Dao, P. Ravikumar, P. Likitha and B. V. V. Raj—These authors equally contributed to 98% of this paper.

Y. Watanobe and I. Paik—These authors have equally contributed to 2% of this paper.

© Springer Nature Switzerland AG 2022

H. Fujita et al. (Eds.): IEA/AIE 2022, LNAI 13343, pp. 831–843, 2022.

https://doi.org/10.1007/978-3-031-08530-7_70

BASE³ properties and are designed to be efficient when reading and computing on columns. Snowflake and BigQuery are two examples of columnar databases. Both row and columnar databases have their respective advantages and disadvantages. So, the user and/or application requirements determine the appropriate database layout. Generally, row databases are better suited to online transaction processing (OLTP), whereas columnar databases are better suited to online analytical processing (OLAP). Since the primary objective of OLAP is to uncover meaningful information in data, this paper makes an effort to discover stable periodic-frequent patterns in a columnar database.

Periodic-frequent pattern mining is a useful and essential big data analytical technique. It involves identifying all patterns that satisfy the *minimum support* (*minSup*) and *maximum periodicity* (*maxPer*) constraints which are specified by user. *MinSup* measure constraints the minimum number of transactions in a database where a pattern must appear. *MaxPer* measure constraints the maximum time interval within which a pattern must reappear. Periodic-frequent pattern has been used for the analysis of market-basket analysis, which involves finding the sets of items purchased by the customers periodically. Consider a following example:

$$\{Bread, Butter\} [support = 25\%, periodicity = 2 h].$$

This pattern provides information that 25% of the customers have purchased the items ‘Bread’ and ‘Butter’ at least once every two hours. Such information may be helpful to the managers of a supermarket for inventory management and product placement. Periodic-frequent pattern mining was extended to find fuzzy periodic-frequent patterns [1], partial periodic patterns [2], and high utility periodic patterns [3]. However, this technique has the major disadvantage of being overly strict. The reason is that a pattern is discarded if there exist only one period that exceeds *maxPer*. For example, a pattern that shows a customer purchases bread every day would be discarded if the customer skipped only one day.

To deal with the above problem, some studies was proposed a model to find partial periodic patterns [4] by relaxing the *maxPer* constraint, i.e., some of the periods of a pattern is greater than the user-specified *maxPer* value. Unfortunately, this [4] model is accepting some of the patterns which are having very lengthy periods. For example, purchasing bread can be considered periodic even if a customer purchases it on multiple days but without purchasing it again for a month. The length of periods for some patterns can vary significantly in a real-world database so that traditional models for discovering periodic-frequent patterns are insufficient.

A new class of periodic-frequent patterns named stable periodic-frequent patterns were introduced by Philippe et al. [5], whose recurrence deviation in the database is within the user-specified threshold value. These patterns overcome the above mentioned limitations of the periodic-frequent patterns. Furthermore, a pattern-growth algorithm, called Stable Periodic-Frequent Pattern-growth (SPP-growth), was proposed to discover desired patterns in a temporal database. However, there exist two limitations as follows:

³ BASE stands for Basically Available, Soft state, and Eventually consistent.

- SPP-growth is designed to discover stable periodic-frequent patterns only in row databases. So, desired patterns in columnar databases cannot be found by this algorithm. In addition, this would be costly to transform a big columnar database into a row database.
- In the SPP-growth algorithm, huge memory is required to complete a tree-based recursive mining process, and runtime consumption is also very high.

To that end, a novel algorithm, named Stable Periodic-frequent Pattern-Equivalence CLass Transformation (SPP-ECLAT), is proposed in this paper to discover stable periodic-frequent patterns in a columnar temporal database. The proposed algorithm is shown to be efficient in terms of both memory and runtime.

The remainder of the paper is organized as follows. The related work is presented in Sect. 2. The model of stable periodic-frequent pattern is provided in detail in Sect. 3. The SPP-ECLAT algorithm is then described in Sect. 4. Section 5 shows the evaluation results and discussions. Finally, conclusions and future research directions are given in Sect. 6.

