electronics

MDPI

Article

Efficient Discovery of Periodic-Frequent Patterns in Columnar Temporal Databases

Penugonda Ravikumar [1,2,†], Palla Likhitha [2,†], Bathala Venus Vikranth Raj [2,†], Rage Uday Kiran [1,3,*,†], Yutaka Watanobe [1] and Koji Zettsu [3]

[1] Department of Computer and Information Systems, University of AIZU, Aizuwakamatsu 965-8580, Japan; raviua138@gmail.com (P.R.); yutaka@u-aizu.ac.jp (Y.W.)
[2] IIIT-RK Valley, RGUKT-Andhara Pradesh, Andhra Pradesh 521202, India; likhithapalla7@gmail.com (P.L.); venusvikranthraj12@gmail.com (B.V.V.R.)
[3] NICT, Yokosuka-shi 239-0847, Japan; zettsu@nict.go.jp
* Correspondence: udayrage@u-aizu.ac.jp; Tel.: +81-242-37-2500
† These authors have equally contributed to 90% of this work. Remaining authors have equally contributed to 10% of this work.

Abstract: Discovering periodic-frequent patterns in temporal databases is a challenging problem of great importance in many real-world applications. Though several algorithms were described in the literature to tackle the problem of periodic-frequent pattern mining, most of these algorithms use the traditional horizontal (or row) database layout, that is, either they need to scan the database several times or do not allow asynchronous computation of periodic-frequent patterns. As a result, this kind of database layout makes the algorithms for discovering periodic-frequent patterns both time and memory inefficient. One cannot ignore the importance of mining the data stored in a vertical (or columnar) database layout. It is because real-world big data is widely stored in columnar database layout. With this motivation, this paper proposes an efficient algorithm, Periodic Frequent-Equivalence CLass Transformation (PF-ECLAT), to find periodic-frequent patterns in a columnar temporal database. Experimental results on sparse and dense real-world and synthetic databases demonstrate that PF-ECLAT is memory and runtime efficient and highly scalable. Finally, we demonstrate the usefulness of PF-ECLAT with two case studies. In the first case study, we have employed our algorithm to identify the geographical areas in which people were periodically exposed to harmful levels of air pollution in Japan. In the second case study, we have utilized our algorithm to discover the set of road segments in which congestion was regularly observed in a transportation network.

Keywords: data mining; pattern mining; periodic-frequent patterns; columnar databases

Citation: Ravikumar, P.; Likhitha, P.; Venus Vikranth Raj, B.; Uday Kiran, R.; Watanobe, Y.; Zettsu, K. Efficient Discovery of Periodic-Frequent Patterns in Columnar Temporal Databases. *Electronics* **2021**, *10*, 1478. https://doi.org/10.3390/electronics10121478

Academic Editor: Manohar Das

Received: 28 May 2021
Accepted: 16 June 2021
Published: 19 June 2021

1. Introduction

Depending on the layout of recording data on a storage device, databases can be broadly classified into two types, namely row databases and columnar databases (row and columnar databases are also referred to as horizontal and vertical databases, respectively). Row databases organize data as records, keeping all of the data associated with a record next to each other in a storage device. These databases are mostly based on ACID (ACID stands for Atomicity, Consistency, Isolation, and Duration) properties and are optimized for reading and writing rows efficiently. Examples of horizontal databases include MySQL [1] and Postgres [2]. Columnar databases organize data in the form of fields, keeping all of the data associated with a field next to each other in a storage device. These databases are mostly based on BASE (BASE stands for Basically Available, Soft state, and Eventually consistent) properties and are optimized for reading and computing on columns efficiently. Examples of columnar databases include Snowflake [3] and BigQuery [4]. Row databases and columnar databases have their own merits and demerits. As a result, there exists

no universally accepted best data layout for any application. Selecting an appropriate database layout depends on the user and/or application requirements. In general, row databases are suitable for online transaction processing (OLTP), while columnar databases are suitable for online analytical processing (OLAP). Since an objective of OLAP involves finding useful information in the data, the supportive move has been made in this paper to find user interest-based patterns in a columnar database. For the pattern mining algorithms, the columnar databases provide an advantage of cutting off several costly operations such as pattern support counting and candidate pruning as these operations become simple binary operations.

Frequent pattern mining is an important knowledge discovery technique with numerous practical applications such as market-basket analysis [5], air pollution analysis [6], and congestion analysis [7]. This technique was initially proposed by Agarwal et al. [8] to discover the set of frequently purchased itemsets (or patterns) in a super-market database. Several competent techniques were discussed in the literature [9–11] to enumerate all frequent patterns from a transactional database. Luna et al. [12] recently presented a survey on the advances that happened in the past 25 years of frequent pattern mining. The popular adoption and the successful adoption of this technique has been hindered by its limitation to discover regularities that may exist in a temporal database. When confronted with this problem in real-world applications, researchers have generalized the frequent pattern model to discover periodic-frequent patterns in a temporal database. This generalized model involves discovering all patterns in a temporal database that satisfy the user-specified *minimum support* (*minSup*) and *maximum periodicity* (*maxPer*) constraints. The *minSup* controls the minimum number of transactions that a pattern must cover in the database. The *maxPer* controls the maximum time interval within which a pattern must reoccur in the data. A classical application of periodic-frequent pattern mining is air pollution analytics. It involves identifying the geographical areas in which people were regularly exposed to harmful levels of air pollution. A periodic-frequent pattern discovered in our air pollution database is as follows:

