IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 31 December 2022, accepted 27 January 2023, date of publication 1 February 2023, date of current version 9 February 2023.

*Digital Object Identifier 10.1109/ACCESS.2023.3241313*

RESEARCH ARTICLE

# Finding Stable Periodic-Frequent Itemsets in Big Columnar Databases

**HONG N. DAO[1], PENUGONDA RAVIKUMAR[1,2], PALLA LIKHITHA[1], RAGE UDAY KIRAN[1,3,4], (Member, IEEE), YUTAKA WATANOBE[1], (Member, IEEE), AND INCHEON PAIK[1], (Senior Member, IEEE)**

[1]Department of Computer Science and Engineering, The University of Aizu, Aizuwakamatsu, Fukushima 965-8580, Japan
[2]IIIT-Idupulapaya, RGUKT, Nuzividu, Andhra Pradesh 521202, India
[3]National Institute of Information and Communications Technology, Tokyo 184-0015, Japan
[4]Institute of Industrial Science, The University of Tokyo, Tokyo 113-8654, Japan

Corresponding author: Rage Uday Kiran (udayrage@u-aizu.ac.jp)

**ABSTRACT** Stable periodic-frequent itemset mining is essential in big data analytics with many real-world applications. It involves extracting all itemsets exhibiting stable periodic behaviors in a temporal database. Most previous studies focused on finding these itemsets in row (temporal) databases and disregarded the occurrences of these itemsets in columnar databases. Furthermore, the naïve approach of transforming a columnar database into a row database and then applying the existing algorithms to find interesting itemsets is not practicable due to computational reasons. With this motivation, this paper proposes a framework to discover stable periodic-frequent itemsets in columnar databases. Our framework employs a novel depth-first search algorithm that compresses a given columnar database into a unified dictionary and mines it recursively to find all stable periodic-frequent itemsets. The dictionary holds the information pertaining to itemsets and their temporal occurrences in a database. Experimental results on six databases demonstrate that the proposed algorithm is computationally efficient and scalable.

**INDEX TERMS** Columnar databases, stable periodic-frequent itemset, itemset mining.

## I. INTRODUCTION

Database systems play a crucial role in storing the big data generated by real-world applications. Depending on the layout used for storing the data, one can broadly classify the databases into two types: *row databases* and *columnar databases*.[1] Row databases are primarily based on ACID[2] properties and organize the data as records by keeping the data associated with a record next to each other in a storage device. The popular row databases include MySQL [1] and Postgres [2]. In contrast, columnar databases are based on BASE [3] properties and organize data into fields and store all of the data associated with a field next to each other in a storage device. The popular columnar databases include BigQuery [3], HBase [4], and Snowflake [5]. Both row and columnar databases have their respective advantages and disadvantages. Henceforth, no universally accepted best data layout exists for any given application. Selecting a right database layout is subjective to user and/or application requirements.

Extracting meaningful information from the data is a crucial task of data mining. Frequent Itemset Mining (FIM) [6], [7], [8], [9], [10], [11], [12] is a renowned data mining technique that aims to discover all frequently occurring itemsets in the data. Numerous algorithms have been presented in the literature to discover frequent itemsets effectively. One can broadly classify these approaches into the following three types: (*i*) candidate-generate-and-test algorithms (e.g., Apriori [8] and Partitioning [13]), (*ii*) pattern-growth algorithms (e.g., FP-Growth [14], HMine [15], and [16]),

The associate editor coordinating the review of this manuscript and approving it for publication was Laura Celentano.

[1]Columnar and row databases are referred as vertical and horizontal databases, respectively.
[2]ACID is an acronym for Atomicity, Consistency, Isolation, and Duration.
[3]BASE is an acronym for Basically Available, Soft state, and Eventually consistent.

and (*iii*) vertical format algorithms (e.g. ECLAT [17] and CHARM [18]). The candidate-generate-and-test and pattern-growth approaches can find frequent itemsets only in row databases, while vertical format approaches can find frequent itemsets in both row and columnar databases. Henceforth, researchers are putting forth efforts to develop efficient vertical format algorithms. This paper also develops an efficient vertical format algorithm to discover a class of frequent itemsets, called stable periodic-frequent itemsets, in columnar temporal databases.

A temporal database represents a temporally ordered set of transactions. Crucial information that can empower the domain experts to gain competitive advantage lies hidden in this data. Tanbeer et al. [19] described the model of periodic-frequent itemset to discover regularities in a temporal database. This model involves discovering all itemsets in a database that satisfy the user-specified *minimum support* (*minSup*) and *maximum periodicity* (*maxPer*) constraints. The *minSup* controls the minimum number of transactions in which a itemset must appear in the database. The *maxPer* controls the maximum time interval within which an itemset must reappear. A classic application of periodic-frequent itemset mining is market-basket analytics. It involves identifying the itemsets that the customers regularly purchase in a supermarket. An example of a periodic-frequent itemset is as follows:

$$\{Cheese, Wine\}\ [support = 20\%, periodicity = 5\ hour].$$

The above itemset provides information that 20% of the customers have purchased the products 'Cheese' and 'Wine' at least once every 5 hours. Therefore, supermarket managers may find this information beneficial for inventory management and product placement.

In the literature, the periodic-frequent itemset model was extended to discover fuzzy periodic-frequent itemset [20], rare periodic-frequent itemset [21], partial periodic itemset [22], [23], and high utility periodic-frequent itemset [24]. However, the successful real-world adoption of this model has been affected by the following obstacle: "*Since maxPer controls the maximum inter-arrival time of an itemset in a database, the basic model of periodic-frequent itemset considers any itemset uninteresting if anyone of its inter-arrival time is more than the user-specified maxPer value [19], [25]. In other words, the strict restriction that **all periods** of a itemset must be within the user-specified maxPer constraint often prunes all of those interesting itemsets that have exhibited stable (or partial) periodic behavior in a database.*"

When confronted with this problem in real-world applications, researchers introduced the model of stable periodic-frequent itemset [26] to find all of those interesting itemset that have exhibited stable periodic behavior in columnar database. This model provides a function to find interesting itemsets that have a stable periodic behavior. A pattern-growth algorithm, called Stable Periodic-frequent Pattern-growth (SPP-growth), was described to find stable-periodic itemsets in temporal databases. Unfortunately, this algorithm can discover the interesting itemsets in row (temporal) databases only. Therefore, whenever we give a columnar temporal database as an input to the SPP-growth algorithm, it has to be converted into a row temporal database to get interesting itemsets. As a result, the above algorithms will take longer to run and use more memory because of this conversion overhead. With this motivation, this paper proposes a generic algorithm to find stable periodic-frequent itemset in both row and columnar temporal databases effectively. To the best of our knowledge, this is the first algorithm that focuses on finding stable periodic-frequent itemset in columnar temporal database. It should be noted that existing algorithms find the itemsets only in row databases.

Discovering stable periodic-frequent itemset in columnar databases is significant and challenging because of some reasons as follows:

1) The importance of discovering frequent itemset in columnar databases was first discussed in the work of Zaki [27], where the depth-first-search algorithm, named Equivalence Class Transformation (ECLAT), was proposed to extract frequent itemset in a columnar database. However, the ECLAT algorithm cannot be directly applied to find stable periodic-frequent itemset in a columnar temporal database. The reason is ECLAT algorithm completely disregards the temporal occurrence information of an itemset in the data.
2) Reducing search space (itemset lattice) is a challenging task in itemset mining. The process of recursively mining the constructed tree increases the memory and runtime requirements of the SPP-growth algorithm.
3) One can transform a columnar temporal database into a row database and then apply those available algorithms to extract stable periodic-frequent itemset. However, we should avoid such a transformation process due to its high computational cost.

Against this backdrop, we have extended the functionality of ECLAT [27] to mine stable periodic-frequent itemsets by introducing a new algorithm called Stable Periodic-frequent Pattern – Equivalence Class Transformation (SPP-ECLAT) in columnar temporal database. This paper is a substantially extended version of our conference paper [28] which reported a preliminary version of SPP-ECLAT. This paper extends the related work by extensively understanding the current literature, presenting the complexity analysis of our algorithm, and performing in-depth experiments studying the memory, runtime, and scalability of the mining algorithms. In this paper, we show that SPP-ECLAT outperforms SPP-growth [26] on both synthetic and real-world databases by a very large margin.

The key contributions of this paper are summarized as follows:

1) An efficient and novel SPP-ECLAT algorithm is proposed to ensure that the discovered stable periodic-frequent itemsets (SPIs) not only satisfy

the user-specified *minimum support* and *maximum periodicity* thresholds but are stable itemsets based on the user-specified *maximum lability* threshold in any big columnar temporal databases.

2) In SPP-ECLAT, the observed *Lability* information is stored in a unique, compact list-based data structure called SPP-List. The newly introduced *maximum lability* measure considers the periodic behavior of an itemset as stable when the lability value is low. On the other hand, if the value is high, it means the itemsets are unstable. So stable itemset can be found using this measure, given a limit on the maximum lability.
3) On six synthetic and real-world databases, we compare the performance of the proposed SPP-ECLAT algorithm against that of the current state-of-the-art SPP-Growth algorithm. This indicates that the SPP-ECLAT algorithm outperforms the SPP-Growth algorithm with respect to runtime requirements and memory consumption. Furthermore, the scalability of the SPP-ECLAT algorithm is also shown to demonstrate the efficacy and productivity of the proposed algorithm on big columnar databases relative to those of the state-of-the-art SPP-Growth algorithm.

The rest of the paper is organized as follows. Related work is presented in Section II. The model of a stable periodic-frequent itemset is explained in Section III. The SPP-ECLAT algorithm is then presented in Section IV. Section V describes the experimental results. The discussion is given in Section VI. Finally, the conclusion and future work is given in Section VII.

## II. RELATED WORK

In this section, we will review the previous work related to frequent itemset mining, periodic-frequent itemset mining, and stable periodic-frequent itemset mining.

### A. FREQUENT ITEMSET MINING

Argawal et al. [6] introduced frequent itemset mining to find interesting relationships among different data items. An algorithm, called Apriori [6], was also introduced to discover all frequent itemsets in a row (transactional) database. This algorithm works in a breadth-first manner that uses frequent $k$-itemsets to form candidate $(k + 1)$-itemsets, from which frequent $(k + 1)$-itemsets are obtained. Many extensions of Apriori have been proposed in the literature [13], [29]. Essentially, they have the same general structure, with some additional techniques to optimize certain steps within the algorithm. Though Apriori can find all wanted frequent itemset, it has to scan the database several times to generate a complete set of itemset. Thus, it is a very time-consuming process. Beside Apriori algorithm, Argawal et al. [30] proposed two other algorithms called AprioriTid and AprioriHybrid. The AprioriTid algorithm reduces the processing time of the support counting procedure by replacing every transaction in the database with a set of candidate itemsets that appears in that transaction. This is done repeatedly at every iteration k. It is demonstrated in [8] that although AprioriTid is much faster in the later iterations, it performs slower than Apriori in early iterations. Therefore, the AprioriHybrid algorithm has been proposed [30], which combines Apriori and AprioriTid. Basically, the hybrid algorithm uses Apriori for the initial iterations and then switches to AprioriTid. Even though the AprioriTID algorithm have utilized a vertical database representation, this algorithm is based on the breadth-first search technique.

The first algorithm to generate all frequent itemsets in a depth-first search manner, called Eclat, is proposed by Zaki [31]. Eclat is a vertical database layout algorithm. This algorithm utilizes the TID-list data structure for the mining task. Eclat applies the depth-first approach to find frequent itemset and scan the database only two times. In the first round, it scans the entire database to find all frequent items. In the second round, the TID-list of the frequent items is generated. The Eclat algorithm uses common $(k-1)$-prefixes to organize frequent k-itemsets into disjoint equivalence classes. Then the candidate $(k + 1)$ itemsets can be found by joining two frequent $k$-itemsets from the same classes. The main advantage of utilizing TID-list is that, only by intersecting the TID-lists of the two subsets, the support of a candidate itemset is simply computed. A simple check on the received TID-list can tell whether the new itemset is frequent.

The frequent pattern-growth (FP-growth) algorithm proposed by Han et al. [9] is a tree-based algorithm to discover frequent itemset in a database. This algorithm uses the divide-and-conquer method. In this algorithm, frequent itemset are mined from the fp-tree, and there is no need for a candidate frequent itemset. In the first step, a list of frequent itemsets is generated and sorted in their descending support order. This list is represented as a node structure, containing the item name, support count, and a pointer to a node in the tree that has the same prefix. These nodes then are used to create an fp-tree. The paths from the root to leaf nodes are arranged in the decreasing order of their support. Frequent itemset are extracted from the fp-tree starting from the leaf nodes. To mine frequent itemset(s) each prefix path subtree is processed recursively. The only differences between Eclat and FP-growth are the process to count the support of every candidate itemset and how they represent the database. In fact, it is difficult to say which algorithm performs better. Over two decades, many other FIM algorithms have been proposed, mainly by extending the Apriori, Eclat, and FP-growth algorithms to find a frequent itemset. However, frequent itemset mining algorithms are inapplicable to identify itemset that appear in a temporal database regularly.

Besides, many studies focus on finding new kinds of itemset and rules present in a large amount of data. This is especially important with the emergence of Big data. Over nearly 30 past years, various itemset have been identified, namely sequential and time-series itemset [32], [33], [34], high utility itemset [35], [36], [37], structural itemset [38], [39], temporal (periodic) itemset [40], [41], [42], [43].

### B. PERIODIC-FREQUENT ITEMSET MINING

The target of periodic frequent itemset mining is to identify how regularly the itemset occur in a temporal database. In Tanbeer et al. [19], the problem of mining periodic frequent itemset was first introduced and correspondingly a model called PF-growth was proposed to tackle this problem. Compared to the classic FIM which only employs the *minSup* constraint, PFIM includes one more parameter called *maxPer*. This algorithm performs in two steps. First, it represents the database by a periodic-frequent tree (PF-tree), and items in a PF-tree are arranged in the descending item support order. Second, the algorithm mines the PF-tree by using FP-growth mining technique to find all periodic-frequent itemset.

Amphawan et al. [44] proposed an efficient algorithm called Mining Top-K Periodic-frequent itemset (MTKPP), which is based on a depth-first search and a vertical database representation. This algorithm mines periodic-frequent itemset without using the *minSup* constraint and provides a list-based data structure called the top-K list to maintain the set of k regular itemset with the highest support. MTKPP algorithm uses this top-K list during the mining process to generate candidate itemset. Kiran and Reddy [45] introduced an efficient model that extended multiple *minSup*'s and multiple *maxPer*'s to discover periodic-frequent itemset consisting of both frequent and rare items. This model used two different constraints to identify useful itemset, namely minimum item support and maximum item periodicity. Each itemset satisfies different *minSup* and *maxPer* values based on the available items in the itemset. That study also proposed a pattern-growth algorithm using a novel and efficient tree-based data structure, named a multi-constraint periodic-frequent tree, to find the complete set of frequent and rare items.