2 Related Work

Tanbeer et al. [6] described a novel pattern-growth algorithm to discover periodic-frequent patterns in a transactional database. Amphawan et al. [7] have identified the most frequent patterns as candidate patterns and generated the Top-k periodic-frequent patterns with the help of the best-first search strategy. Uday et al. [8] have designed a novel concept named *local periodicity* to prune the non-periodic patterns locally. Authors have discarded the patterns whose *local periodicity* is less than the user-specified *maxPer* value. As a result, most of the non-periodic patterns tid-lists were not completely built, resulting in a decrease in the computational time of the proposed algorithm. Anirudh et al. [9] have designed an approach to reduce the memory consumption of the pattern growth approach. In general, PF-trees have maintained the transaction identifiers (tid) in a particular node named as tail-node. However, in real-world applications, it is highly impractical to maintain the complete tid-list. Hence the authors have designed a new strategy named periodic summaries to be maintained at the tail-node to reduce the memory consumption while generating periodic-frequent patterns. Ravi et al. [10] have introduced PF-ECLAT, to find periodic-frequent patterns in a columnar databases. All the algorithms mentioned above will discard a non-periodic pattern if any of the period or local period exceeds the value of the *maxPer* constraint. Kiran et al. [11] have classified patterns as partial periodic and full periodic patterns. In real-world applications, some patterns will occur only at a particular point of time named partial periodic patterns. However, this algorithm cannot be applied to mine stable periodic-frequent patterns. The proposed algorithm has calculated the *period – support* measure as a count of the periods of the patterns whose value is less than the user-specified *maxPer*. Unfortunately, it completely ignores the periods' deviation from the *maxPer*.

A new interestingness measure called lability was exploited by Philippe et al. [5] to determine the interestingness of stable periodic-frequent patterns in

transactional databases. Authors had described a new strategy named lability. The lability of a pattern is the cumulative sum of the difference between each period length and *maxper*. A novel measure *maxLa* was also used to assess the stability of a pattern’s periodic behavior in a database. Ruimeng et al. [12] discussed a model to find stable periodic-frequent patterns in uncertain databases. Fournier-Viger et al. [13] utilized the concept of top-K mining to generate the stable periodic-frequent patterns. It has to be noted that all of these algorithms find the patterns in only row databases. In this paper, we have devised an algorithm to find the patterns in columnar databases.

3 Model of Stable Periodic-Frequent Patterns

Assume that we have a **pattern** (or an itemset) Y , $Y \subseteq I$, where I is the set of items. Denoted k -**pattern** as the pattern that has k items, $k \geq 1$. $t_k = (ts, X)$ is a **transaction** with X being the pattern and ts being timestamp, $ts \in \mathbb{R}^+$. A set of transactions constitute a **temporal database** denoted by TDB over I , i.e., $TDB = \{t_1, \dots, t_d\}$, $d = |TDB|$. Let ts^Y be the timestamp of pattern Y , $Y \subseteq X$, which occurs in transaction t_i (or t_i contains Y), $t_i = (ts, X)$, $i \geq 1$. Denoted TS^Y as a set of timestamps $\{ts_j^Y, \dots, ts_i^Y\}$, $j, k \in [1, d]$ and $j \leq i$. TS^Y can be consider as an **ordered set of timestamps** of pattern Y .

Example 1. Assume that we have a set of items $I = \{p, q, r, s, t, u\}$. Table 1 shows a row temporal database. Table 2 shows a columnar temporal database which is converted from above row database. In Table 3 we show for each item the temporal occurrences over the whole database. The set of items ‘ r ’ and ‘ q ’, i.e., $\{r, q\}$ is a pattern. This pattern will be represented as ‘ rq ’ for brevity. This pattern is denoted as 2-pattern because it contains two items. The occurrences of pattern ‘ rq ’ are at the timestamps of 1, 3, 6, 8, 9, and 10. Therefore, we have a list of timestamps containing ‘ rq ’, i.e., $TS^{rq} = \{1, 3, 6, 8, 9, 10\}$.

Definition 1 (The support of Y). Denoted $sup(Y)$ the **support** of Y which is the number of transactions containing Y in TDB . That is, $sup(Y) = |TS^Y|$.

Example 2. The support of ‘ rq ’, i.e., $sup(rq) = |TS^{rq}| = 6$.

Definition 2 (Frequent pattern Y). The pattern Y is a **frequent pattern** if $sup(Y) \geq minSup$, where $minSup$ is a minimum support value indicated by user.

Example 3. Suppose $minSup = 5$, then rq is a frequent pattern because of $sup(rq) \geq minSup$.

Definition 3. (Periodicity of Y). Denoted ts_m^Y and ts_n^X , $j \leq m < n \leq k$ the two consecutive timestamps in TS^Y . The time difference between ts_n^Y and ts_m^Y is given by a **period** of Y , denoted by p_z^Y . That is, $p_z^Y = ts_n^Y - ts_m^Y$. Denoted $P^Y = (p_1^Y, p_2^Y, \dots, p_n^Y)$ the set of all periods for pattern Y . The **periodicity** of Y , denoted by $per(Y) = maximum(p_1^Y, p_2^Y, \dots, p_n^Y)$.