$$\{1197, 1631, 1156\} \quad [support = 54\%,\ periodicity = 6\ hours].$$

The above pattern indicates that the people living close to the sensor identifiers, 1197, 1631, and 1156, were frequently and regularly (i.e., at least once every 6 hours) exposed to harmful levels of PM2.5. The produced information may help the users for various purposes, such as introducing location-specific pollution control policies and suggesting alternative residential areas for people with healthcare issues.

Several algorithms (e.g., PFP-growth [13], PFP-growth++ [14], and PS-growth [15]) have been described in the literature to find periodic-frequent patterns in a row database. To the best of our knowledge, there exists no algorithm that can find periodic-frequent patterns in a columnar temporal database. We can find periodic-frequent patterns by transforming a columnar temporal database into a row database. However, we must avoid such a naïve transformation process due to its high computational cost. With this motivation, this paper makes an effort to find periodic-frequent patterns in a columnar temporal database effectively.

Finding periodic-frequent patterns in columnar databases is non-trivial and challenging due to the following reasons:

1. Zaki et al. [16] first discussed the importance of finding frequent patterns in columnar databases. Besides, a depth-first search algorithm, called Equivalence CLass Transformation (ECLAT), was also described to find frequent patterns in a columnar database. Unfortunately, this algorithm cannot be directly used to find periodic-frequent patterns in a columnar temporal database. It is because the ECLAT algorithm completely disregards the temporal occurrence information of an item in the database.
2. The space of items in a database gives rise to an itemset lattice. The size of this lattice is $2^n - 1$, where n represents the total number of items in a database. This lattice

represents the search space for finding interesting patterns. Reducing this vast search space is a challenging task in pattern mining.

Example 1. *Figure 1a shows the itemset lattice of the three items a, b, and c. The size of this itemset lattice is* $2^3 - 1 = 7$. *The mining algorithm has to effectively search this huge lattice to find all desired partial periodic patterns in a columnar temporal database.*

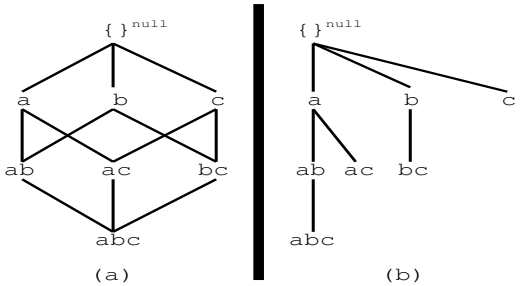

Figure 1. Search space of the items a, b, and c. (**a**) Itemset lattice and (**b**) Depth-first search on the lattice.

Against this background, we propose a novel and generic ECLAT algorithm to find all periodic-frequent patterns in a columnar temporal database. We call our algorithm Periodic Frequent-Equivalence CLass Transformation (PF-ECLAT). This paper is a substantially extended version of our conference paper [17] which reports a preliminary version of PF-ECLAT. In this paper, we have extended the related work by extensively understanding current literature. More importantly, the experimental results section (Section 5) has been greatly expanded by considering additional databases and algorithms. In this paper, we show that PF-ECLAT not only outperforms PFP-growth++ [14] but also outperforms PS-growth [15] on all databases irrespective of *maxPer* and *minSup* values. Finally, an additional case study on traffic congestion analytics was also presented to demonstrate our patterns' usefulness.

The main contributions of this paper are summarised as follows:

- This paper proposes a novel algorithm, called PF-ECLAT, to find periodic-frequent patterns in a columnar temporal database.
- To the best of our knowledge, this is the first algorithm that aims to find periodic-frequent patterns in a columnar temporal database. A key advantage of this algorithm over the state-of-the-art algorithms is that it can also be employed to find periodic-frequent patterns in a horizontal database.
- Experimental results on synthetic and real-world databases demonstrate that our algorithm is memory and runtime efficient and highly scalable.
- Finally, our algorithm's usefulness was demonstrated with two case studies. The first case study is air pollution analytics, where the proposed algorithm was used to identify geographical areas in which people were regularly exposed to harmful air pollutants in the whole of Japan. The second case study is traffic congestion analytics, where our algorithm was employed to find the set of road segments in which congestion was regularly observed in a transportation network.

The rest of the paper is organized as follows. Section 2 reviews the work related to our method. Section 3 introduces the model as a periodic-frequent pattern. Section 4 presents the proposed algorithm. Section 5 shows the experimental results. Section 6 concludes the paper with future research directions.