Amphawan et al. [44] proposed a novel technique called approximate periodicity to reduce the calculation time requirements of mining periodic-frequent itemset. This algorithm splits the transactional timeline into intervals with different *maxPer* values. The interval information is stored only when there exists a itemset in that interval. The authors also proposed a tree structure, called Interval Transaction-ids List tree (ITL-tree). The goal of this technique is to maintain the occurrence information in a highly compact manner by using interval transaction-ids list instead of tid-list. Then the approximate periodicity of each itemset can be found. To generate all periodic-frequent itemsets, a itemset growth mining technique is also used by a bottom-up traversal of the ITL-tree based on *minSup* and *maxPer* constraints. An interesting novel measure was proposed by Kiran and Reddy [46] to extract periodic-frequent itemset in a transactinonal database, which is called periodic-ratio. The authors defined that some itemset which appear almost periodically in the database can be considered interesting itemset. Therefore, a periodic interestingness of a frequent itemset is calculated as the ratio of its periodic occurrences in a database. A itemset can be defined as a potential itemset if its support is greater than *minSup* and its periodic interestingness is greater than the user-specified minimum periodic-ratio. Then an extended periodic-frequent tree was built based on these potential itemset. Also a pattern-growth algorithm was proposed to find the complete set of periodic-frequent itemset. Ravikumar et al. [47] have described an algorithm named PF- ECLAT, to efficiently discover periodic-frequent itemsets in a columnar temporal databases. Some other variations of the above models were also proposed [44], [48], and [49] to find periodic-frequent itemset. However, all these algorithms have a drawback that, if only one of the periods of a itemset exceeds *maxPer*, this itemset is discarded. Kiran et al. [40] proposed a model called partial PFP mining that relaxes the maximum periodicity constraint by considering that a itemset X is (partial) periodic if its periodic-frequency is no less than a user-specified threshold. However, this algorithm cannot be applied to find stable-periodic-frequent itemset. The reason is that it measures the periodicity of a itemset by counting the number of times where the periods of a itemset are less than *maxPer* without considering how much these periods deviate from *maxPer*.

To overcome the drawback of periodic frequent itemset mining, Fournier-Viger et al. [26] proposed a model to find stable periodic-frequent itemset in a transactional database.

### C. STABLE PERIODIC-FREQUENT ITEMSET MINING

Fournier-Viger et al. [26] introduced a concept called *lability*, which is the cumulative sum of the difference between each period length and *maxPer* constraint. A novel parameter, called *maximum lability* (*maxLa*), was also used to assess the stability of a periodic behavior of a itemset in a database. An algorithm named SPP-growth to mine stable periodic-frequent itemset was presented with two steps. First, the database is represented as a stable periodic-frequent tree (SPP-tree), and then the algorithm mines the SPP-tree to find all stable periodic-frequent itemset. He et al. [50] discussed a model to find stable periodic-frequent itemset in an uncertain database. The authors proposed a Stable Periodic Frequent Itemset Mining (SPFIM) algorithm on an uncertain database by considering both the frequency and periodicity of itemset. That is, an itemset X in an uncertain transaction database is considered a stable periodic frequent itemset if the support count and stability value of itemset X meet the minimum support threshold (*minSup*) and the stability threshold (*maxLa*). Fournier-Viger et al. [51] proposed a model using the concept of top-$K$ mining to generate stable periodic-frequent itemset. This study introduced an algorithm that, rather than using a *minSup* threshold, the user can directly specify parameter $k$, where $k$ represents the number of itemset that the user wants to find. The output of the

**TABLE 1.** Item's dictionary with their timestamp list.

| *ts* | items | *ts* | items |
|---|---|---|---|
| 1 | *b,c,d,e* | 7 | *d,e,f* |
| 2 | *a,b,c* | 8 | *b,c,d* |
| 3 | *a,b,c,d* | 9 | *a,b,c,d* |
| 4 | *e,f* | 10 | *a,b,c* |
| 5 | *a,c,d* | 11 | *a,c,e* |
| 6 | *a,b,c* | 12 | *b,c,d* |

**TABLE 2.** Columnar database.

| | items | | | | | | | items | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| *ts* | *a* | *b* | *c* | *d* | *e* | *f* | *ts* | *a* | *b* | *c* | *d* | *e* | *f* |
| 1 | 0 | 1 | 1 | 1 | 1 | 0 | 7 | 0 | 0 | 0 | 1 | 1 | 1 |
| 2 | 1 | 1 | 1 | 0 | 0 | 0 | 8 | 0 | 1 | 1 | 1 | 0 | 0 |
| 3 | 1 | 1 | 1 | 1 | 0 | 0 | 9 | 1 | 1 | 1 | 1 | 0 | 0 |
| 4 | 0 | 0 | 0 | 0 | 1 | 1 | 10 | 1 | 1 | 1 | 0 | 0 | 0 |
| 5 | 1 | 0 | 1 | 1 | 0 | 0 | 11 | 1 | 0 | 1 | 0 | 1 | 0 |
| 6 | 1 | 1 | 1 | 0 | 0 | 0 | 12 | 0 | 1 | 1 | 1 | 0 | 0 |

**TABLE 3.** Item's dictionary with their timestamp list.

| item | TS-list |
|---|---|
| *a* | 2,3,5,6,9,10,11 |
| *b* | 1,2,3,6,8,9,10,12 |
| *c* | 1,2,3,5,6,8,9,10,11,12 |
| *d* | 1,3,5,7,8,9,12 |
| *e* | 1,4,7,11 |
| *f* | 4,7 |

algorithm is the top-K most frequent itemset that have a stable periodic behavior. To the best of our knowledge, up to now, there have been only three above references related to the study of finding a Stable Periodic-Frequent Itemset in row databases.

Because all of the above algorithms find the itemsets in row databases, whenever we give a columnar temporal database as an input to the above algorithms, it has to be converted into a row temporal database to get interesting itemsets. As a result, the above algorithms will take longer to run and use more memory because of this conversion overhead. In contrast, the algorithm proposed in our work is different in that it deals specifically with columnar databases.

## III. MODEL OF STABLE PERIODIC-FREQUENT ITEMSET

Let $O = \{o_1, o_2, \cdots, o_n\}$, $n \geq 1$, be a set of objects (or items). Let $Y \subseteq O$ be an itemset (or a pattern). Let $t_a = (ts, X)$, $a \geq 1$, be a transaction, where $ts \in \mathbb{R}^+$ represents the timestamp and $X$ is an itemset. Let $TDB = \{t_1, \cdots, t_d\}$, $d \geq 1$ be a temporal database representing an ordered set of transactions such that $t_a.ts \leq t_b.ts$, where $1 \leq a < b \leq d$. Let $TS^Y = \{ts_i^Y, \cdots, ts_j^Y\}, i, j \in [1, d]$, denote a set of timestamps containing $Y$ in $TDB$.

*Example 1:* Assume that we have a set of items $I = \{a, b, c, d, e, f\}$. Table 1 shows a row temporal database constituting of these items. Without loss of generality, this database can be viewed as a columnar temporal database as shown in Table 2. In Table 3, we show the dictionary storing the items and their temporal occurrence information in the database. The set of items '$b$' and '$c$', i.e., $\{b, c\}$ is a itemset. This itemset will be represented as '$bc$' for brevity. This itemset is denoted as 2-itemset because it contains two items. The occurrences of itemset '$bc$' are at the timestamps of $1, 2, 3, 6, 8, 9, 10,$ and $12$. Therefore, we have a list of timestamps containing '$bc$', i.e., $TS^{bc} = \{1, 2, 3, 6, 8, 9, 10, 12\}$.

*Definition 1 (The Support of Y):* The **support** of $Y$, denoted as $sup(Y)$, represents the number of transactions containing $Y$ in $TDB$. That is, $sup(Y) = |TS^Y|$.

*Example 2:* The *support* of '$bc$', i.e., $sup(bc) = |TS^{bc}| = 8$.

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

*Example 3:* If $minSup = 5$, then $bc$ is said to be a frequent itemset because $sup(bc) \geq minSup$.

*Definition 3 (Periodicity of Y):* Let $ts_m^Y$ and $ts_n^X, j \leq m < n \leq k$, denote 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 $P^Y = (p_1^Y, p_2^Y, \cdots, p_n^Y)$ the set of all *periods* for itemset $Y$. The ***periodicity*** of $Y$, denoted by $per(Y) = maximum(p_1^Y, p_2^Y, \cdots, p_n^Y)$.

*Example 4:* All periods of the itemset '$bc$' are: $p_1^{bc} = 1\ (= 1 - ts_{initial})$, $p_2^{bc} = 1\ (= 2 - 1)$, $p_3^{bc} = 1\ (= 3 - 2)$, $p_4^{bc} = 3\ (= 6 - 3)$, $p_5^{bc} = 2\ (= 8 - 6)$, $p_6^{bc} = 1\ (= 9 - 8)$, $p_7^{bc} = 1\ (= 10 - 9)$, $p_8^{bc} = 2\ (= 12 - 10)$, and $p_9^{bc} = 0\ (= ts_{final} - 12)$, where first transaction time stamp is denoted by $ts_{initial} = 0$ and the last transaction's time stamp is denoted by, $ts_{final} = |TDB| = 12$. The *periodicity* of $bc$, i.e., $per(bc) = maximum(1, 1, 1, 3, 2, 1, 1, 2, 0) = 3$.

*Definition 4 (Periodic-Frequent Itemset Y):* The frequent itemset $Y$ be considered as **periodic-frequent itemset** 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 itemset '$bc$' is called as a periodic-frequent itemset as $per(bc) \leq maxPer$.

*Definition 5 (Lability of an Itemset):* Let $ts_{i+1}^Y$ and $ts_i^Y, i \in [0, sup(Y)]$, be 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 $Y$: $la(Y) = \{la(Y, 0), la(Y, 1), \cdots, la(Y, sup(Y))\}$, and $|la(Y)| = |per(Y)| = sup(Y) + 1$.

*Example 6:* Consider an item $a$. If $maxPer$ =2, then the *lability* of $a$ are: $la(a, 0) = max(0, la(p, -1) + p_0^p - maxPer) = max(0, 0 + 2 - 2) = 0$, $la(a, 1) = max(0, 0 + 1 - 2) = 0$, $la(a, 2) = max(0, 0 + 2 - 2) = 0$, $la(a, 3) = max(0, 0 + 1 - 2) = 0$, $la(a, 4) = max(0, 0 + 3 - 2) = 1$, and $la(a, 5) = max(0, 1 + 1 - 2) = 0$. Therefore, the sequence of labilities of $a$ in the database, i.e., $la(a) = \{0, 0, 0, 0, 1, 0\}$.

Based on Definition 5, the periodic itemset 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 itemset will increase when a period of a itemset 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 itemset no more than *maxPer*. Therefore, according to the periodic characteristic of a itemset, 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 itemset can be found using this measure given a limit on the *maximum lability*.

*Definition 6 (Stable Periodic-Frequent Itemset):* For a itemset $Y$, denote la(Y) the set of all i-th *lability*. The stability of the itemset is defined by $maxLa(Y) = \max(la(Y))$. itemset $Y$ is a SPI 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 SPIs are $a$: (7,1), $b$: (8,1), $c$: (10,0), $d$: (7,1), $bc$: (8,1),$bca$: (5,1), $cd$: (6,1), $ca$: (7,1), where each SPI $Y$ is annotated with $Y$: ($sup(Y)$, $maxLa(Y)$).

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

*Definition 7 (Problem Definition):* Given a temporal database (*TDB*) with *minimum support* (*minSup*), *maximum periodicity* (*maxPer*), and *maximum lability* (*maxLa*) constraints, our objective is to discover the complete set of stable periodic-frequent itemset having *support* higher or equal to *minSup* and *lability* lower or equal to *maxLa* constraints.

## IV. THE PROPOSED ALGORITHM: SPP-ECLAT

The itemset lattice represents the search space of stable periodic-frequent itemset mining. The size of this lattice is $2^n - 1$, where $n$ represents the total number of items in a database. Using the *downward closure property* (see Property 1) and the depth-first search technique, the proposed SPP-ECLAT searches this huge lattice and finds the complete set of SPIs. Briefly, the SPP-ECLAT algorithm involves the following two steps: ($i$) find the stable periodic items (or 1-itemsets) from a database (Section IV-A) and ($ii$) discover the complete set of stable periodic $k$-itemsets, $k > 1$, by recursively mining the previously generated stable periodic itemsets (Section IV-B). We now explain each of these steps in detail.

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

### A. MINING A 1-STABLE PERIODIC-FREQUENT ITEMSET

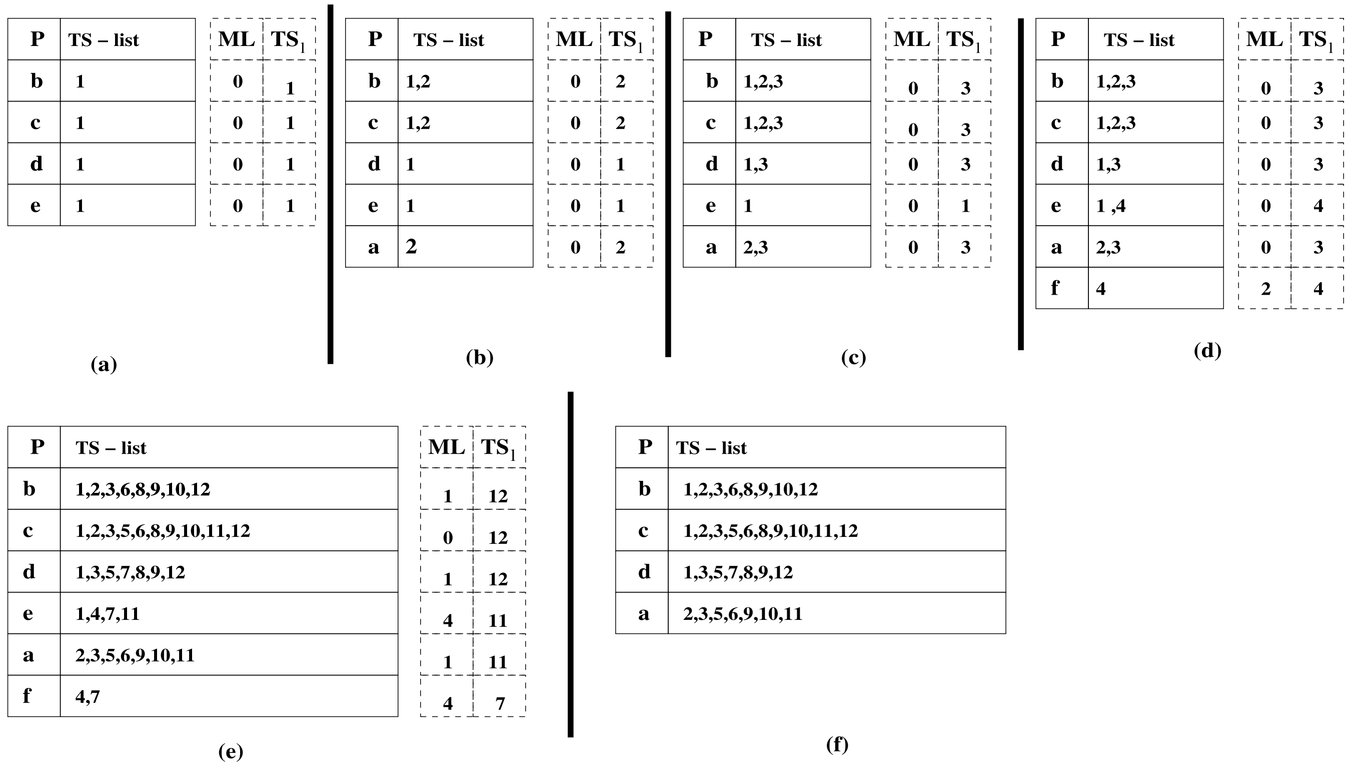
The proposed algorithm can find stable periodic-frequent itemsets in both row and columnar databases. The proposed algorithm achieves this ability by transforming a row and columnar database into a unified data structure constituting of candidate items and transaction identifiers. This data structure is called SPP-list.