Table 1. Row database

ts	Items	ts	Items
1	<i>grs</i>	6	<i>pqr</i>
2	<i>pq</i>	7	<i>stu</i>
3	<i>pqrs</i>	8	<i>grs</i>
4	<i>tu</i>	9	<i>pqrs</i>
5	<i>prs</i>	10	<i>pqrs</i>

Table 2. Columnar database

Items						Items							
<i>ts</i>	<i>p</i>	<i>q</i>	<i>c</i>	<i>s</i>	<i>e</i>	<i>u</i>	<i>ts</i>	<i>p</i>	<i>q</i>	<i>r</i>	<i>s</i>	<i>t</i>	<i>u</i>
1	0	1	1	1	0	0	6	1	1	1	0	0	0
2	1	1	0	0	0	0	7	0	0	0	1	1	1
3	1	1	1	1	0	0	8	0	1	1	1	0	0
4	0	0	0	0	1	1	9	1	1	1	1	0	0
5	1	0	1	1	0	0	10	1	1	1	1	0	0

Table 3. Timestamp list of an item

Item	TS-list
<i>p</i>	2,3,5,6,9,10
<i>q</i>	1,2,3,6,8,9,10
<i>r</i>	1,3,5,6,8,9,10
<i>s</i>	1,3,5,7,8,9,10
<i>t</i>	4,7
<i>u</i>	4,7

Example 4. All periods of the pattern ‘*rq*’ are : $p_1^{rq} = 1$ ($= 1 - ts_{initial}$), $p_2^{rq} = 2$ ($= 3 - 1$), $p_3^{rq} = 3$ ($= 6 - 3$), $p_4^{rq} = 2$ ($= 8 - 6$), $p_5^{rq} = 1$ ($= 9 - 8$), $p_6^{rq} = 1$ ($= 10 - 9$), and $p_8^{rq} = 0$ ($= ts_{final} - 10$), where first transaction time stamp is denoted by $ts_{initial} = 0$ and the last transaction’s time stamp is denoted by, $ts_{final} = |TDB| = 10$. The *periodicity* of *rq*, i.e., $per(rq) = maximum(1, 2, 3, 2, 1, 1, 1, 0) = 3$.

Definition 4 (Periodic-frequent pattern *Y*). The frequent pattern *Y* be considered as **periodic-frequent pattern** if $per(Y) \leq maxPer$, here *maxPer* is maximum periodicity value which is specified by user.

Example 5. Let the user-specified $maxPer = 3$, in this case the frequent pattern ‘*rq*’ is called as a periodic-frequent pattern as $per(rq) \leq maxPer$.

Definition 5 (Lability of an itemset). Denoted ts_{i+1}^Y and ts_i^Y , $i \in [0, sup(Y)]$ two consecutive time stamps where *Y* occurs in *TDB*. We call *i*-th lability of *Y* denoted by $la(Y, i) = max(0, la(Y, i - 1) + p_i^Y - maxPer)$, where $la(Y, -1) = 0$. For simplicity, the following short form is used

$$la(Y, i) = max(0, la(Y, i - 1) + ts_{i+1}^Y - ts_i^Y - maxPer)$$

The following is a list of periods which represent the lability of an itemset *Y*: $la(Y) = \{la(Y, 0), la(Y, 1), \dots, la(Y, sup(Y))\}$, and $|la(Y)| = |per(Y)| = sup(Y) + 1$.

Example 6. Given an item *p*. If $maxPer = 2$, the parameters for calculating its lability are $la(p, 0) = max(0, la(p, -1) + p_0^p - maxPer) = max(0, 0 + 2 - 2) = 0$, $la(p, 1) = 0$, $la(p, 2) = 0$, $la(p, 3) = 0$, $la(p, 4) = 1$, $la(p, 5) = 0$, and $la(p, 6) = 0$. Therefore, the lability of *p* is $la(p) = \{0, 0, 0, 0, 1, 0, 0\}$.

Based on Definition 5, the periodic pattern can be considered as stable (lability is zero) if all its periods are less than or equal to *maxPer*. The lability of a period of a pattern will increase when a period of a pattern larger than *maxPer*, and these exceeding values are accumulated using the measure of lability. The value of lability will be reduced when periods of a pattern no more than *maxPer*. Therefore, according to the periodic characteristic of a pattern, its *lability* will

vary over time, and each value exceeding $maxPer$ is accumulated. A periodic behavior is considered stable when lability value is low while a high value means an unstable one. So stable pattern can be found using this measure given a limit on the maximum lability.

Definition 6 (Stable periodic-frequent pattern). For a pattern Y , denote $la(Y)$ the set of all i -th lability. The stability of the pattern is defined by $maxla(Y) = \max(la(Y))$. Pattern Y is a SPP if $sup(Y) \geq minSup$ and $maxla(Y) \leq maxLa$.