2. Related Work

2.1. Frequent Pattern Mining

Frequent pattern mining was introduced by Agarwal et al. [8] to identify interesting items (called frequent patterns) appearing in many transactions of the market-basket

database. However, later on, it has been used to disclose the correlation between different items according to their co-occurrence in a database. Apriori [8] algorithm is the first frequent pattern mining algorithm. It generates one-length frequent patterns after a single scan of the database, and these patterns will be used to generate the subsequent length patterns (called candidate patterns). These candidate patterns will be scanned against the database to extract the frequent patterns. Even though Apriori is a complete algorithm, it will take several database scans to generate the complete set of frequent patterns. As a result, it is a time-consuming algorithm. This issue has been resolved in ECLAT [16] and AprioriTID [8] algorithms, which will scan the entire database once and stores it in a vertical data layout format. Each row consists of two columns: the first column represents an item, and the second column stores the transaction number in which the item has appeared in the database. Based on this data layout, we can calculate the frequency of each item without scanning the database. Even though the vertical data layout format is used in both ECLAT and AprioriTID, both follow different strategies during generating frequent patterns. In particular, ECLAT follows a depth-first search strategy, while AprioriTID follows a breadth-first strategy to find frequent patterns. Several other algorithms were also developed in the literature [9–11,18] to find frequent patterns. Luna et al. [12] conducted a detailed survey on frequent pattern mining and presented the improvements that happened in the past 25 years. However, frequent pattern mining is inappropriate for identifying patterns that are regularly appearing in a temporal database.

2.2. Periodic-Frequent Pattern Mining

Tanbeer et al. [13] introduced the idea of periodic-frequent pattern mining. A highly compacted periodic frequent-tree (PF-tree) was constructed and applied a pattern growth technique to generate all periodic-frequent patterns in a database based on the user-specified $minSup$ and $maxPer$ constraints.

Amphawan et al. [19] designed an efficient best-first search-based algorithm named Mining Top-K Periodic-frequent Patterns (MTKPP) without using the user-specified $minSup$ constraint. Authors have introduced a list-based data structure named the top-K list to maintain k periodic-frequent patterns with the highest support. These top-K lists have been used during the mining process in the MTKPP algorithm to generate candidate patterns. If the candidate patterns periodicity is less than the user-specified $maxPer$ and support is greater than the support of the kth pattern in the top-K list, it will be included in the top-K periodic-frequent patterns list.

Uday et al. [20] introduced an extended multiple minimum support and multiple maximum periodicity model to efficiently discover periodic-frequent patterns consisting of both frequent and rare items. Authors have used two different constraints: minimum item support and maximum item periodicity, to identify useful patterns. Each pattern satisfies different minimum support and maximum periodicity based on the items available in it. Authors have also introduced a pattern-growth algorithm to discover the complete set of frequent and rare items using a novel and efficient tree-based data structure, called a multi-constraint periodic-frequent tree.

Amphawan et al. [21] introduced a novel technique to discover periodic-frequent patterns in a transactional database named approximate periodicity. It is used to reduce the time to calculate the periodicity of an item. Authors have introduced a novel and efficient tree-based data structure, called the Interval Transaction-ids List tree (ITL-tree), to maintain the occurrence information of an item in a compact manner using an interval transaction-ids list. A pattern-growth mining technique is also used to discover the complete set of periodic-frequent patterns by a bottom-up traversal of the ITL-tree based on the user-defined $minSup$ and $maxPer$ thresholds.

Uday et al. [22] introduced an interesting novel measure to discover periodic-frequent patterns in a transactional database named periodic-ratio. The authors have identified some of the interesting patterns which are almost periodically appearing in the database. A frequent pattern's periodic interestingness is calculated as the proportion of its periodic

occurrences in a database. A potential pattern is defined as a pattern whose periodic interestingness is greater than the user-specified minimum periodic-ratio, and support is greater than the user-specified $minSup$. These potential patterns were used to construct an extended periodic-frequent tree. Authors have also introduced an extended pattern-growth algorithm to discover the complete set of periodic-frequent patterns.

Rashid et al. [23] introduced an interesting novel measure for mining regularly frequent patterns in a transactional database named maximum variance ($maxVar$). A highly compacted regularly frequent pattern tree was constructed and applied a pattern growth technique to generate the set of regularly frequent patterns in a database based on the user-specified $minSup$ and $maxVar$ constraints. The $minSup$ controls the minimum number of transactions that a pattern must cover in the database. The $maxVar$ controls the maximum variance of intervals at which a pattern reoccurs in a database.

Uday et al. [14] introduced a novel greedy approach to discover periodic-frequent patterns. Authors have designed a two-phase architecture named expanding and shrinking to store all the patterns with support and periodicity efficiently. Where these phases have effectively utilized the newly introduced local periodicity concept. Finally, created a PF-tree++ and applied a pattern growth technique to generate periodic-frequent patterns in a database based on the user-specified $minSup$ and $maxPer$.

Anirudh et al. [15] introduced a novel concept of periodic summaries to find the periodic-frequent patterns in temporal databases. Authors have introduced a novel concept called periodic summaries-tree to maintain the time stamp information of the patterns in a database and designed a pattern growth algorithm to generate a complete set of periodic-frequent patterns.

Unfortunately, all of the above algorithms have used the concept of a row database. As a result, these algorithms cannot be directly applied to a columnar database.