**Algorithm 1** StablePeriodicFrequentItems(Temporal Database (*TDB*), Minimum Support (*minSup*), Maximum Periodicity (*maxPer*), Maximum Lability(*maxLa*):

```
1: Definition: $SPP\text{-}list = (Y, TS\text{-}list(Y))$ is a dictionary with the temporal occurrence
   information of a itemset in a TDB; $TS_l$ is a temporary variable of list type to store the
   timestamp of the final occurrence of a itemset; la and ML are temporary variable of
   list type to store the lability and the Maximum Lability of a itemset; last is a term for
   the final timestamp; support is a temporary varibale of list type to store the support
   of a itemset.
2: Initate $ts_{cur} = 0$
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 exit 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:            $la[j] = max(0, la[j] + ts_{cur} - TS_l[j] - maxPer)$
12:            $ML[j] = max(la[j], ML[j])$
13:            Update $TS_l[j] = ts_{cur}$.
14:        end if
15:    end for
16:    $last = ts_{cur}$
17: end for
18: for each item $j$ in SPP-list do
19:     $la[j] = max(0, la[j] + last - TS_l[j] - maxPer)$
20:     $ML[j] = max(la[j], ML[j])$
21:     $s[j] = length(TS\text{-}list[j])$
22:     if $s[j] < minSup$ and $ML[j] > maxla$ then
23:         Prune j from SPP-list
24:     end if
25: end for
26: After the pruning the final list of itemset available in the SPP-list is sorted in
    ascending order or descending order of the corresponding itemset's support. Initiate
    pi as Null. Call SPP-ECLAT(SPP-List, pi).
```

Denote $SPP\text{-}list = (Y, TS\text{-}list(Y))$ a dictionary with the temporal occurrence information of a itemset in a *TDB*; $TS_l$ is a temporary variable of list type to store the *timestamp* of the final occurrence of a itemset; *la* and *ML* are temporary variable of list type to store the *lability* and the *Maximum Lability* of a itemset; *last* is a term for the final timestamp; *support* is a temporary varibale of list type to store the *support* of a itemset. This part focuses on discovering 1-itemset 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-itemset 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, "$1 : b, c, d, e$", with $ts_{cur} = 1$ inserts the items $b$, $c$, $d$ and $e$, in the SPP-list. We have the timestamps of these items is 1 ($= ts_{cur}$). Similarly, $ML$ and $TS_l$ contents were updated to 0 and 1, respectively (lines 7 and 8 in Algorithm 1). Fig. 1(a) shows the generated SPP-list from the 1st transaction. After reading the 2nd one, "$2 : a, b, c$", 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



(a)

| P | TS – list | ML | $TS_l$ |
|---|---|---|---|
| b | 1 | 0 | 1 |
| c | 1 | 0 | 1 |
| d | 1 | 0 | 1 |
| e | 1 | 0 | 1 |

(b)

| P | TS – list | ML | $TS_l$ |
|---|---|---|---|
| b | 1,2 | 0 | 2 |
| c | 1,2 | 0 | 2 |
| d | 1 | 0 | 1 |
| e | 1 | 0 | 1 |
| a | 2 | 0 | 2 |

(c)

| P | TS – list | ML | $TS_l$ |
|---|---|---|---|
| b | 1,2,3 | 0 | 3 |
| c | 1,2,3 | 0 | 3 |
| d | 1,3 | 0 | 3 |
| e | 1 | 0 | 1 |
| a | 2,3 | 0 | 3 |

(d)

| P | TS – list | ML | $TS_l$ |
|---|---|---|---|
| b | 1,2,3 | 0 | 3 |
| c | 1,2,3 | 0 | 3 |
| d | 1,3 | 0 | 3 |
| e | 1 ,4 | 0 | 4 |
| a | 2,3 | 0 | 3 |
| f | 4 | 2 | 4 |

(e)

| P | TS – list | ML | $TS_l$ |
|---|---|---|---|
| b | 1,2,3,6,8,9,10,12 | 1 | 12 |
| c | 1,2,3,5,6,8,9,10,11,12 | 0 | 12 |
| d | 1,3,5,7,8,9,12 | 1 | 12 |
| e | 1,4,7,11 | 4 | 11 |
| a | 2,3,5,6,9,10,11 | 1 | 11 |
| f | 4,7 | 4 | 7 |

(f)

| P | TS – list |
|---|---|
| b | 1,2,3,6,8,9,10,12 |
| c | 1,2,3,5,6,8,9,10,11,12 |
| d | 1,3,5,7,8,9,12 |
| a | 2,3,5,6,9,10,11 |

**FIGURE 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 itemset.

Fig. 1(b). After reading the 3rd one, "3 : $a, b, c, d$", updates the TS-list, $ML$ and $TS_l$ values of $a$, $b$, $c$, and $d$ in the SPP-list. Fig. 1(c) shows the SPP-list which is generated after reading the $3^{b,c,d,a}$ one. After reading the 4th one, "4 : $e, f$" with $ts_{cur} = 4$, inserts the new items $e$ and $f$ into the SPP-list by adding 4 ($= ts_{cur}$) in their TS-list. Simultaneously, the $ML$ and $TS_l$ values as 2 and 4. Fig. 1(d) shows the SPP-list which is generated after reading the 4th. We repeat the whole process for the remaining transactions. Fig. 1(e) depicts the final SPP-list which is generated after scanning the whole database. The itemset $e$ and $f$ 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 itemset available in the SPP-list are considered as 1-stable periodic-frequent itemset. Those itemset are sorted in descending order in terms of their *support* values. Fig. 1(f) shows the final SPP-list.

### B. FINDING ALL INTERESTING ITEMSET FROM SPP-LIST

The detailed procedure for finding stable periodic-frequent itemset 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 itemset $b$, as this is the initial itemset in the SPP-list (line 2 in Algorithm 2). Fig. 2(a) shows a record of its *support* and *lability*. Since $b$ is a stable periodic-frequent itemset, we move to its child node $bc$.

**Algorithm 2** SPP-ECLAT(SPP-List, Pi)

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

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

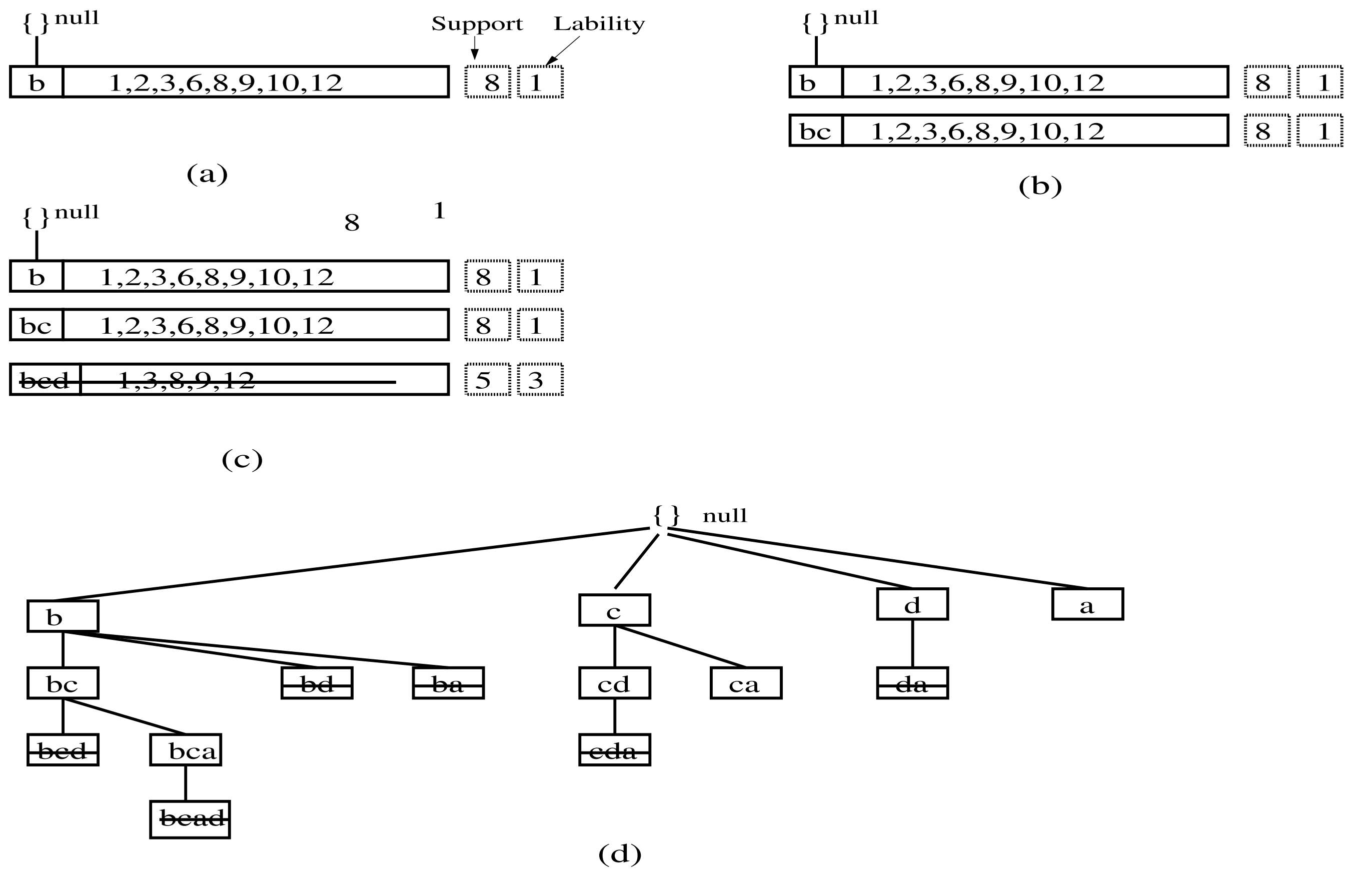


**FIGURE 2.** The complete process of discovering stable periodic-frequent itemset using SPP-ECLAT algorithm.

**TABLE 4.** Statistics of the databases.

| S.No | Database | Type | Nature | Sparsity value | Transaction Length (in count) min. | avg. | max. | Database Size (in count) |
|---|---|---|---|---|---|---|---|---|
| 1 | T10I4D100K | Synthetic | Sparse | 0.9883881839 | 1 | 10 | 29 | 100,000 |
| 2 | Retail | Real | Sparse | 0.9998717616 | 2 | 12 | 77 | 88,162 |
| 3 | T20I6D100K | Synthetic | Sparse | 0.9788118926 | 2 | 19 | 46 | 199,844 |
| 4 | BMS-WebView-1 | Real | Sparse | 0.994948416 | 1 | 3 | 267 | 59,602 |
| 5 | BMS-WebView-2 | Real | Sparse | 0.9986160999 | 2 | 5 | 161 | 77,512 |
| 6 | Mushroom | Real | Dense | 0.9925705221 | 23 | 23 | 23 | 8,124 |

## V. EXPERIMENTAL RESULTS

This section evaluates the performance of the SPP-ECLAT against the state-of-the-art SPP-growth [26] algorithm. Through experiment results, we will show that the SPP-ECLAT algorithm is more efficient in memory consumption and runtime than SPP-growth. The scalability of the SPP-ECLAT algorithm is also shown to demonstrate the superior efficacy and productivity over the SPP-Growth algorithm on big columnar temporal databases. The implementations of these two algorithms are available at PAMI [52]. Note that the metric for runtime is seconds and memory is bytes throughout the experimentation.

### A. EXPERIMENTAL SETUP

The algorithms, SPP-growth and SPP-ECLAT, were developed in Python 3.7 and executed on a Gigabyte R282-z94 rack server 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 various real-world databases including T10I4D100K, Retail, T20I6D100K, BMS-WebView-1, BMS-WebView-2 and Mushrooms. The characteristics of these databases are shown in the Table 4. The **T10I4D100K** and **T20I6D100K** synthetic databases are generated according to the properties of market basket data. The procedure of constructing these databases is described in [6]. These spare databases have been widely employed to evaluate various itemset-mining algorithms in the literature. The **BMS-WebView-1** and **BMS-WebView-2** are a real-world sparse databases containing clickstream data from e-commerce sites. Each transaction is a viewing session consisting of all the viewed product detail pages where each product detail view is an item. These databases contain very long transactions and they were used in KDD CUP 2000 competition [53]. The **Retail** is a real-world sparse database consisting of basket databases in a retail supermarket store. The Retail database is provided by Brijs [54]. The **Mushrooms** is a real-world dense database containing different species of gilled mushrooms prepared from the UCI mushrooms dataset. All of the above databases have been downloaded from SPMF repository [55].

### B. EXPERIMENT 1: EVALUATION OF SPP-GROWTH AND SPP-ECLAT ALGORITHMS BY VARYING *minSup* AND *maxLa* VALUES

In this experiment, we study the impact of *minSup* and *maxLa* constraints on the number of itemsets generated, the runtime requirements of SPP-Growth and SPP-ECLAT algorithms, and the memory consumed by SPP-Growth and SPP-ECLAT algorithms. Please note that the *maxPer* constraint has been fixed at a particular value for each database throughout this experimentation.