Example 7. Given the above example, if the user specified $minSup=4$, $maxPer=2$, and $maxLa=1$, the complete set of SPPs are $p: (6,1)$, $ps: (4,0)$, $psr: (4,0)$, $pq: (5,0)$, $pqr: (4,0)$, $pr: (5,0)$, $q: (7,1)$, $qs: (5,0)$, $qsr: (5,0)$, $qr: (6,0)$, $r: (7,0)$, $rs:(6,0)$ and $s: (7,0)$, where each SPP Y is annotated with $Y: (sup(Y), maxLa(Y))$.

Be noted that if $maxLa = 0$, SPPs are the traditional PFPs. Therefore, the PFPs is a special case of SPPs.

Definition 7 (Problem definition). Considering a temporal database (TDB) with minimum support ($minSup$), maximum periodicity ($maxPer$), and maximum lability ($maxLa$) constraints. The purpose of this task is discovering the complete set of stable periodic-frequent patterns that have support higher or equal to $minSup$ and lability lower or equal to $maxLa$ constraints.

4 Our Mining Algorithm: SPP-ECLAT

This section shows the process of mining the stable periodic-frequent patterns in two steps using SPP-ECLAT algorithm and our algorithm works in two steps. First, SPP-ECLAT algorithm utilizes the Depth-First Search (DFS) strategy on the itemset lattice. Second, this algorithm employs the *downward closure property* (see Property 1) of stable periodic-frequent patterns to minimize the huge search space of the lattice effectively.

Property 1. If A is a stable periodic-frequent pattern, then $\forall A \subset B$ and $A \neq \emptyset$, A is also a stable periodic-frequent pattern.

4.1 Mining 1-Stable Periodic-Frequent Patterns

This part focuses on discovering 1-patterns by SPP-list. The detailed steps are shown in Algorithm 1, which works on a row database shown in Table 1. Let $minSup = 5$ and $maxPer = 2$ and $maxLa = 1$.

The 1-patterns are first generated by reading the whole database transactions at once. Then, the row database is converted to the columnar database. After reading the 1st transaction, “1 : qrs ”, with $ts_{cur} = 1$ inserts the items q , r and s , in the SPP-list. We have the timestamps of these items is 1 ($= ts_{cur}$). Similarly, ML and TS_i contents were updated to 0 and 1, respectively (lines 7 and 8 in

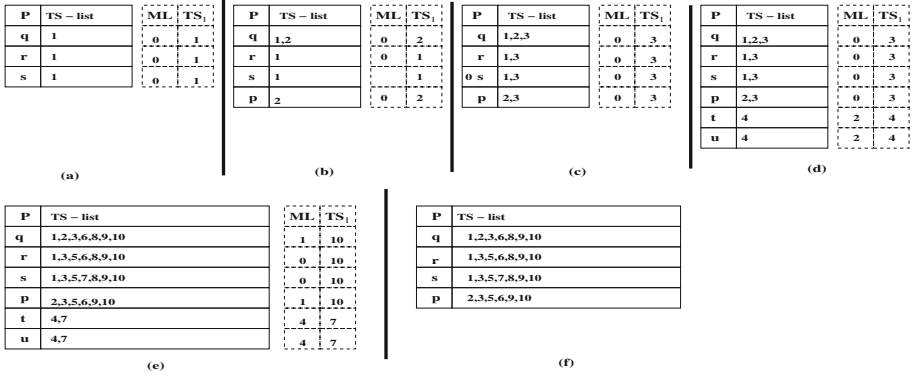


Fig. 1. SPP-list generation process. (a) content of the list after reading the 1st transaction, (b) after reading the 2nd one, (c) after reading the 3rd one, (d) after reading the 4th one, (e) Final content after reading the whole database, and (f) The complete list of 1-stable periodic-frequent patterns

Algorithm 1). Figure 1(a) shows the generated SPP-list from the 1st transaction. After reading the 2nd one, “2 : pq”, with $ts_{cur} = 2$ inserts the new items p into the SPP-list by adding 2 ($= ts_{cur}$) in their TS-list. At the same instant, the ML and TS_l contents were updated to 0 and 2, respectively. Besides 2 ($= ts_{cur}$) was added to the TS-list of existing items q with ML and TS_l contents were updated to 0 and 2, respectively (lines 10 and 13 in Algorithm 1). The SPP-list which is generated after reading the 2nd one is shown in Fig. 1(b). After reading the 3rd one, “3 : pgrs”, updates the TS-list, ML and TS_l values of p , q , r , and s in the SPP-list. Figure 1(c) shows the SPP-list which is generated after reading the 3rd one. After reading the 4th one, “4 : tu” with $ts_{cur} = 4$, inserts the new items e and u into the SPP-list by adding 4 ($= ts_{cur}$) in their TS-list. Simultaneously, the ML and TS_l values as 2 and 4. Figure 1(d) shows the SPP-list which is generated after reading the 4th. We repeat the whole process for the remaining transactions. Figure 1(e) depicts the final SPP-list which is generated after scanning the whole database. The pattern t and u are pruned (using the Property 1) from the SPP-list as its *support* value is no more than the *minSup* value and ML value is greater than *maxLa* (lines 15 to 20 in Algorithm 1). The complete list of patterns available in the SPP-list are considered as 1-stable periodic-frequent patterns. Those patterns are sorted in descending order in terms of their *support* values. Figure 1(f) shows the final SPP-list.