3. Periodic-Frequent Pattern Model

Let I be the set of items. Let $X \subseteq I$ be a **pattern** (or an itemset). A pattern containing β, $\beta \geq 1$, number of items is called a **β-pattern**. A **transaction**, $t_k = (ts, Y)$, is a tuple, where $ts \in \mathbb{R}^+$ represents the timestamp at which the pattern Y has occurred. A **temporal database** TDB over I is a set of transactions, i.e., $TDB = \{t_1, \cdots, t_m\}$, $m = |TDB|$, where $|TDB|$ can be defined as the number of transactions in TDB. For a transaction $t_k = (ts, Y)$, $k \geq 1$, such that $X \subseteq Y$, it is said that X occurs in t_k (or t_k contains X) and such a timestamp is denoted as ts^X. Let $TS^X = \{ts^X_j, \cdots, ts^X_k\}$, $j, k \in [1, m]$ and $j \leq k$, be an **ordered set of timestamps** where X has occurred in TDB.

Example 2. *Let $I = \{a, b, c, d, e, f\}$ be the set of items. A hypothetical row temporal database generated from I is shown in Table 1. Without loss of generality, this row temporal database can be represented as a columnar temporal database as shown in Table 2. The temporal occurrences of each item in the entire database are shown in Table 3. The set of items 'b' and 'c', i.e., $\{b, c\}$ is a pattern. For brevity, we represent this pattern as 'bc'. This pattern contains two items. Therefore, it is a 2-pattern. The pattern 'bc' appears at the timestamps of 1, 3, 4, 6, 9, and 10. Therefore, the list of timestamps containing 'bc', i.e., $TS^{bc} = \{1, 3, 4, 6, 9, 10\}$.*

Table 1. Row database.

ts	Items	*ts*	Items
1	*abcf*	6	*abcd*
2	*bd*	7	*ab*
3	*abcd*	8	*cd*
4	*abce*	9	*abcd*
5	*cef*	10	*bcf*

Table 2. Columnar database.

ts	Items						*ts*	Items					
	a	***b***	***c***	***d***	***e***	***f***		***a***	***b***	***c***	***d***	***e***	***f***
1	1	1	1	0	0	1	6	1	1	1	1	0	0
2	0	1	0	1	0	0	7	1	1	0	0	0	0
3	1	1	1	1	0	0	8	0	0	1	1	0	0
4	1	1	1	0	1	0	9	1	1	1	1	0	0
5	0	0	1	0	1	1	10	0	1	1	0	0	1

Table 3. List of *ts* of an item.

Item	TS-List
a	1, 3, 4, 6, 7, 9
b	1, 2, 3, 4, 6, 7, 9, 10
c	1, 3, 4, 5, 6, 8, 9, 10
d	2, 3, 6, 8, 9
e	4, 5
f	1, 5, 10

Definition 1 (The $support$ of X)**.** *The number of transactions containing X in TDB is defined as the* ***support*** *of X and denoted as $sup(X)$. That is, $sup(X) = |TS^X|$.*

Example 3. *The support of 'bc,' i.e., $sup(bc) = |TS^{bc}| = |\{1,\ 3,\ 4,\ 6,\ 9,\ 10\}| = 6$.*

Definition 2 (Frequent pattern X)**.** *The pattern X is said to be a* ***frequent pattern*** *if $sup(X) \geq minSup$, where $minSup$ refers to the user-specified* minimum support *value.*

Example 4. *If the user-specified $minSup = 5$, then bc is said to be a frequent pattern because of $sup(bc) \geq minSup$.*

Definition 3 (*Periodicity* of X)**.** *Let ts_q^X and ts_r^X, $j \leq q < r \leq k$, be the two consecutive timestamps in TS^X. The time difference (or an inter-arrival time) between ts_r^X and ts_q^X is defined as a* **period** *of X, say p_a^X. That is, $p_a^X = ts_r^X - ts_q^X$. Let $P^X = (p_1^X,\ p_2^X, \cdots,\ p_r^X)$ be the set of all periods for pattern X. The* **periodicity** *of X, denoted as $per(X) = maximum(p_1^X,\ p_2^X, \cdots,\ p_r^X)$.*

Example 5. *The periods for this pattern are: $p_1^{bc} = 1\ (= 1 - ts_{initial})$, $p_2^{bc} = 2\ (= 3 - 1)$, $p_3^{bc} = 1\ (= 4 - 3)$, $p_4^{bc} = 2\ (= 6 - 4)$, $p_5^{bc} = 3\ (= 9 - 6)$, $p_6^{bc} = 1\ (= 10 - 9)$, and $p_7^{bc} = 0\ (= ts_{final} - 10)$, where $ts_{initial} = 0$ represents the timestamp of initial transaction and $ts_{final} = |TDB| = 10$ represents the timestamp of final transaction in the database. The* periodicity *of bc, i.e., $per(bc) = maximum(1,\ 2,\ 1,\ 2,\ 3,\ 1,\ 0) = 3$.*