First, we have shown the number of SPIs generated by SPP-Growth and SPP-ECLAT algorithms in Figure 3 by varying the value of *maxLa*. In detail, Figure 3(a) to Figure 3(c), shows the number of SPIs generated by both the algorithms in the T10I4D100K database, respectively, for different *minSup* and *maxPer* values. From Figure 3(d) to Figure 3(f), shows the number of SPIs generated by both the algorithms in the Retail database, respectively, for different *minSup* and *maxPer* values. From Figure 3(g) to Figure 3(i), shows the number of SPIs generated by both the algorithms in the T20I6D100K database, respectively, for different *minSup* and *maxPer* values. From Figure 3(j) to Figure 3(l), shows the number of SPIs generated by both the algorithms in the BMS-WebView1 database, respectively, for different *minSup* and *maxPer* values. From Figure 3(m) to Figure 3(o), shows the number of SPIs generated by both the algorithms in the BM-WebView-2 database, respectively, for different *minSup* and *maxPer* values. From Figure 3(p) to Figure 3(r), shows the number of SPIs generated by both the algorithms in the Mushrooms database, respectively, for different *minSup* and *maxPer* values. It is to be noted that both algorithms will generate an equal number of itemsets. Therefore, both the curves were overlapped throughout the figures. The following two observations can be drawn from these figures: (*i*) The *minSup* constraint has negative effect on the generation of SPIs. That is, increase in *minSup* decreases the number of SPIs, and vice-versa. It is because many itemsets fail to satisfy the increased *minSup* value. (*ii*) The *maxLa* constraint has positive effect on the generation of SPIs in the T10I4D100k, T20I6D100k, Retai, and BMS-WebView1 sparse database. That is, increase in *maxLa* increases the number of SPIs, and vice-versa. It is because higher *maxLa* values facilitate the itemsets to have their inter-arrival times further away from the user-specified *maxPer* value. (*iii*) In the dense Mushrooms database, the *maxLa* constraint does not affect the generation of SPIs. It is because, in a dense database, the mined periodic-frequent itemsets are the ones that appear regularly in the database, i.e. having a stable behavior.

Next, we have shown the runtime requirements of SPP-Growth and SPP-ECLAT algorithms in Figure 4 by varying the value of *maxLa*. In detail, Figure 4 (a) to Figure 4(c), shows the runtime requirements of both the algorithms in the T10I4D100K database, respectively, for different *minSup* and *maxPer* values. Figure 4 presents the performance comparison of the two algorithms in three cases, minSup = 0.5% and maxPer = 0.4% (Figure 4(a)), minSup = 0.8% and maxPer = 0.4% (Figure 4(b)), minSup = 1% and maxPer = 0.4% (Figure 4(c)). The results in these three cases all show that SPP-ECLAT is faster than SPP-Growth.

From Figure 4(d) to Figure 4(f), shows the runtime requirements of both the algorithms in the Retail database, respectively, for different *minSup* and *maxPer* values. Figure 4 presents the performance comparison of the two algorithms in three cases, minSup = 0.8% and maxPer = 2% (Figure 4(d)), minSup = 0.9% and maxPer = 2% (Figure 4(e)), minSup = 1% and maxPer = 2% (Figure 4(f)). The results in these three cases all show that, in general, SPP-ECLAT is faster than SPP-Growth.

From Figure 4(g) to Figure 4(i), shows the runtime requirements of both the algorithms in the T20I6D100K database, respectively, for different *minSup* and *maxPer* values. The runtime analysis is given in Figure 4 shows the performance of the two algorithms in three cases, minSup = 3% and maxPer = 4% (Figure 4(g)), minSup = 5% and maxPer = 4% (Figure 4(h)), minSup = 7% and maxPer = 4% (Figure 4(i)). The results in these three cases all show that SPP-ECLAT is faster than SPP-Growth.

From Figure 4(j) to Figure 4(l), shows the runtime requirements of both the algorithms in the BMS-WebView1 database, respectively, for different *minSup* and *maxPer* values. Figure 4 depicts the performance comparison of the two algorithms in three cases, minSup = 0.5% and maxPer = 5% (Figure 4(j)), minSup = 0.8% and maxPer = 5% (Figure 4(k)), minSup = 1% and maxPer = 5% (Figure 4(l)). The results in these three cases all show that SPP-ECLAT is faster than SPP-Growth.

From Figure 4(m) to Figure 4(o), shows the runtime requirements of both the algorithms in the BMS-WebView2 database, respectively, for different *minSup* and *maxPer* values. Figure 4 shows the performance comparison of the two algorithms in three cases, minSup = 0.06% and maxPer = 5% (Figure 4(m)), minSup = 0.08% and maxPer = 0.5% (Figure 4(o)), minSup = 0.1% and maxPer = 0.5% (Figure 4(l)). The results in these three cases show that SPP-ECLAT is faster than SPP-Growth; however, the difference in runtime comparison between the two algorithms is small, it is only around 0.3 seconds on average .

From Figure 4(p) to Figure 4(r), shows the analysis for the runtime requirements of both the algorithms in the Mushrooms database, respectively, for different *minSup* and *maxPer* values. Figure 4 presents the performance

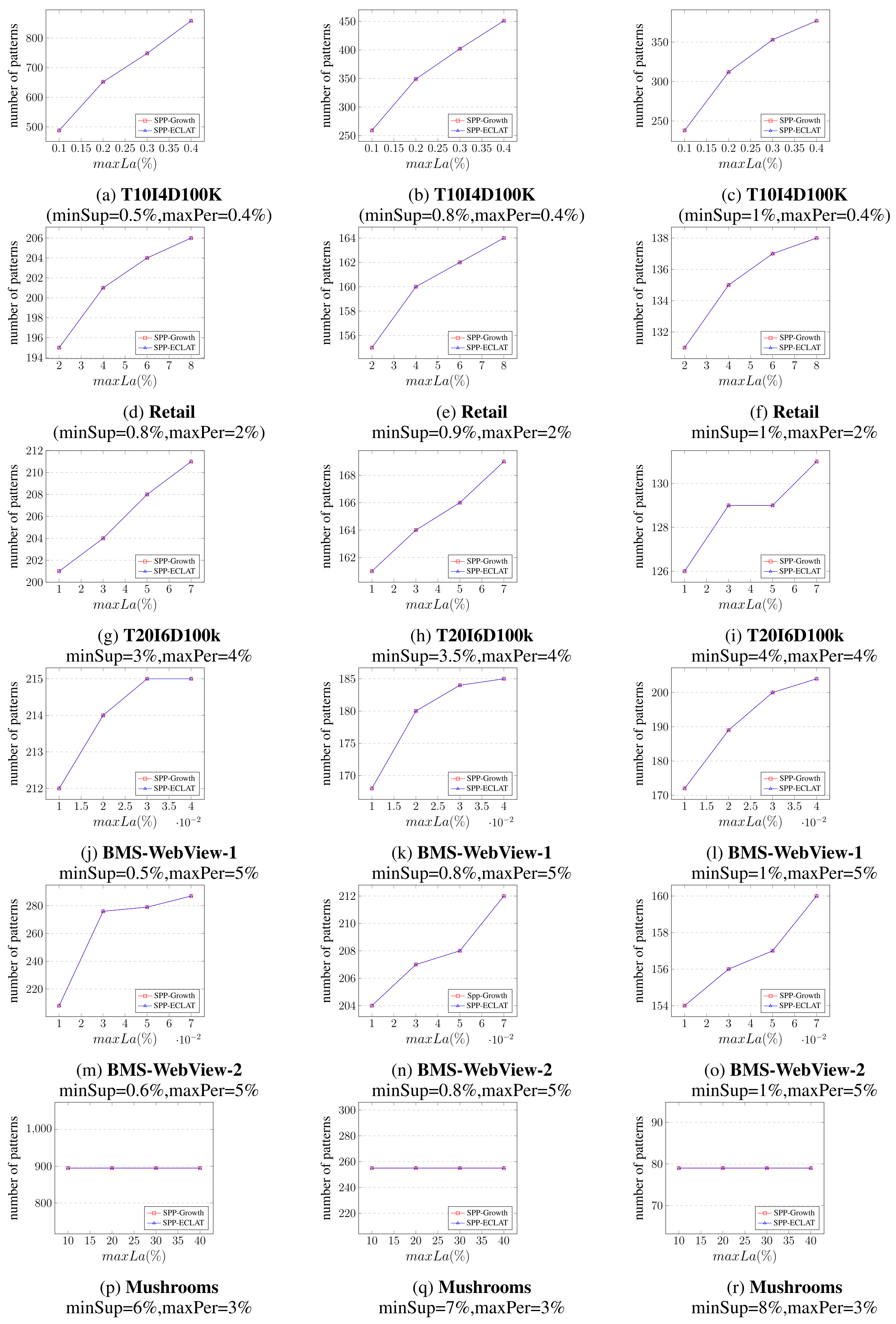
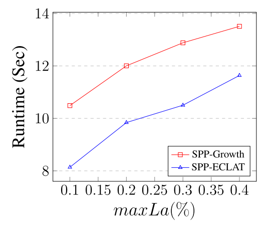
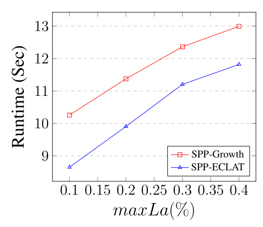


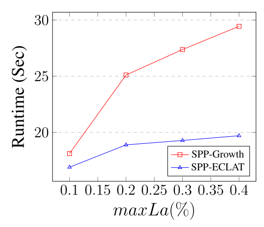
**FIGURE 3.** Number of stable periodic-frequent itemsets generated in various databases by varying *minSup* and *maxLa* values.



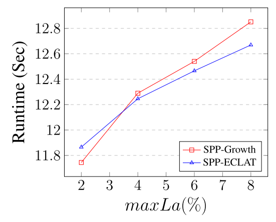
(a) **T10I4D100K**
$minSup$=0.5%, $maxPer$=0.4%



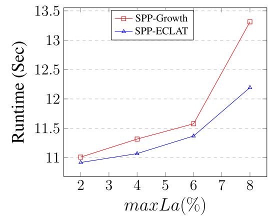
(b) **T10I4D100K**
$minSup$=0.8%, $maxPer$=0.4%



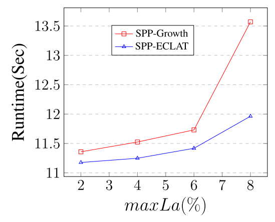
(c) **T10I4D100K**
$minSup$=1%,$maxPer$=0.4%



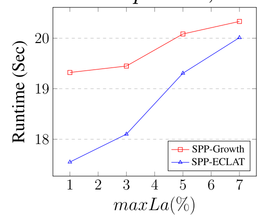
(d) **Retail**
$minSup$=0.8%, $maxPer$=2%



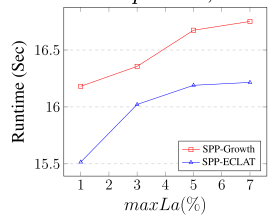
(e) **Retail**
$minSup$=0.9%, $maxPer$=2%



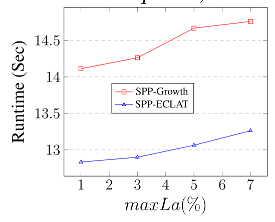
(f) **Retail**
$minSup$=1%, $maxPer$=2%



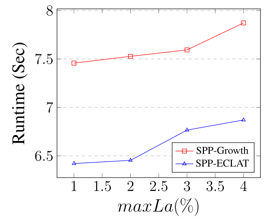
(g) **T20I6D100k**
$minSup$=3%,$maxPer$=4%



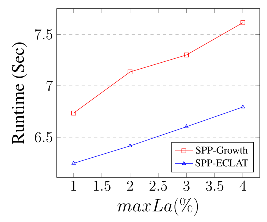
(h) **T20I6D100k**
$minSup$=5%,$maxPer$=4%



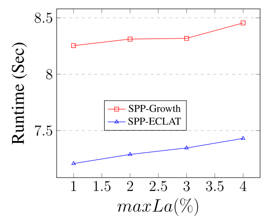
(i) **T20I6D100k**
$minSup = 7\%, maxPer$=4%



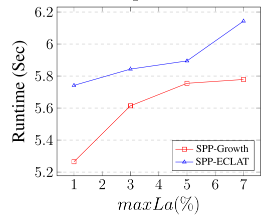
(j) **BMS-WebView-1**
$minSup$=0.5%, $maxPer$=5%



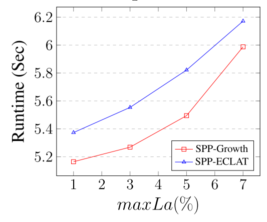
(k) **BMS-WebView-1**
$minSup$=0.8%, $maxPer$=5%



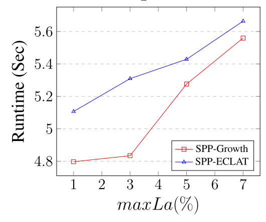
(l) **BMS-WebView-1**
$minSup$=1%, $maxPer$=5%



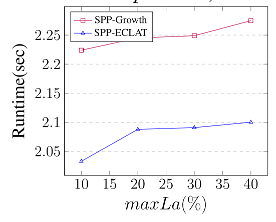
(m) **BMS-WebView-2**
$minSup$=0.6%, $maxPer$=5%



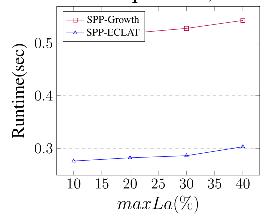
(n) **BMS-WebView-2**
$minSup$=0.8%, $maxPer$=5%



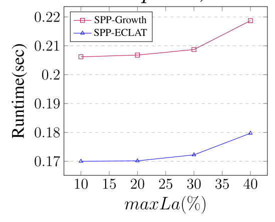
(o) **BMS-WebView-2**
$minSup$=1%, $maxPer$=5%



(p) **Mushroom**
$minSup$=6%, $maxPer$=3%



(q) **Mushroom**
$minSup$=7%, $maxPer$=3%



(r) **Mushrooms**
$minSup$=8%, $maxPer$=3%

**FIGURE 4.** **Runtime requirements of SPP-Growth and SPP-ECLAT algorithms at different *maxLa*.**

comparison of the two algorithms in three cases, minSup = 6% and maxPer = 3% (Figure 4(m)), minSup = 7% and maxPer = 3% (Figure 4(n)), minSup = 8% and maxPer = 3% (Figure 4(o)). The results in these three cases all show that SPP-Growth requires more time than SPP-ECLAT.