4.2 Finding All Interesting Patterns from SPP-ECLAT

The detailed procedure for finding stable periodic-frequent patterns is shown in Algorithm 2. Given the newly generated SPP-list, the procedure of this algorithm is carried out as follows. Initially we choose the pattern q , as this is the initial pattern in the SPP-list (line 2 in Algorithm 2). Figure 2(a) shows a record of its *support* and *lability*. Since q is a stable periodic-frequent pattern, we move

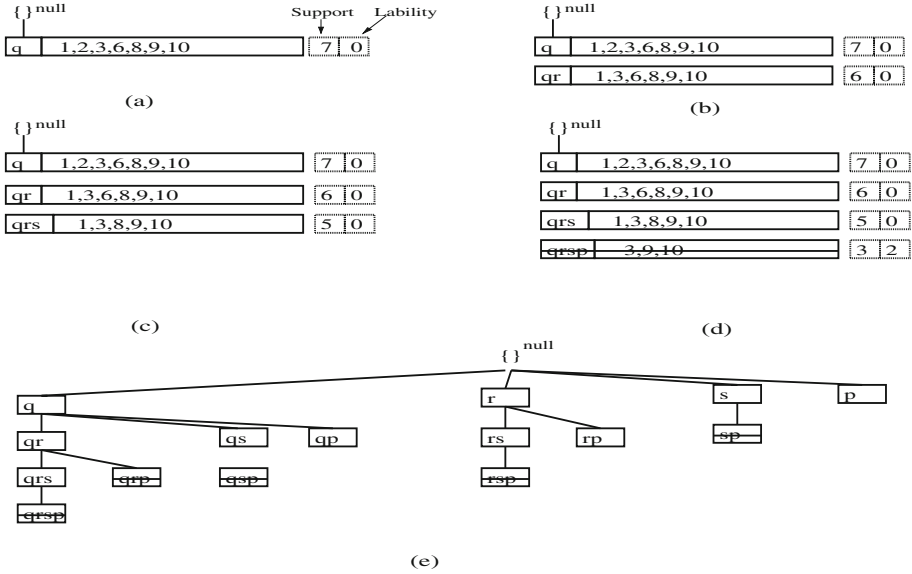


Fig. 2. The complete process of discovering stable periodic-frequent patterns using SPP-ECLAT algorithm

to its child node qr . TS-list of qr is generated by performing intersection of TS-lists of q and r , i.e., $TS^{qr} = TS^q \cap TS^r$ (lines 2 and 3 in Algorithm 2). This *support* and *lability* of qr are recorded, as shown in Fig. 2(b). We check whether qr is a stable periodic-frequent pattern or unstable periodic frequent pattern (line 4 in Algorithm 2). Since qr is stable periodic-frequent pattern we move it to its child node qrs . Next, TS-list will be generated by performing the intersection of TS-lists of qr and s , i.e., $TS^{qrs} = TS^{qr} \cap TS^s$. Figure 2(c) shows a record of *support* and *lability* of qrs . Then qrs is identified as a stable periodic-frequent pattern. Later, we shift to its child node $qrsp$. We produce its TS-list by performing intersection of TS-lists of qrs and p , i.e., $TS^{qrsp} = TS^{qrs} \cap TS^p$. Because a *support* of $qrsp$ is less than $minSup$ and *lability* is greater than $maxLa$, the pattern $qrsp$ will be remove from the stable periodic-frequent patterns list as shown in Fig. 2(d). We repeat the process to find all stable periodic-frequent patterns for remaining nodes in the tree. Figure 2(e) shows the final list of generated stable periodic-frequent patterns. Since we can reduces the search space and the computational cost effectively our proposed approach is efficient.

5 Experimental Results

This section evaluates the performance of the SPP-ECLAT against the state-of-the-art algorithm named SPP-growth [5]. It shows that the SPP-ECLAT algorithm is more efficient in memory consumption and runtime than SPP-growth.