Definition 4 (Periodic-frequent pattern X)**.** *The frequent pattern X is said to be a* ***periodic-frequent pattern*** *if $per(X) \leq maxPer$, where $maxPer$ refers to the user-specified* maximum periodicity *value.*

Example 6. *If the user-defined $maxPer = 3$, then the frequent pattern 'bc' is said to be a periodic-frequent pattern because $per(bc) \leq maxPer$. Similarly, bca and ba are also periodic-frequent patterns because $TS^{bca} = \{1,\ 3,\ 4,\ 6,\ 9\}$, $TS^{ba} = \{1,\ 3,\ 4,\ 6,\ 9,\ 10\}$, $sup(bca) = 5$, $sup(ba) = 6$, $per(bca) = 3$, and $per(ba) = 3$. The complete set of periodic-frequent patterns discovered from Table 3 are shown in Figure 3f without (i.e., Strikethrough) mark on the text.*

Definition 5 (Problem definition)**.** *Given a temporal database (TDB) and the user-specified* minimum support *($minSup$) and* maximum periodicity *($maxPer$) constraints, the aim is to*

discover the complete set of periodic-frequent patterns that have $support$ no less than $minSup$ and periodicity no more than the $maxPer$ constraints.

4. Proposed Algorithm

In this section, we first describe the procedure for finding one length periodic-frequent patterns (or 1-patterns) and transforming row database to columnar database. Next, we will explain the PF-ECLAT algorithm to discover a complete set of periodic-frequent patterns in columnar temporal databases. PF-ECLAT algorithm employs depth-first search (DFS) and the *downward closure property* (see Property 1) of periodic-frequent patterns to reduce the huge search space effectively.

Property 1 (The downward closure property [13])**.** *If Y is a periodic-frequent pattern, then $\forall X \subset Y$ and $X \neq \emptyset$, X is also a periodic-frequent pattern.*

4.1. PF-ECLAT Algorithm

4.1.1. Finding One Length Periodic-Frequent Patterns

Algorithm 1 describes the procedure to find 1-patterns using PFP-list, which is a dictionary. We now describe this algorithm's working using the row database shown in Table 1. Let $minSup = 5$ and $maxPer = 3$.

Algorithm 1 PeriodicFrequentItems(Row database (TDB), minimum support ($minSup$), maximum periodicity ($maxPer$)

1: Let $PFP\text{-}list = (X, TS\text{-}list(X))$ be a dictionary that records the temporal occurrence information of a pattern in a TDB. Let TS_l be a temporary list to record the *timestamp* of the last occurrence of an item in the database. Let Per be a temporary list to record the *periodicity* of an item in the database. Let $support$ be another temporary lists to record the *support* of an item in the database.
2: **for** each transaction $t_{cur} \in TDB$ **do**
3: Set $ts_{cur} = t_{cur}.ts$;
4: **for** each item $i \in t_{cur}.X$ **do**
5: **if** i does not exit in PFP-list **then**
6: Insert i and its timestamp into the PFP-list. Set $TS_l[i] = ts_{cur}$ and $Per[i] = (ts_{cur} - ts_{initial})$;
7: **else**
8: Add i's timestamp in the PFP-list. Update $TS_l[i] = ts_{cur}$ and $Per[i] = max(Per[i], (ts_{cur} - TS_l[i]))$;
9: **for** each item i in PFP-list **do**
10: $support[i] = length(TS\text{-}list(i))$
11: **if** $support[i] < minSup$ **then**
12: Prune i from the PFP-list;
13: **else**
14: Calculate $Per[i] = max(Per[i], (ts_{final} - TS_l[i]))$;
15: **if** $Per[i] > maxPer$ **then**
16: Prune i from the PFP-list.
17: Sort the remaining items in the PFP-list in ascending order or descending order of their $support$. Call PF-ECLAT(PFP-List).

We will scan the complete database once to generate 1-patterns and transforming the row database into a columnar database. The scan on the first transaction, "$1 : abcf$", with $ts_{cur} = 1$ inserts the items a, b, c, and f in the PFP-list. The timestamps of these items are set to 1 ($= ts_{cur}$). Similarly, Per and TS_l values of these items were also set to 1 and 1, respectively (lines 5 and 6 in Algorithm 1). The PFP-list generated after scanning the first transaction is shown in Figure 2a. The scan on the second transaction, "$2 : bd$", with $ts_{cur} = 2$ inserts the new item d into the PFP-list by adding 2 ($= ts_{cur}$) in its TS-list. Simultaneously, the Per and TS_l values were set to 2 and 2, respectively. On the other

hand, 2 ($= ts_{cur}$) was added to the TS-list of already existing item b with Per and TS_l set to 1 and 2, respectively (lines 7 and 8 in Algorithm 1). The PFP-list generated after scanning the second transaction is shown in Figure 2b. The scan on the third transaction, "$3 : bcd$", updates the TS-list, Per and TS_l values of b, c, and d in the PFP-list. The PFP-list generated after scanning the third transaction is shown in Figure 2c. The scan on the fourth transaction, "$4 : abce$", with $ts_{cur} = 4$ inserts the new item e into the PFP-list by adding 4 ($= ts_{cur}$) in its TS-list. Simultaneously, the Per and TS_l values were set to 4 and 4, respectively. On the other hand, updates the TS-list, Per, and TS_l values of already existing items a, b, c, and e in the PFP-list. The PFP-list generated after scanning the fourth transaction is shown in Figure 2d. A similar process is repeated for the remaining transactions in the database. The final PFP-list generated after scanning the entire database is shown in Figure 2e. The pattern e and f are pruned (using Property 1) from the PFP-list as its $support$ value is less than the user-specified $minSup$ value (lines 10 to 15 in Algorithm 1). The remaining patterns in the PFP-list are considered periodic-frequent patterns and sorted in descending order of their $support$ values. The final PFP-list generated after sorting the periodic-frequent patterns is shown in Figure 2f.