It can be observed that the SPP-ECLAT runs faster than the SPP-Growth algorithm. The good performance of SPP-ECLAT is a result of the effectiveness of periodic calculation and pruning techniques. The following are some noteworthy findings that can be derived from this figure: (*i*) If we increase the *maxLa* value, then subsequently, both algorithms' runtime requirements increase. The primary reason for this observation is that both the algorithms will discover many SPIs in any database if the *maxLa* value continues to increase. (*ii*) SPI-ECLAT generates SPIs much faster than SPP-Growth under any given *maxLa* in BMS-WebView-1, Retail, T10I4D100K, and T20I6D100K, and Mushrooms databases. More importantly, we can also observe that at high *maxLa* values, SPP-ECLAT algorithm generates the SPIs much faster than SPP-Growth algorithm. The reason is that SPP-ECLAT uses the downward closure property and the depth-first search technique, so the SPIs are generated by simply performing intersection of SPP-list. The process is repeated to find all SPIs. (*iii*) With the BMS-WebView-2 dataset, which contains long transactions and many distinct items, the SPP-ECLAT algorithm takes more time than the SPP-Growth algorithm. It is because the SPP-ECLAT algorithm is based on the downward closure property and the depth-first search technique, so it does not require scanning the database each time; but to generate all SPIs, it has to perform the intersection of the SPP-list. So the long SPP-list requires more time to repeat the intersection process.

Finally, we have shown the memory consumption details of SPP-Growth and SPP-ECLAT algorithms in Figure 5 by varying the value of *maxLa*. In detail, Figure 5(a) to Figure 5(c), shows the memory consumption of both the algorithms in the T10I4D100K database, respectively, for different *minSup* and *maxPer* values. Figure 5(a) depicts the comparison of the two algorithms in three cases, minSup = 0.5% and maxPer = 0.4% (Figure 5(a)), minSup = 0.8% and maxPer = 0.4% (Figure 5(b)), minSup = 1% and maxPer = 0.4% (Figure 5(c)). The results all show that SPP-ECLAT consumes less memory than SPP-Growth in all cases. When maxLa is 0.1%, SPP-ECLAT consumes less memory than SPP-Growth by 26 MB on average. As maxLa is increased, the performance gap becomes bigger (up to 62 MB).

From Figure 5(d) to Figure 5(f), shows the memory consumption of both the algorithms in the Retail database, respectively, for different *minSup* and *maxPer* values. Figure 5 depicts the performance of the two algorithms in three cases, minSup = 0.8% and maxPer = 2% (Figure 5(d)), minSup = 0.9% and maxPer = 2% (Figure 5(e)), minSup = 1% and maxPer = 2% (Figure 5(f)). The results all show that SPP-ECLAT performs consistently and consumes less memory than SPP-Growth by 145 MB on average.

From Figure 5(g) to Figure 5(i), shows the memory consumption of both the algorithms in the T20I6D100K database, respectively, for different *minSup* and *maxPer* values. Figure 5 presents the performance comparison of the two algorithms in three cases, minSup = 3% and maxPer = 4% (Figure 5(g)), minSup = 3.5% and maxPer = 4% (Figure 5(h)), minSup = 4% and maxPer = 4% (Figure 5(i)). In these cases, SPP-ECLAT consumes less memory than SPP-Growth by around 130MB on average.

From Figure 5(j) to Figure 5(l), shows the memory consumption of both the algorithms in the BMS-WebView1 database, respectively, for different *minSup* and *maxPer* values. Figure 5 shows the performance comparison of the two algorithms in three cases, minSup = 0.5% and maxPer = 5% (Figure 5(j)), minSup = 0.8% and maxPer = 5% (Figure 5(k)), minSup = 1% and maxPer = 5% (Figure 5(l)). The gain of SPP-ECLAT in terms of memory here is about 4MB on average.

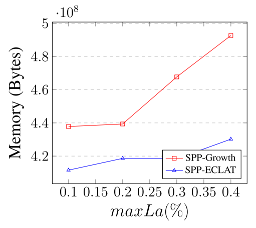
From Figure 5(m) to Figure 5(o), shows the memory consumption of both the algorithms in the BMS-WebView2 database, respectively, for different *minSup* and *maxPer* values. Figure 5 shows the performance comparison of the two algorithms in three cases, minSup = 0.6% and maxPer = 5% (Figure 5(m)), minSup = 0.8% and maxPer = 5% (Figure 5(n)), minSup = 1% and maxPer = 0.5% (Figure 5(o)). The results show that the memory consumption of SPP-ECLAT is around 18MB less than SPP-Growth.

From Figure 5(p) to Figure 5(r), shows the memory consumption of both the algorithms in the Mushroom database, respectively, for different *minSup* and *maxPer* values. Figure 5 presents the performance of the two algorithms in three cases, minSup = 6% and maxPer = 3% (Figure 5(m)), minSup = 7% and maxPer = 3% (Figure 5(n)), minSup = 8% and maxPer = 3% (Figure 5(o)). In these cases, SPP-ECLAT consumes less than 58MB on average.

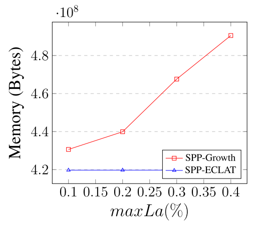
It can be observed that the SPP-ECLAT consumes less memory than the SPP-Growth algorithm. The following are some noteworthy findings that can be derived from this figure: (*i*) If we increase the *maxLa* value, then subsequently, both algorithms' memory consumption increase. The primary reason for this observation is that both the algorithms will discover many SPIs in any database if the *maxLa* value continues to increase. (*ii*) SPP-ECLAT generates SPIs using a SPP-list structure, which helps reduce the search space on every database. (*iii*) With the BMS-WebView-2 dataset, the processing time of the proposed algorithm is comparable to the SPP-Growth algorithm; however, it is interesting to note that the SPP-ECLAT algorithm requires much less memory than the SPP-Growth algorithm.

## C. EXPERIMENT 2: EVALUATION OF SPP-GROWTH AND SPP-ECLAT ALGORITHMS BY VARYING minSup AND maxPer

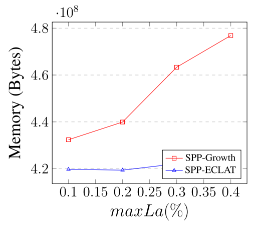
In the previous experiment, we have evaluated the performance of the SPP-Growth and SPP-ECLAT algorithms by varying *minSup* and *maxLa* values. In this experiment,



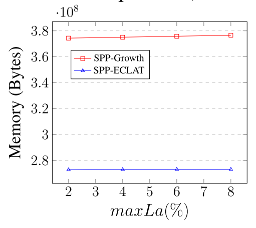
(a) **T100I4D100K**
$minSup$=0.5%,$maxPer$=0.4%



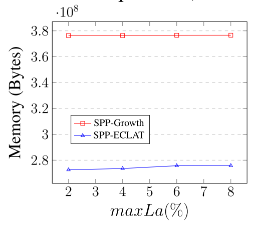
(b) **T100I4D100K**
$minSup$=0.8%,$maxPer$=0.4%



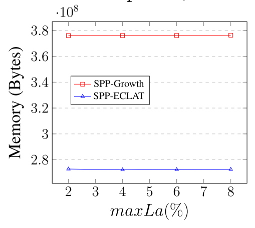
(c) **T100I4D100K**
$minSup$=1%,$maxPer$=0.4%



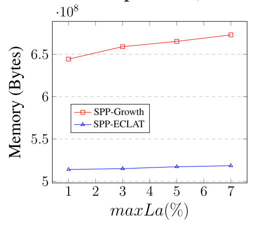
(d) **Retail**
$minSup$=0.8%, $maxPer$=2%



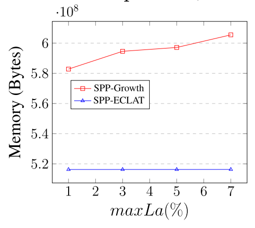
(e) **Retail**
$minSup$=0.9%, $maxPer$=2%



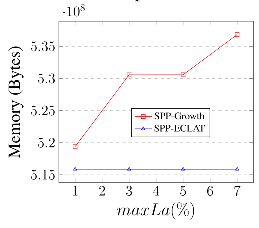
(f) **Retail**
$minSup$=1%,$maxPer$=2%



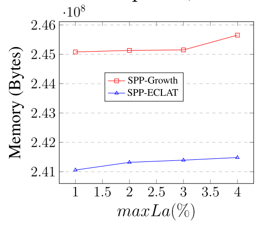
(g) **T20I6D100K**
$minSup$=3%, $maxPer$=4%



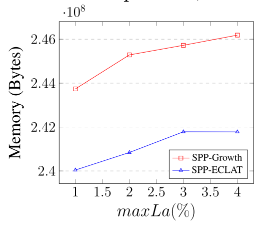
(h) **T20I6D100K**
$minSup$=3.5%, $maxPer$=4%



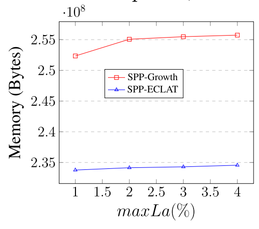
(i) **T20I6D100K**
$minSup$=4%, $maxPer$=4%



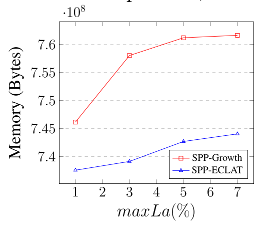
(j) **BMS-WebView-1**
$minSup$=0.5%,$maxPer$=5%



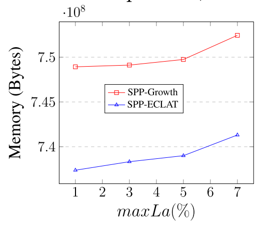
(k) **BMS-WebView-1**
$minSup$=0.8%,$maxPer$=5%



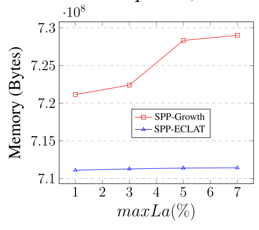
(l) **BMS-WebView-1**
$minSup$=1%,$maxPer$=5%



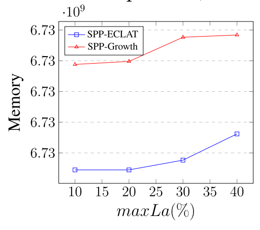
(m) **BMS-WebView-2**
$minSup$=0.6%,$maxPer$=5%



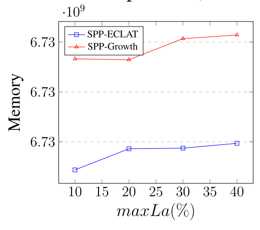
(n) **BMS-WebView-2**
$minSup$=0.8%,$maxPer$=5%



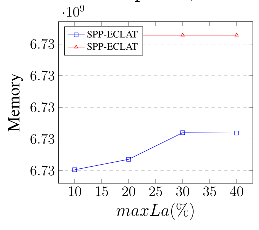
(o) **BMS-WebView-2**
$minSup$=1%,$maxPer$=5%



(p) **Mushrooms**
$minSup$=6%, $maxPer$=3%



(q) **Mushrooms**
$minSup$=7%, $maxPer$=3%



(r) **Mushrooms**
$minSup$=8%, $maxPer$=3%

**FIGURE 5. Memory consumption of SPP-Growth and SPP-ECLAT algorithms at different *maxLa*.**

we study the impact of *minSup* and *maxLa* constraints on the number of itemsets generated, the runtime requirements of SPP-Growth and SPP-ECLAT algorithms, and the memory consumed by SPP-Growth and SPP-ECLAT algorithms. Please note that the *maxLa* constraint has been fixed at a particular value for each database throughout this experimentation.

First, the number of SPIs generated by SPP-Growth and SPP-ECLAT algorithms is shown in Figure 6 by varying the value of *maxPer*. In detail, Figure 6(a) to Figure 6(c), shows the number of SPIs generated by the two algorithms in the T10I4D100K database, respectively, for different *minSup* and *maxLa* values. From Figure 6(d) to Figure 6(f), shows the number of SPIs generated by the two algorithms in the Retail database, respectively, for different *minSup* and *maxLa* values. From Figure 6(g) to Figure 6(i), shows the number of SPIs generated by the two algorithms in the T20I6D100K database, respectively, for different *minSup* and *maxLa* values. From Figure 6(j) to Figure 6(l), shows the number of SPIs generated by the two algorithms in the BMS-WebView-1 database, respectively,for different *minSup* and *maxLa* values. From Figure 6(m) to Figure 6(o), shows the number of SPIs generated by the two algorithms in the BMS-WebView-2 database, respectively,for different *minSup* and *maxLa* values. From Figure 6(p) to Figure 6(r), shows the number of SPIs generated by the two algorithms in the Mushrooms database respectively,for different *minSup* and *maxLa* values. It is to be noted that both algorithms will generate an equal number of itemsets. Therefore, both the curves were overlapped throughout the figures. The following two observations can be drawn from these figures: (*i*) The *maxPer* constraint has positive effect on the generation of SPIs. That is, increase in *maxPer* increases the number of SPIs, and vice-versa. It is because, if we increase the value of the *maxPer*, then most of the non-periodic itemsets have become periodic with an increase in the maximum inter-arrival time duration. (*ii*) The *minSup* constraint has negative effect on the generation of SPIs. That is, increase in *minSup* decreases the number of SPIs, and vice-versa. It is because many itemsets fail to satisfy the increased *minSup* value.

Next, the runtime requirements of SPP-Growth and SPP-ECLAT algorithms is shown in Figure 7 by varying the value of *maxPer*. In detail, Figure 7(a) to Figure 7(c), shows the runtime requirements of the two algorithms in the T10I4D100K database, respectively, for different *minSup* and *maxLa* values. Figure 7 presents the performance comparison of the two algorithms in three cases, minSup = 0.5% and maxLa = 0.4% (Figure 7(a)), minSup = 0.8% and maxLa = 0.4% (Figure 7(b)), minSup = 1% and maxLa = 0.4% (Figure 4(c)). The results in these three cases all show that SPP-Growth requires more time than SPP-ECLAT.