Algorithm 1. StablePeriodicFrequentItems(Temporal database (TDB), minimum support ($minSup$), maximum periodicity ($maxPer$), maximum Liability ($maxLa$):

```

1: Definition:  $SPP-list = (Y, TS-list(Y))$  is a dictionary with the temporal occurrence information
of a pattern in a  $TDB$ ;  $TS_i$  is a temporary variable of list type to store the timestamp of the
final occurrence of a pattern;  $la$  and  $ML$  are temporary variable of list type to store the liability
and the Maximum Liability of a pattern;  $last$  is a term for the final timestamp;  $support$  is a
temporary variable of list type to store the support of a pattern.
2: Initiate  $ts_{cur} = 0$ 
3: for each transaction  $t_{cur} \in TDB$  do
4:   Set  $ts_{cur} = t_{cur}.ts$ ;
5:   for each item  $j \in t_{cur}.Y$  do
6:     if  $j$  does not exist in SPP-list then
7:       SPP-list is updated by inserting  $j$  and corresponding timestamp value
8:        $la[j] = \max(0, ts_{cur} - maxPer)$ . Set  $ML[j] = la[j]$ 
9:     else
10:      Add  $j$ 's timestamp in the SPP-list.
11:       $la[j] = \max(0, la[j] + ts_{cur} - TS_i[j] - maxPer)$ 
12:       $ML[j] = \max(la[j], ML[j])$ 
13:      Update  $TS_i[j] = ts_{cur}$ .
14:    $last = ts_{cur}$ 
15: for each item  $j$  in SPP-list do
16:    $la[j] = \max(0, la[j] + last - TS_i[j] - maxPer)$ 
17:    $ML[j] = \max(la[j], ML[j])$ 
18:    $s[j] = \text{length}(TS-list[j])$ 
19:   if  $s[j] < minSup$  and  $ML[j] > maxLa$  then
20:     Prune  $j$  from SPP-list
21: After the pruning the final list of patterns available in the SPP-list is sorted in ascending
order or descending order of the corresponding pattern's support. Initiate  $\pi_i$  as Null. Call SPP-
ECLAT(SPP-List,  $\pi_i$ ).

```

Algorithm 2. SPP-ECLAT(SPP-List, π_i)

```

1: for each item  $j$  in SPP-List do
2:   Set  $Y = j \cup \pi_i$  and  $TS^Y = TS^j \cap TS^{\pi_i}$ ;
3:   Calculate support and Liability of  $X$ ;
4:   if  $sup(TS^Y) \geq minSup$  and  $la(TS^Y) \leq maxLa$  then
5:     Add  $j$  to  $\pi_i$  and  $Y$  is considered as stable periodic-frequent pattern;
6:     SPP-ECLAT(SPP-list[j+1:],  $\pi_i$ );

```

The algorithms, SPP-growth and SPP-ECLAT, were developed in Python 3.7 and executed on a machine containing two AMD EPIC 7542 cpus and 600 GB RAM. The operating system of this machine is Ubuntu Server OS 20.04. The experiments have been conducted on real-world (T10I4D100K, Retail, and Mushroom) databases. The complete statistics of the databases is shown in the Table 4.

In this experiment, we have fixed the values of $minSup$, $maxPer$ for all the three databases. Subsequently, we have evaluated the performance of both the algorithms by varying the $maxLa$ parameter for all the three databases. Figure 3(a)–3(c) shows the number of stable periodic-frequent patterns generated in T10I4D100K, Retail, and Mushroom databases at different $maxLa$ values. After careful observation of the mentioned graphs, we can conclude that raises in $maxLa$ positively affect the total count of the number of stable periodic-frequent patterns. With an increase in the $maxLa$ threshold, most of the patterns have become stable periodic-frequent patterns.

Table 4. Statistics of the databases used

S. No	Database	Type	Nature	Transaction Length (in count)			Database Size (in count)
				min.	avg.	max.	
1	T10I4D100K	Synthetic	Sparse	1	10	29	1,00,000
2	Retail	Real	Sparse	2	12	77	88,162
3	Mushroom	Real	Dense	23	23	23	8,124

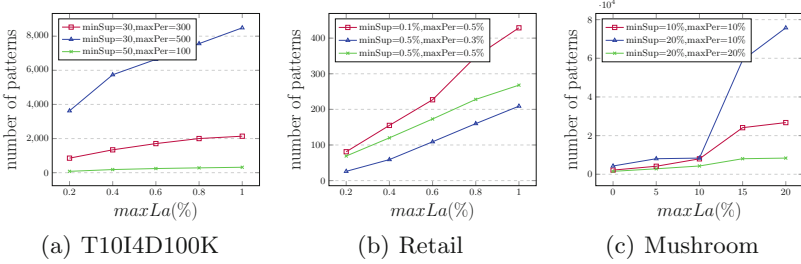


Fig. 3. Number of stable periodic-frequent patterns generated in various datasets

We have varied the values of $minSup$, $maxPer$, and $maxLa$ parameters and shown the runtime requirements of SPP-growth and SPP-ECLAT algorithms in Fig. 4. Specifically, for Retail database, the runtime requirement of both algorithms are shown in Fig. 4(a)–4(c), respectively. For T10I4D100K database, the runtime requirement of both algorithms are shown in Fig. 4(d)–4(f), respectively. For Mushroom database, the runtime requirement of both algorithms are shown in Fig. 4(g)–4(i), respectively. After careful observation of the mentioned graphs, we can conclude that raises in the value of the $maxLa$ parameter shows the raising trend in the graphs. However, we can conclude that SPP-ECLAT algorithms always consumes relatively less runtime than the SPP-growth algorithm.