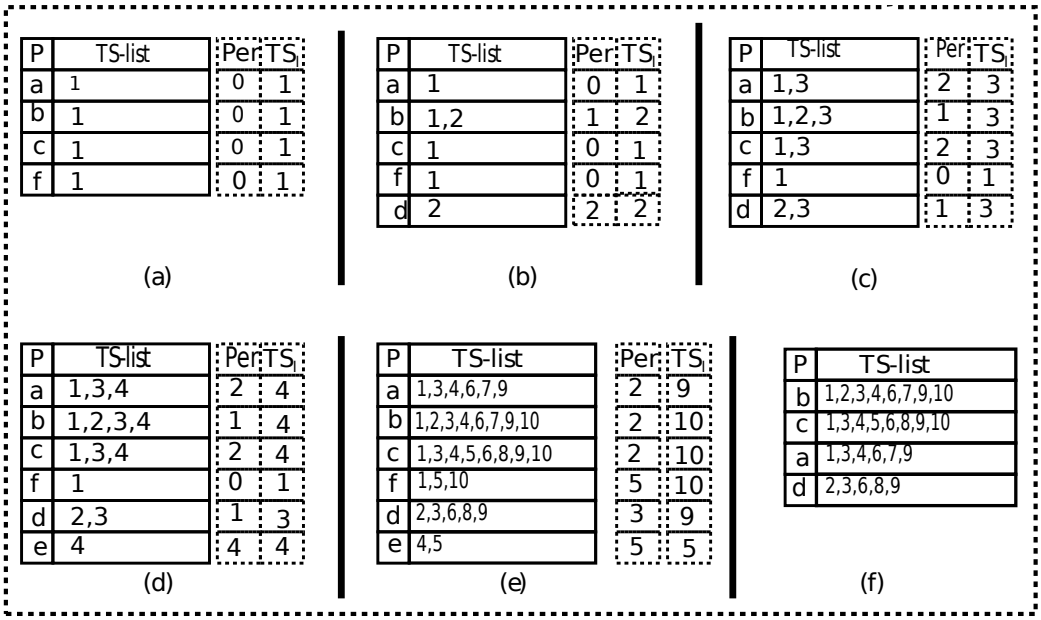

Figure 2. Finding periodic-frequent patterns. (**a**) after scanning the first transaction, (**b**) after scanning the second transaction, (**c**) after scanning the third transaction, (**d**) after scanning the fourth transaction, (**e**) after scanning the entire database, and (**f**) final list of periodic-frequent patterns sorted in descending order of their $support$ (or the size of TS-list).

4.1.2. Finding Periodic-Frequent Patterns Using PFP-List

Algorithm 2 describes the procedure for finding all periodic-frequent patterns in a database. We now describe the working of this algorithm using the newly generated PFP-list.

We start with item b, which is the first pattern in the PFP-list (line 2 in Algorithm 2). We record its $support$ and $periodicity$, as shown in Figure 3a. Since b is a periodic-frequent pattern, we move to its child node bc and generate its TS-list by performing intersection of TS-lists of b and c, i.e., $TS^{bc} = TS^b \cap TS^c$ (lines 3 and 4 in Algorithm 2). We record $support$ and $periodicity$ of bc, as shown in Figure 3b. We verify whether bc is a periodic-frequent or uninteresting pattern (line 5 in Algorithm 2). Since bc is the periodic-frequent pattern, we move to its child node bca and generate its TS-list by performing intersection of TS-lists of bc and a, i.e., $TS^{bca} = TS^{bc} \cap TS^a$. We record $support$ and $periodicity$ of bca, as shown in Figure 3c, and identified it as a periodic-frequent pattern. We once again, move to its child node $bcad$ and generate its TS-list by performing intersection of TS-lists of bca and bcd, i.e., $TS^{bcad} = TS^{bca} \cap TS^{bcd}$. As $support$ of $bcad$ is less than the user-specified $minSup$, we

will prune the pattern *bcad* from the periodic-frequent patterns list as shown in Figure 3d. As *bcad* is the leaf node in the set-enumeration tree (or as there exists no superset of *bcad*), we construct *bcd* with $TS^{bcd} = TS^{bc} \cap TS^{d}$. As $support$ of *bcd* is less than the user-specified $minSup$, we will prune the pattern *bcd* from the periodic-frequent patterns list as shown in Figure 3e. A similar process is repeated for remaining nodes in the set-enumeration tree to find all periodic-frequent patterns. The final list of periodic-frequent patterns generated from Table 1 is shown in Figure 3f. The above approach of finding periodic-frequent patterns using the downward closure property is efficient because it effectively reduces the search space and the computational cost. The correctness of our algorithm is based on Properties 2–4, and shown in Lemma 1.