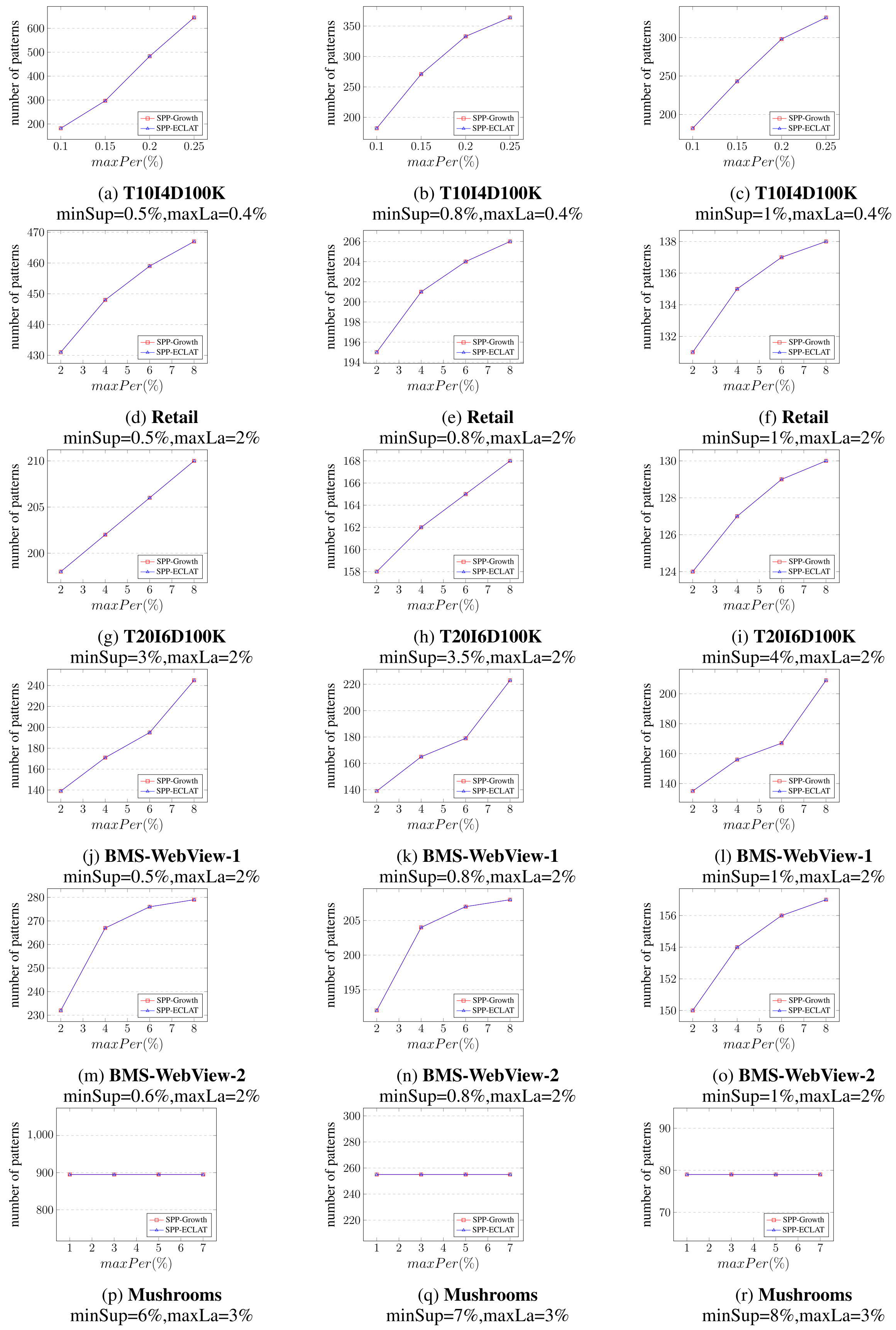
From Figure 7(d) to Figure 7(f), shows the runtime requirements of the two algorithms in the Retail database, respectively, for different *minSup* and *maxLa* values.Figure 7 presents the performance comparison of the two algorithms in three cases, minSup = 0.5% and maxLa = 2% (Figure 7(d)), minSup = 0.8% and maxLa = 2% (Figure 7(e)), minSup = 1% and maxLa = 2% (Figure 4(f)). The results in these three cases all show that SPP-Growth requires more time than SPP-ECLAT. From Figure 7(g) to Figure 7(i), shows the runtime requirements of the two algorithms in the T20I6D100K database, respectively, for different *minSup* and *maxLa* values.Figure 7 presents the performance comparison of the two algorithms in three cases, minSup = 3% and maxLa = 2% (Figure 7(g)), minSup = 3.5% and maxLa = 2% (Figure 7(h)), minSup = 4% and maxLa = 4% (Figure 4(i)). The results in these three cases all show that SPP-Growth requires more time than SPP-ECLAT.

From Figure 7(j) to Figure 7 (l), shows the runtime requirements of the two algorithms in the BMS-WebView-1 database, respectively,for different *minSup* and *maxLa* values. Figure 7 presents the performance comparison of the two algorithms in three cases, minSup = 0.5% and maxLa = 2% (Figure 7(j)), minSup = 0.8% and maxLa = 2% (Figure 7(k)), minSup = 1% and maxLa = 2% (Figure 4(l)). The results in these three cases all show that SPP-Growth requires more time than SPP-ECLAT.

From Figure 7(m) to Figure 7(o), shows the runtime requirements of the two algorithms in the BMS-WebView-2 database, respectively,for different *minSup* and *maxLa* values. Figure 7 presents the performance comparison of the two algorithms in three cases, minSup = 0.6% and maxLa = 2% (Figure 7(m)), minSup = 0.8% and maxLa = 2% (Figure 7(n)), minSup = 1% and maxLa = 2% (Figure 4(o)). The results in these three cases all show that SPP-Growth requires less time than SPP-ECLAT.

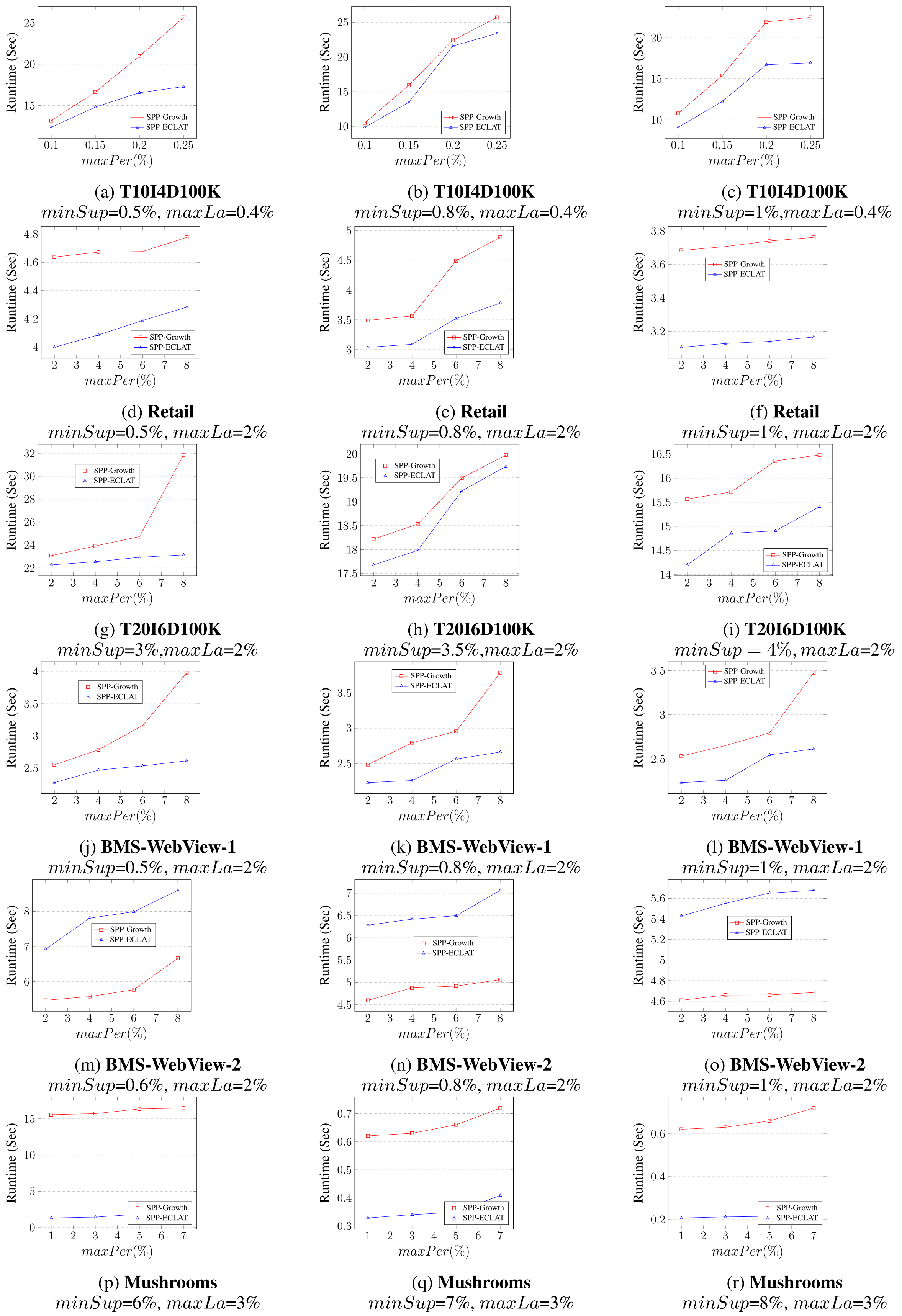
From Figure 7(q) to Figure 7(r), shows the runtime requirements of the two algorithms in the Mushrooms database respectively,for different *minSup* and *maxLa* values. Figure 7 presents the performance comparison of the two algorithms in three cases, minSup = 6% and maxLa = 3% (Figure 7(p)), minSup = 7% and maxLa = 3% (Figure 7(q)), minSup = 8% and maxLa = 3% (Figure 4(r)). The results in these three cases show that SPP-Growth requires more time than SPP-ECLAT.

It can be observed that the SPP-ECLAT runs faster than the SPP-Growth algorithm in most case. The good performance of SPP-ECLAT is a result of the effectiveness of periodic calculation and pruning techniques. The following are some noteworthy findings that can be derived from this figure: (*i*) If we increase the *maxPer* value, then subsequently, both algorithms' runtime requirements increase. The primary reason for this observation is that both the algorithms will discover many SPIs in any database if the *maxPer* value continues to increase. (*ii*) SPI-ECLAT generates SPIs much faster than SPP-Growth under any given *maxPer* in BMS-WebView-1, Retail, T10I4D100K, T20I6D100K, and Mushrooms databases. More importantly, we can also observe that at high *maxPer* values, SPP-ECLAT algorithm generates the SPIs much faster than SPP-Growth algorithm. The reason is SPP-ECLAT using the downward closure



(a) **T10I4D100K** minSup=0.5%,maxLa=0.4%
(b) **T10I4D100K** minSup=0.8%,maxLa=0.4%
(c) **T10I4D100K** minSup=1%,maxLa=0.4%
(d) **Retail** minSup=0.5%,maxLa=2%
(e) **Retail** minSup=0.8%,maxLa=2%
(f) **Retail** minSup=1%,maxLa=2%
(g) **T20I6D100K** minSup=3%,maxLa=2%
(h) **T20I6D100K** minSup=3.5%,maxLa=2%
(i) **T20I6D100K** minSup=4%,maxLa=2%
(j) **BMS-WebView-1** minSup=0.5%,maxLa=2%
(k) **BMS-WebView-1** minSup=0.8%,maxLa=2%
(l) **BMS-WebView-1** minSup=1%,maxLa=2%
(m) **BMS-WebView-2** minSup=0.6%,maxLa=2%
(n) **BMS-WebView-2** minSup=0.8%,maxLa=2%
(o) **BMS-WebView-2** minSup=1%,maxLa=2%
(p) **Mushrooms** minSup=6%,maxLa=3%
(q) **Mushrooms** minSup=7%,maxLa=3%
(r) **Mushrooms** minSup=8%,maxLa=3%

**FIGURE 6. Number of stable periodic-frequent itemsets generated in various databases by varying *minSup* and *maxPer*.**



(a) **T10I4D100K** $minSup$=0.5%, $maxLa$=0.4%

(b) **T10I4D100K** $minSup$=0.8%, $maxLa$=0.4%

(c) **T10I4D100K** $minSup$=1%,$maxLa$=0.4%

(d) **Retail** $minSup$=0.5%, $maxLa$=2%

(e) **Retail** $minSup$=0.8%, $maxLa$=2%

(f) **Retail** $minSup$=1%, $maxLa$=2%

(g) **T20I6D100K** $minSup$=3%,$maxLa$=2%

(h) **T20I6D100K** $minSup$=3.5%,$maxLa$=2%

(i) **T20I6D100K** $minSup = 4\%$, $maxLa$=2%

(j) **BMS-WebView-1** $minSup$=0.5%, $maxLa$=2%

(k) **BMS-WebView-1** $minSup$=0.8%, $maxLa$=2%

(l) **BMS-WebView-1** $minSup$=1%, $maxLa$=2%

(m) **BMS-WebView-2** $minSup$=0.6%, $maxLa$=2%

(n) **BMS-WebView-2** $minSup$=0.8%, $maxLa$=2%

(o) **BMS-WebView-2** $minSup$=1%, $maxLa$=2%

(p) **Mushrooms** $minSup$=6%, $maxLa$=3%

(q) **Mushrooms** $minSup$=7%, $maxLa$=3%

(r) **Mushrooms** $minSup$=8%, $maxLa$=3%

**FIGURE 7. Runtime requirements of SPP-Growth and SPP-ECLAT algorithms at diffrent *maxPer*.**

property and the depth-first search technique, so the SPIs are generated by simply peforming intersection of SPP-list. The process is repeated to find all SPIs. (*iii*) With the BMS-WebView-2 dataset, which contains long transactions and many distinct items, SPP-ECLAT algorithm takes more time than SPP-Growth algorithm. It is because SPP-ECLAT algorithm is based on the downward closure property and the depth-first search technique, so it doesn't require scanning the database each time, but generating all SPIs it has to perform the intersection of the SPP-list. So the long SPP-List requires more time to repeat the intersection process.

Finally, we have shown the memory consumption details of SPP-Growth and SPP-ECLAT algorithms in Figure 8 by varying the value of *maxPer*. In detail, Figure 8(a) to Figure 8(c), shows the memory consumption of both the algorithms in the T10I4D100K database, respectively, for different *minSup* and *maxLa* values. Figure 8 presents the performance of the two algorithms in three cases, minSup = 0.5% and maxLa = 0.4% (Figure 8(a)), minSup = 0.8% and maxLa = 0.4% (Figure 8(b)), minSup = 1% and maxLa = 0.4% (Figure 8(c)). The results all show that SPP-ECLAT consumes less memory than SPP-Growth. In detail, the average memory consumption difference between the two algorithms is around 120MB. The more maxLa value increase, the more significant the difference memory consumption between the two algorithms, from 80MB (when maxLa value is 0.1%) to 170MB (when maxLa value is 0.4%).

From Figure 8(d) to Figure 8(f), shows the memory consumption of both the algorithms in the Retail database, respectively, for different *minSup* and *maxLa* values. Figure 8 presents the performance of the two algorithms in three cases, minSup = 0.5% and maxLa = 2% (Figure 8(d)), minSup = 0.8% and maxLa = 2% (Figure 8(e)), minSup = 1% and maxLa = 2% (Figure 8(f)). The results all show that SPP-ECLAT consumes less memory than SPP-Growth. The magnitude of the difference in memory consumption is around 70MB.