We have varied the values of $minSup$, $maxPer$, and $maxLa$ parameters and shown the memory consumption of SPP-growth and SPP-ECLAT algorithms in Fig. 5. Specifically, for Retail database, the memory consumption of both algorithms are shown in Fig. 5(a)–5(c), respectively. For T10I4D100K database, the runtime requirement of both algorithms are shown in Fig. 5(d)–Fig. 5(f), respectively. For Mushroom database, the runtime requirement of both algorithms are

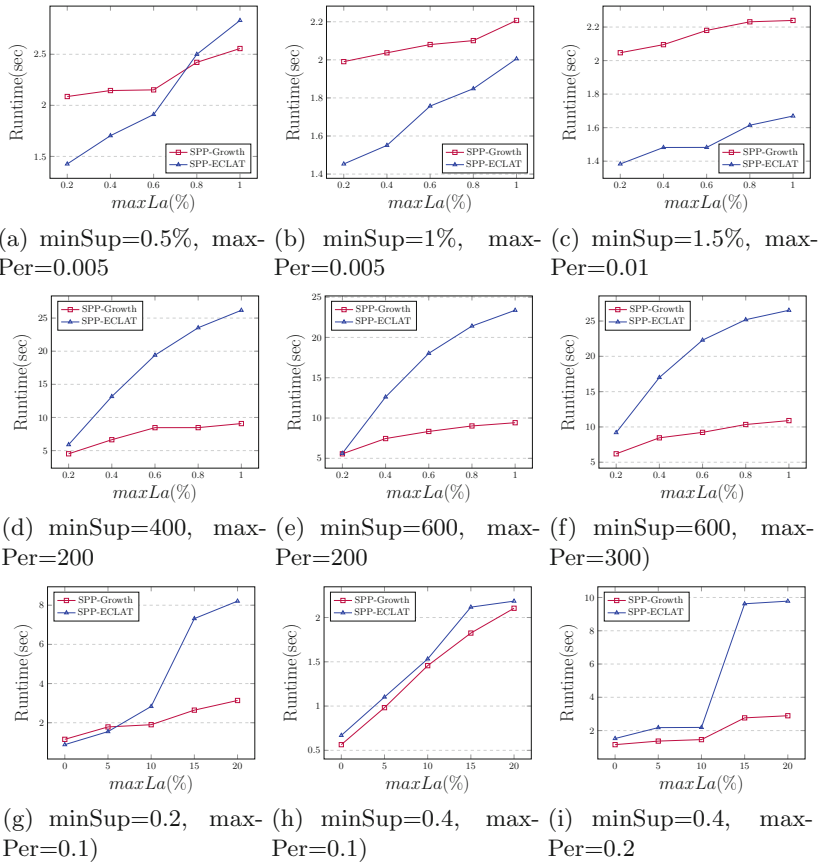


Fig. 4. Runtime comparison of the two algorithms

shown in Fig. 5(g)–5(i), respectively. After careful observation of the mentioned graphs, we can conclude that raises in the value of the $maxLa$ parameter shows the raising trend in the graphs. However, we can conclude that SPP-ECLAT algorithms always consumes relatively less memory than the SPP-growth algorithm.