Algorithm 2 PF-ECLAT(PFP-List)

```
1: for each item i in PFP-List do
2:    Set pi = ∅ and X = i;
3:    for each item j that comes after i in the PFP-list do
4:       Set Y = X ∪ j and TS^Y = TS^X ∩ TS^j;
5:       if sup(TS^Y) ≥ minSup and per(TS^Y) ≤ maxPer then
6:          Add Y to pi and Y is considered as periodic-frequent itemset;
7:    PF-ECLAT(pi)
```

Property 2. *Let X, Y, $Z \subset I$ be three patterns such that $X \neq \emptyset$, $Y \neq \emptyset$, $Z \neq \emptyset$, $X \cap Y = \emptyset$ and $Z = X \cup Y$. If TS^X and TS^Y denote the set of timestamps at which patterns X and Y have respectively occurred in the database, then the set of timestamps at which Z has appeared in the database, i.e., $TS^Z = TS^X \cap TS^Y$.*

Property 3. *If $minSup > |TS^X|$, then X cannot be frequent pattern. Moreover, $\forall Z \supset X$, Z cannot be a frequent pattern.*

Proof. If $X \subset Z$, then $TS^X \supseteq TS^Z$. Thus, $minSup > |TS^X| \geq |TS^Z|$. Thus, Z cannot be a frequent pattern. Hence proved. □

Property 4. *If $per(TS^X) > maxPer$, then X cannot be periodic pattern. Moreover, $\forall Z \supset X$, Z cannot be a periodic pattern.*

Proof. If $X \subset Z$, then $TS^X \supseteq TS^Z$. Thus, $per(TS^Z) \geq per(TS^X) > maxPer$. Thus, Z cannot be a periodic pattern. Hence proved. □

Lemma 1. *Let $X, Y, Z \subset I$ be three patterns such that $X \neq \emptyset$, $Y \neq \emptyset$, $Z \neq \emptyset$, $X \cap Y = \emptyset$ and $Z = X \cup Y$. If X or Y is not periodic-frequent patterns, then Z cannot be a periodic-frequent pattern. In other words, we do not need to check whether Z is a periodic-frequent pattern if any one of its supersets is not a periodic-frequent pattern.*

Proof. The correctness of the above statement is straightforward to prove from Properties 2–4. Hence proved. □