From Figure 8(g) to Figure 8(i), shows the memory consumption of both the algorithms in the T20I6D100K database, respectively, for different *minSup* and *maxLa* values. Figure 8 presents the performance of the two algorithms in three cases, minSup = 3% and maxLa = 2% (Figure 8(g)), minSup = 3.5% and maxLa = 2% (Figure 8(h)), minSup = 4% and maxLa = 2% (Figure 8(i)). in these cases, SPP-ECLAT consumes less memory than SPP-Growth by 140MB on average.

From Figure 8(j) to Figure 8(l), shows the memory consumption of both the algorithms in the BMS-WebView-1 database, respectively, for different *minSup* and *maxLa* values. Figure 8 presents the performance of the two algorithms in three cases, minSup = 0.5% and maxLa = 2% (Figure 8(j)), minSup = 0.8% and maxLa = 2% (Figure 8(k)), minSup = 1% and maxLa = 2% (Figure 8(l)). The results show that SPP-ECLAT consumes less memory than SPP-Growth, around 8MB on average.

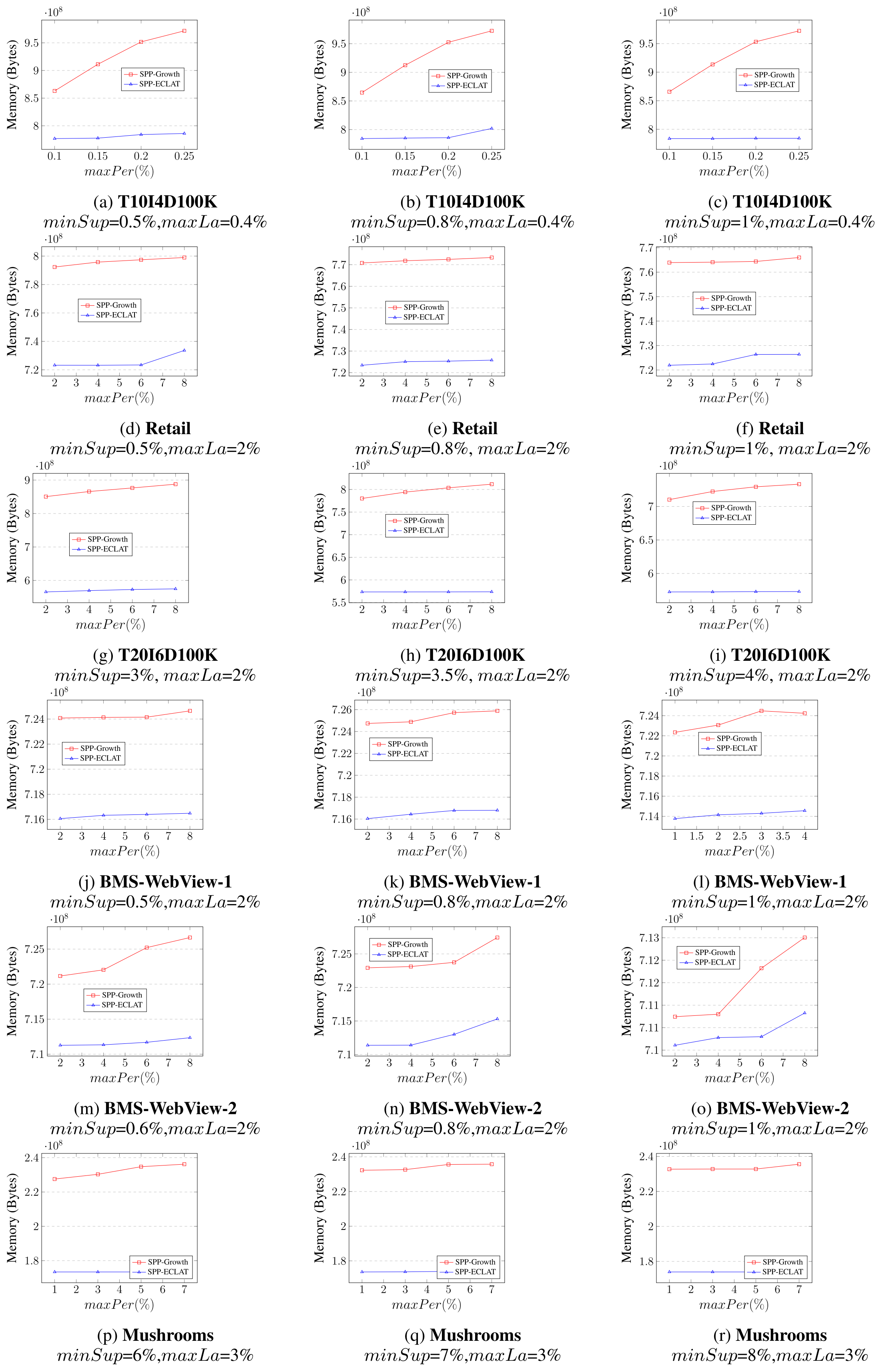
From Figure 8(m) to Figure 8(o), shows the memory consumption of both the algorithms in the BMS-WebView-2 database, respectively, for different *minSup* and *maxLa* values. Figure 8 presents the performance of the two algorithms in three cases, minSup = 0.6% and maxLa = 2% (Figure 8(m)), minSup = 0.8% and maxLa = 2% (Figure 8(n)), minSup = 1% and maxLa = 2% (Figure 8(o)). The results all show that SPP-ECLAT consumes less memory than SPP-Growth. In detail, the difference in memory consumption when maxLa = 2% is around 9MB, and when maxLa is increasing up to 4%, the difference in memory consumption of the two algorithms is around 10MB

From Figure 8(p) to Figure 8(r), shows the memory consumption of both the algorithms in the Mushrooms database, respectively, for different *minSup* and *maxLa* values.Figure 8 presents the performance of the two algorithms in three cases, minSup = 6% and maxLa = 3% (Figure 8(p)), minSup = 7% and maxLa = 3% (Figure 5(q)), minSup = 8% and maxLa = 3% (Figure 5(r)). In these cases, SPP-ECLAT consumes less than 56MB on average.

It can be observed that the SPP-ECLAT consumes relatively less memory than the SPP-Growth algorithm. The following are some noteworthy findings that can be derived from this figure: (*i*) If we increase the *maxPer* value, then subsequently, both algorithms' memory consumption increase. The primary reason for this observation is that both the algorithms will discover many SPIs in any database if the *maxPer* value continues to increase. (*ii*) SPP-ECLAT generates SPIs using a SPP-list structure, which helps reduce the search space on every database. (*iii*) It should highlight that in BMS-WebView-2 dataset. The processing time of the proposed algorithm for this dataset is comparable to the SPP-Growth algorithm; however, it is interesting to note that SPP-ECLAT algorithm requires much less memory in the memory consumption test than the SPP-Growth algorithm.

## D. EXPERIMENT 3: SCALABILITY OF THE SPP-GROWTH AND SPP-ECLAT ALGORITHMS

In this experiment, we have used the Kosarak database, a sparse real-world database, to perform the scalability operation. The scalability operation depends on the number of items and the number of records (e.g., transactions). Therefore, to analyze the complexity, we need to think about every operation an algorithm performs and how it is affected by the number of items and records. Thus, the sparse dataset has been chosen because when we divide the database into equal portions, each portion has a different number of items, so we will see clearly how the algorithm is doing. This scalability operation is utilized to discover the efficacy and productivity of the proposed algorithm on big columnar temporal databases. Therefore, in this experiment we divide the Kosarak database into five equal portions, each with 0.2 million transactions. We evaluate the performance of both the SPP-Growth and SPP-ECLAT algorithms, where the database size is varied from 200000 to 1000000 transactions. Figure 9 shows the results in terms



(a) **T10I4D100K** $minSup$=0.5%,$maxLa$=0.4%

(b) **T10I4D100K** $minSup$=0.8%,$maxLa$=0.4%

(c) **T10I4D100K** $minSup$=1%,$maxLa$=0.4%

(d) **Retail** $minSup$=0.5%,$maxLa$=2%

(e) **Retail** $minSup$=0.8%, $maxLa$=2%

(f) **Retail** $minSup$=1%, $maxLa$=2%

(g) **T20I6D100K** $minSup$=3%, $maxLa$=2%

(h) **T20I6D100K** $minSup$=3.5%, $maxLa$=2%

(i) **T20I6D100K** $minSup$=4%, $maxLa$=2%

(j) **BMS-WebView-1** $minSup$=0.5%,$maxLa$=2%

(k) **BMS-WebView-1** $minSup$=0.8%,$maxLa$=2%

(l) **BMS-WebView-1** $minSup$=1%,$maxLa$=2%

(m) **BMS-WebView-2** $minSup$=0.6%,$maxLa$=2%

(n) **BMS-WebView-2** $minSup$=0.8%,$maxLa$=2%

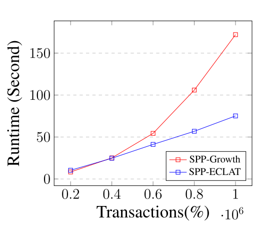
(o) **BMS-WebView-2** $minSup$=1%,$maxLa$=2%

(p) **Mushrooms** $minSup$=6%,$maxLa$=3%

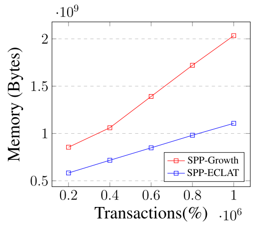
(q) **Mushrooms** $minSup$=7%,$maxLa$=3%

(r) **Mushrooms** $minSup$=8%,$maxLa$=3%

**FIGURE 8. Memory consumption of SPP-Growth and SPP-ECLAT algorithms at different *maxPer*.**



(a) Runtime



(b) Memory

**FIGURE 9. Scalability of the SPP-Growth and SPP-ECLAT algorithms.**

of the runtimes and memory consumption levels of both SPP-Growth and SPP-ECLAT algorithms under different database sizes when $maxLa = 0.04$ (in %), $minSup = 0.01$ (in %), and $maxPer = 0.05$ (in %). Some of the important observations that can be drawn from this figure are as follows. ($i$) If we keep increasing the database size, then both algorithms' runtimes and memory requirements will increase almost linearly. ($ii$) SPP-ECLAT consumes less runtime and memory than the SPP-Growth algorithm under any given database size.

## VI. DISCUSSION SECTION

In this section, we have compared the time complexity analysis of both the algorithms. Let us consider a columnar temporal database containing '$p$' number of distinct items and total number of transactions represented as '$q$'. Let us assume that all the items in $q$ are interesting, and every item is present in every transaction; i.e., the generated lists contain $q$ entries each.

In the literature, SPP-Growth [26] is the only state-of-the-art algorithm that uses the concept of SPP-list constructions to generate complete SPIs. The major contributions of SPP-Growth and its complexities are as follows: First, the complete database is scanned, and the items in each transaction are stored as a prefix-tree. In the worst case, if every item is present in every transaction, then the time complexity for this operation is $O(p*q)$. Second, we construct the SPP-lists with a complexity of $O(p*q)$. After the initial prefix-tree construction, SPP-Growth recursively performs a depth-first search to find all the interesting itemsets. The number of possible itemsets is $n = 2^p - 1$. In real-world applications, the number of itemsets considered depends on the database's characteristics and the algorithms' parameters. If $minSup$, $maxPer$, or $maxLa$ are increased, fewer itemsets may be considered due to applying the search space pruning strategies. Finally, for each considered itemset $\Gamma$ that extends an itemset $\Delta$, SPP-Growth traverses the node-links of the SPP-list of $\Delta$ to create the conditional pattern base, SPP-list, and prefix-tree of $\Gamma$. This construction is done in linear time as these structures of $\Delta$ are traversed once. Therefore, the overall complexity of SPP-Growth is $O(p*q)+O(p*q*n) = O(p*q*n)$.

We complete the generation of SPIs using the SPP-ECLAT with the help of two algorithms. In Algorithm 1, we scan the complete database once to discover the one-length SPIs by constructing an SPP-list data structure. In the worst case, if every item is present in every transaction, then the time complexity for this operation is $O(p*q)$. In the Algorithm 2, we need to merge the TS-list elements of the two current length itemsets to generate the higher length itemsets. In the worst case, if every item is present in every transaction as the length of the TS-list of every itemset becomes $q$, then the time complexity for merging any of the two itemset's TS-lists becomes $O(q)$. This algorithm utilizes the Depth-First Search (DFS) strategy on the itemset lattice. The number of possible itemsets is $n = 2^p - 1$. Therefore, the time complexity for generating all the possible interesting itemsets is $O(n*q)$. The overall time complexity of SPP-ECLAT is $O(q+n*q) = O(q*n)$.

In real-world applications, the overall superiority of SPP-ECLAT ultimately depends on the actual values of the given parameters, such as $p$, $q$, and $n$. Therefore, we conducted rigorous experimentation on six real-world databases to demonstrate that SPP-ECLAT outperforms the state-of-the-art SPP-Growth algorithm.

## VII. CONCLUSION AND FUTURE WORK

This study has proposed an efficient and novel algorithm, called Stable Periodic-frequent Pattern – Equivalence Class Transformation(SPP-ECLAT), to discover stable periodic-frequent itemsets. The output itemsets of the algorithm not only satisfy the user-specified *minimum support* and *maximum periodicity* thresholds but also are stable itemsets based on the user-specified *maximum lability* threshold in any big columnar temporal databases. The SPP-List structure of the SPP-ECLAT algorithm plays an important role in eliminating many itemsets that are not considered to be candidate itemsets from the huge search space. An in-depth examination of the proposed SPP-ECLAT approach on six synthetic and real-world databases revealed that its memory consumption and runtime are efficient and highly scalable relative to those of the state-of-the-art SPP-Growth algorithm.

As for the future work, we will study the Lability concept over different types of itemsets. It is also interesting to work on discovering stable periodic-frequent itemsets in uncertain databases. Furthermore, we will focus on identifying SPIs in static temporal data, and it would be important to investigate stable itemsets in graphs, data streams, and symbolic databases in the future.

## REFERENCES

[1] MySQL. *Mysql*. Accessed: Apr. 15, 2022. [Online]. Available: https://www.mysql.com/
[2] PostGres. *Postgres*. Accessed: Apr. 15, 2022. [Online]. Available: https://www.postgresql.org/
[3] BigQuery. *Bigquery*. Accessed: Apr. 22, 2022. [Online]. Available: https://cloud.google.com/bigquery
[4] L. George, *HBase: The Definitive Guide: Random Access to Your Planet-Size Data*. Sebastopol, CA, USA: O'Reilly Media, 2011.
[5] *Snowflake*. Accessed: Apr. 22, 2022. [Online]. Available: https://www.snowflake.com/

[6] R. Agrawal, T. Imieliński, and A. Swami, "Mining association rules between sets of items in large databases," in *Proc. ACM SIGMOD Int. Conf. Manage. Data (SIGMOD)*, 1993, pp. 207–216.