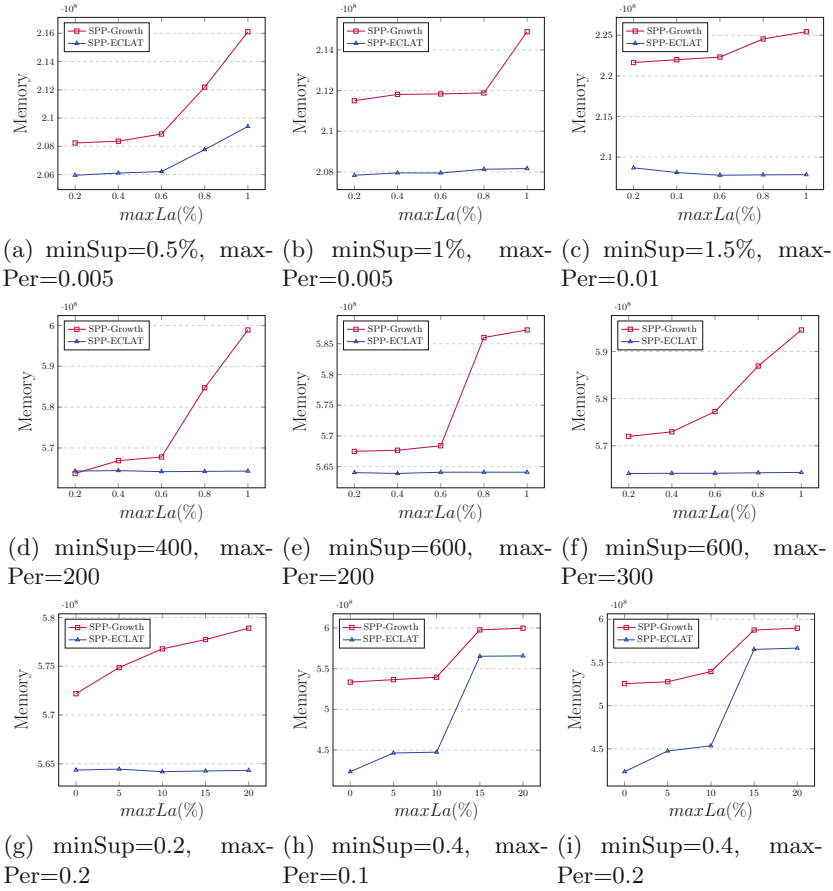


Fig. 5. Memory comparison of the two algorithms

6 Conclusions and Future Work

This paper proposes an efficient algorithm called stable periodic-frequent pattern-equivalence class transformation to discover stable periodic-frequent patterns from columnar temporal databases. The experiment was carryout with different real-world databases. Experimental results show that the proposed algorithm consumes less memory and can generate interesting patterns much faster than the state-of-the-art algorithm. We want to work on parallel algorithms to discover stable periodic-frequent patterns in vast temporal databases as part of future work.

References

1. Kiran, R.U., et al.: Discovering fuzzy periodic-frequent patterns in quantitative temporal databases. In: 2020 (FUZZ-IEEE), pp. 1–8 (2020)
2. Han, J., Dong, G., Yin, Y.: Efficient mining of partial periodic patterns in time series database. In: ICDE, pp. 106–115 (1999)
3. Fournier-Viger, P., Lin, J.C., Duong, Q., Dam, T.: PHM: mining periodic high-utility itemsets. In: ICDM, pp. 64–79 (2016)
4. Kiran, R.U., Venkatesh, J.N., Fournier-Viger, P., Toyoda, M., Reddy, P.K., Kitsuregawa, M.: Discovering periodic patterns in non-uniform temporal databases. In: PAKDD (2017)
5. Fournier-Viger, P., Yang, P., Lin, J.C., Kiran, R.U.: Discovering stable periodic-frequent patterns in transactional data. In: IEA/AIE, pp. 230–244 (2019)
6. Tanbeer, S.K., Ahmed, C.F., Jeong, B.-S., Lee, Y.-K.: Discovering periodic-frequent patterns in transactional databases. In: PAKDD, pp. 242–253 (2009)
7. Amphawan, K., Lenca, P., Surarerks, A.: Mining top-k periodic-frequent pattern from transactional databases without support threshold. In: Advances in Information Technology, pp. 18–29 (2009)
8. Kiran, R.U., Kitsuregawa, M.: Novel techniques to reduce search space in periodic-frequent pattern mining. In: DASFAA, pp. 377–391 (2014)
9. Anirudh, A., Kiran, R.U., Reddy, P.K., Kitsuregawa, M.: Memory efficient mining of periodic-frequent patterns in transactional databases. In: IEEE Symposium Series on Computational Intelligence 2016, pp. 1–8 (2016)
10. Penugonda, R., Palla, L., Rage, U.K., Watanobe, Y., Zettsu, K.: Towards efficient discovery of periodic-frequent patterns in columnar temporal databases. In: IEA/AIE. Springer International Publishing 2021, pp. 28–40 (2021)
11. Kiran, R.U., Shang, H., Toyoda, M., Kitsuregawa, M.: Discovering partial periodic itemsets in temporal databases. In: SSDBM, pp. 30:1–30:6 (2017)
12. He, R., Chen, J., Du, C., Duan, Y.: Stable periodic frequent itemset mining on uncertain datasets. In: IEEE CCET 2021, pp. 263–267 (2021)
13. Fournier Viger, P., Wang, Y., Yang, P., Lin, C.-W., Yun, U., Rage, U.: Tspin: mining top-k stable periodic patterns. Applied Intelligence, 02 2021