[7] J. M. Luna, P. Fournier-Viger, and S. Ventura, "Frequent itemset mining: A 25 years review," *WIREs Data Mining Knowl. Discovery*, vol. 9, no. 6, pp. 2–8, Nov. 2019.

[8] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules," in *Proc. 20th Int. Conf. Very Large Data Bases*, vol. 1215, 1994, pp. 487–499.

[9] J. Han, J. Pei, and Y. Yin, "Mining frequent patterns without candidate generation," *ACM SIGMOD Rec.*, vol. 29, no. 2, pp. 1–12, 2000.

[10] M. Yasir, M. A. Habib, M. Ashraf, S. Sarwar, M. U. Chaudhry, H. Shahwani, M. Ahmad, and C. M. N. Faisal, "D-GENE: Deferring the generation of power sets for discovering frequent itemsets in sparse big data," *IEEE Access*, vol. 8, pp. 27375–27392, 2020.

[11] M. Yasir, M. A. Habib, M. Ashraf, S. Sarwar, M. U. Chaudhry, H. Shahwani, M. Ahmad, and C. M. N. Faisal, "TRICE: Mining frequent itemsets by iterative trimmed transaction LattICE in sparse big data," *IEEE Access*, vol. 7, pp. 181688–181705, 2019.

[12] M. Yasir, M. A. Habib, S. Sarwar, C. M. N. Faisal, M. Ahmad, and S. Jabbar, "HARPP: HARnessing the power of power sets for mining frequent itemsets," *Inf. Technol. Control*, vol. 48, no. 3, pp. 415–431, Sep. 2019, doi: 10.5755/j01.itc.48.3.21137.

[13] A. Savasere, E. R. Omiecinski, and S. B. Navathe, "An efficient algorithm for mining association rules in large databases," Georgia Inst. Technol., Atlanta, GA, USA, Tech. Rep., 1995.

[14] J. Han, J. Pei, and Y. Yin, "Mining frequent patterns without candidate generation," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, May 2000, pp. 1–12.

[15] J. Pei, J. Han, H. Lu, S. Nishio, S. Tang, and D. Yang, "H-mine: Fast and space-preserving frequent pattern mining in large databases," *IIE Trans.*, vol. 39, no. 6, pp. 593–605, Mar. 2007.

[16] G. Grahne and J. Zhu, "High performance mining of maximal frequent itemsets," in *Proc. 6th Int. Workshop High Perform. Data Mining*, vol. 16, 2003, p. 34.

[17] M. J. Zaki, S. Parthasarathy, M. Ogihara, and W. Li, "New algorithms for fast discovery of association rules," in *Proc. KDD*, vol. 97, 1997, pp. 283–286.

[18] M. J. Zaki and C.-J. Hsiao, "CHARM: An efficient algorithm for closed itemset mining," in *Proc. SIAM Int. Conf. Data Mining*, Apr. 2002, pp. 457–473.

[19] S. K. Tanbeer, C. F. Ahmed, B.-S. Jeong, and Y.-K. Lee, "Discovering periodic-frequent patterns in transactional databases," in *Proc. PAKDD*, 2009, pp. 242–253.

[20] R. U. Kiran, C. Saideep, P. Ravikumar, K. Zettsu, M. Toyoda, M. Kitsuregawa, and P. K. Reddy, "Discovering fuzzy periodic-frequent patterns in quantitative temporal databases," in *Proc. IEEE Int. Conf. Fuzzy Syst. (FUZZ-IEEE)*, Jul. 2020, pp. 1–8.

[21] R. U. Kiran and P. K. Reddy, "Mining rare periodic-frequent patterns using multiple minimum supports," in *Proc. 15th Int. Conf. Manage. Data*, 2009, pp. 7–8.

[22] J. Han, G. Dong, and Y. Yin, "Efficient mining of partial periodic patterns in time series database," in *Proc. 15th Int. Conf. Data Eng.*, Mar. 1999, pp. 106–115.

[23] 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, p. 1523, May 2022. [Online]. Available: https://www.mdpi.com/2079-9292/11/10/1523

[24] P. Fournier-Viger, J. C. Lin, Q. Duong, and T. Dam, "PHM: Mining periodic high-utility itemsets," in *Proc. Ind. Conf. Data Mining*, 2016, pp. 64–79.

[25] R. U. Kiran, M. Kitsuregawa, and P. K. Reddy, "Efficient discovery of periodic-frequent patterns in very large databases," *J. Syst. Softw.*, vol. 112, pp. 110–121, Feb. 2016.

[26] P. Fournier-Viger, P. Yang, J. C. Lin, and R. U. Kiran, "Discovering stable periodic-frequent patterns in transactional data," in *Proc. Int. Conf. Ind., Eng. Appl. Appl. Intell. Syst.*, 2019, pp. 230–244.

[27] M. J. Zaki, "Scalable algorithms for association mining," *IEEE Trans. Knowl. Data Eng.*, vol. 12, no. 3, pp. 372–390, May/Jun. 2000.

[28] H. N. Dao, P. Ravikumar, P. Likitha, B. V. V. Raj, R. U. Kiran, Y. Watanobe, and I. Paik, "Towards efficient discovery of stable periodic patterns in big columnar temporal databases," in *Proc. Int. Conf. Ind., Eng. Appl. Appl. Intell. Syst. (IEA/AIE)*, 2022, pp. 831–843.

[29] S. Brin, R. Motwani, J. D. Ullman, and S. Tsur, "Dynamic itemset counting and implication rules for market basket data," in *Proc. ACM SIGMOD Int. Conf. Manage. Data (SIGMOD)*, 1997, pp. 255–264.

[30] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A. Verkamo, "Fast discovery of association rules," *Adv. Knowl. Discovery Data Mining*, vol. 12, no. 1, pp. 307–328, 1996.

[31] M. J. Zaki, "Scalable algorithms for association mining," *IEEE Trans. Knowl. Data Eng.*, vol. 12, no. 3, pp. 372–390, May/Jun. 2000.

[32] R. Agrawal and R. Srikant, "Mining sequential patterns," in *Proc. 11th Int. Conf. Data Eng.*, 1995, pp. 3–14.

[33] M. L. Hetland, "A survey of recent methods for efficient retrieval of similar time sequences," in *Data Mining in Time Series Databases*. Singapore: World Scientific, 2004, pp. 23–42.

[34] C.-F. Huang, Y.-C. Chen, and A.-P. Chen, "An association mining method for time series and its application in the stock prices of TFT-LCD industry," in *Proc. Ind. Conf. Data Mining*. Germany: Springer, 2004, pp. 117–126.

[35] C. F. Ahmed, S. K. Tanbeer, B. S. Jeong, and Y. K. Lee, "Efficient tree structures for high utility pattern mining in incremental databases," *IEEE Trans. Knowl. Data Eng.*, vol. 21, no. 12, pp. 1708–1721, Dec. 2009.

[36] M. Liu and J. Qu, "Mining high utility itemsets without candidate generation," in *Proc. 21st ACM Int. Conf. Inf. Knowl. Manage.*, Oct. 2012, pp. 55–64.

[37] R. Chan, Q. Yang, and Y.-D. Shen, "Mining high utility itemsets," in *Proc. 3rd IEEE Int. Conf. Data Mining*, Nov. 2003, p. 19.

[38] W. Wang, C. Wang, Y. Zhu, B. Shi, J. Pei, X. Yan, and J. Han, "GraphMiner: A structural pattern-mining system for large disk-based graph databases and its applications," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, Jun. 2005, pp. 879–881.

[39] D. J. Cook and L. B. Holder, "Graph-based data mining," *IEEE Intell. Syst. Appl.*, vol. 15, no. 2, pp. 32–41, Mar. 2000.

[40] R. U. Kiran, J. Venkatesh, P. Fournier-Viger, M. Toyoda, P. K. Reddy, and M. Kitsuregawa, "Discovering periodic patterns in non-uniform temporal databases," in *Proc. Pacific-Asia Conf. Knowl. Discovery Data Mining*. Jeju, South Korea: Springer, 2017, pp. 604–617.

[41] D. Zhang, K. Lee, and I. Lee, "Hierarchical trajectory clustering for spatio-temporal periodic pattern mining," *Exp. Syst. Appl.*, vol. 92, pp. 1–11, Feb. 2018.

[42] R. U. Kiran, A. Anirudh, C. Saideep, M. Toyoda, P. K. Reddy, and M. Kitsuregawa, "Finding periodic-frequent patterns in temporal databases using periodic summaries," *Data Sci. Pattern Recognit.*, vol. 3, no. 2, pp. 24–46, 2019.

[43] R. Uday Kiran, Y. Watanobe, B. Chaudhury, K. Zettsu, M. Toyoda, and M. Kitsuregawa, "Discovering maximal periodic-frequent patterns in very large temporal databases," in *Proc. IEEE 7th Int. Conf. Data Sci. Adv. Analytics (DSAA)*, Oct. 2020, pp. 11–20.

[44] K. Amphawan, P. Lenca, and A. Surarerks, "Mining top-K periodic-frequent pattern from transactional databases without support threshold," in *Proc. Adv. Inf. Technol.* Hong Kong: Springer, 2009, pp. 18–29.

[45] R. U. Kiran and P. Krishna Reddy, "Towards efficient mining of periodic-frequent patterns in transactional databases," in *Proc. Int. Conf. Database Expert Syst. Appl.* Spain: Springer, 2010, pp. 194–208.

[46] R. U. Kiran and P. K. Reddy, "An alternative interestingness measure for mining periodic-frequent patterns," in *Proc. Int. Conf. Database Syst. Adv. Appl.* Hong Kong: Springer, 2011, pp. 183–192.

[47] P. Ravikumar, P. Likhitha, B. V. V. Raj, R. U. Kiran, Y. Watanobe, and K. Zettsu, "Efficient discovery of periodic-frequent patterns in columnar temporal databases," *Electronics*, vol. 10, no. 12, p. 1478, Jun. 2021.

[48] R. U. Kiran and M. Kitsuregawa, "Novel techniques to reduce search space in periodic-frequent pattern mining," in *Proc. Int. Conf. Database Syst. Adv. Appl.* Bali, Indonesia: Springer, 2014, pp. 377–391.

[49] A. Surana, R. U. Kiran, and P. K. Reddy, "An efficient approach to mine periodic-frequent patterns in transactional databases," in *Proc. Pacific-Asia Conf. Knowl. Discovery Data Mining*. Shenzhen, China: Springer, 2011, pp. 254–266.

[50] R. He, J. Chen, C. Du, and Y. Duan, "Stable periodic frequent itemset mining on uncertain datasets," in *Proc. IEEE 4th Int. Conf. Comput. Commun. Eng. Technol. (CCET)*, Aug. 2021, pp. 263–267.

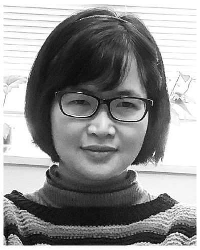
[51] P. Fournier-Viger, Y. Wang, P. Yang, J. C.-W. Lin, U. Yun, and R. U. Kiran, "TSPIN: Mining top-k stable periodic patterns," *Int. J. Speech Technol.*, vol. 52, no. 6, pp. 6917–6938, Apr. 2022.

[52] R. U. Kiran. *PAMI-PyKit: PAttern MIning-Python Kit*. Accessed: Jun. 4, 2020. [Online]. Available: https://github.com/udayRage/pamipykit/tree/master/traditional

[53] *KDD-cup-2000*. Accessed: Sep. 30, 2010. [Online]. Available: https://kdd.org/kdd-cup/view/kdd-cup-2000

[54] T. Brijs, "Retail market basket data set," in *Proc. Workshop Frequent Itemset Mining Implementations (FIMI)*, 2003, pp. 8–15.

[55] P. Fournier-Viger, J. C.-W. Lin, A. Gomariz, T. Gueniche, A. Soltani, Z. Deng, and H. T. Lam, "The SPMF open-source data mining library version 2," in *Machine Learning and Knowledge Discovery in Databases*, B. Berendt, B. Bringmann, É. Fromont, G. Garriga, P. Miettinen, N. Tatti, and V. Tresp, Eds. Cham, Switzerland: Springer, 2016, pp. 36–40.



**HONG N. DAO** received the M.B.A. degree from Chung-Nam National University, South Korea. She is currently pursuing the Ph.D. degree with The University of Aizu. Her research interests include data analytics for economics and big data of smart city.

**PENUGONDA RAVIKUMAR** received the M.E. degree in computer science from the Indian Institute of Science, Bangalore, Karnataka, India. He is currently pursuing the Ph.D. degree in computer and information systems with The University of Aizu, Aizuwakamatsu, Fukushima, Japan, on a deputation basis. He is an Assistant Professor of computer science and engineering with IIIT–RK Valley, Rajiv Gandhi University of Knowledge Technologies, Andhra Pradesh, India. His current research interests include data mining, air pollution data analytics, traffic congestion data analytics, recommender systems, and time series classification. He has published several papers in reputed international conferences, such as IEEE FUZZ, IEEE Big Data, IEEE DSAA, IEEE SSCI-CIDM, IEA/AIE, ISCMI, DEXA, and ACIIDS.

**PALLA LIKHITHA** is currently pursuing the master's degree in computer science and engineering with The University of Aizu, Japan. She has published papers in IEEE BIG DATA 2020 and 2021, ICONIP 2021, and IEA/AIE.

**RAGE UDAY KIRAN** (Member, IEEE) received the Ph.D. degree in computer science from the International Institute of Information Technology, Hyderabad, Telangana, India. He is currently working as an Associate Professor with The University of Aizu, Aizuwakamatsu, Fukushima, Japan. He also works as a Researcher with The University of Tokyo, Tokyo, Japan. He has published over 80 papers in refereed journals and international conferences, such as EDBT, CIKM, IEEE FUZZ, PAKDD, SIGSPATIAL, IEEE BIGDATA, DEXA, and DASFAA. His current research interests include data mining, parallel computation, air pollution data analytics, traffic congestion data analytics, recommender systems, and ICTs for Agriculture.

**YUTAKA WATANOBE** (Member, IEEE) is currently a Senior Associate Professor with the School of Computer Science and Engineering, The University of Aizu, Japan. His research interests include visual programming language, data mining, and cloud robotics.

**INCHEON PAIK** (Senior Member, IEEE) received the M.E. and Ph.D. degrees in electronic engineering from Korea University, in 1987 and 1992, respectively. He is currently a Professor with The University of Aizu, Japan. His research interests include semantic web, web services and their composition, data mining, deep learning, and big data science infrastructure. He is a member of ACM, IEICE, and IPSJ.