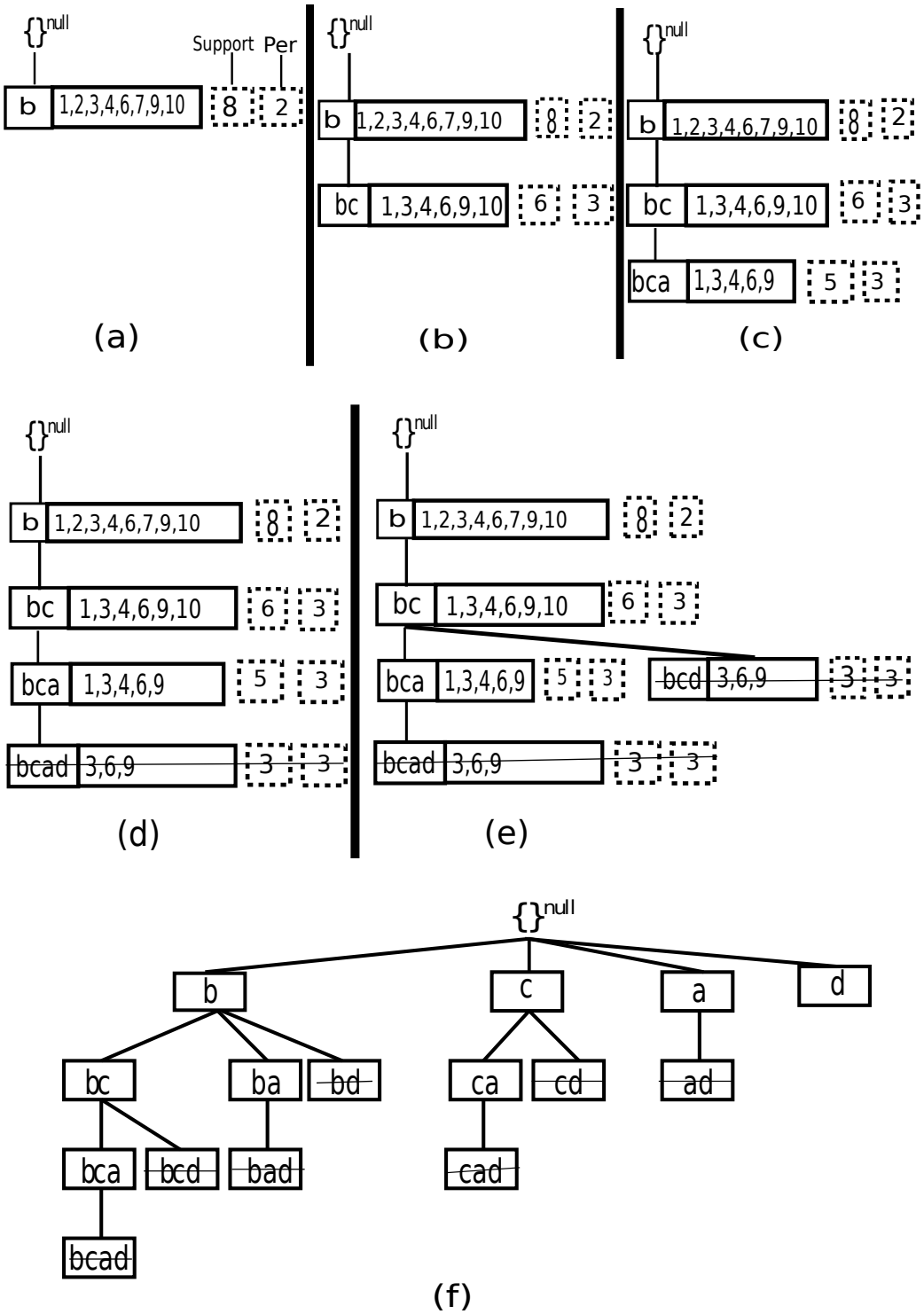

Figure 3. Mining periodic-frequent patterns using DFS: (**a**) identifying '*b*' is periodic-frequent pattern or not, (**b**) identifying '*bc*' is periodic-frequent pattern or not, (**c**) identifying '*bca*' is periodic-frequent pattern or not, (**d**) identifying '*bcad*' is periodic-frequent pattern or not, (**e**) identifying '*bcd*' is periodic-frequent pattern or not, and (**f**) final list of periodic-frequent patterns shown without Strikethrough mark on the text..

5. Experimental Results

In this section, we first compare the PF-ECLAT against the state-of-the-art algorithms (E.g., PFP-growth [13], PFP-growth++ [14], and PS-growth [15]) and show that our algorithm is not only memory and runtime efficient, but also highly scalable as well. Next, we

describe the usefulness of our algorithm with two case studies: traffic congestion analytics and air pollution analytics.

5.1. Experimental Setup

The algorithms PFP-growth, PFP-growth++, PS-growth, and PF-ECLAT, were developed in Python 3.7 and executed on Intel(R) Core i5-3230M CPU between 2.6 GHz to 3.2 GHz, as base frequency and Turbo Boost, respectively with 4GB RAM machine running Ubuntu 18.04 operating system. The experiments have been conducted on both real-world (**BMS-WebView-1**, **Pollution**, **Drought**, **Congestion**, **BMS-WebView-2**, and **Kosarak**) databases and synthetic (**T10I4D100K**).

The **T10I4D100K** is a sparse synthetic database generated using the procedure described in [8]. This database was widely used to evaluate various pattern-mining algorithms. The **BMS-WebView-1** and **BMS-WebView-2** are real-world sparse databases containing clickstream data of an anonymous eCommerce company. These databases were used in KDD Cup 2000. Both of these databases contain very long transactions. The **Kosarak** is a real-world, massive sparse database. This paper employs this database to evaluate the scalability of PFP-growth, PFP-growth++, PS-growth, and PF-ECLAT algorithms. All of the above databases have been downloaded from Sequence Pattern Mining Framework (SPMF) [24] repository. The **Drought** [25] is a very high-dimensional real-world dense database.

Monitoring traffic congestion in smart cities is a challenging problem of great importance in Intelligent Transportation Systems. In this context, JApan Road Traffic Information Center (JARTIC) [26] has set up a nationwide sensor network to monitor traffic congestion throughout Japan. In this network, each sensor means congestion on a road segment at 5-min intervals. The big data generated by this network naturally represents a quantitative (or non-binary) columnar temporal database. We have converted this database into a binary columnar database by specifying a threshold value of 200 m. It is because congestions lengths less than 200 m are often due to waiting time at a red signal. In this expert, we use the binary columnar traffic **Congestion** database produced in Kobe, the 5th largest city in Japan.

Air pollution is a significant cause of the cardio-respiratory problems reported in Japan; on average, 60,000 people die in Japan annually [27]. To confront this problem, The Ministry of Environment, Japan, has set up a sensor network system, called SORAMAME [28], to monitor air pollution throughout Japan. Each sensor in this network collects pollution levels of various air pollutants at hourly intervals. This experiment uses the 3-month data of PM2.5 pollutants generated by all sensors situated throughout Japan. The **Pollution** database is a high-dimensional and dense database containing many long transactions.

The statistics of all the above databases were shown in Table 4. The complete evaluation results and the databases and algorithms have been provided through GitHub [29] to verify the repeatability of our experiments. We are not providing the Congestion databases on GitHub due to confidential reasons.

Table 4. Statistics of the databases.

S. No	Database	Type	Nature	Transaction Length			Total Transactions
				Min.	Avg.	Max.	
1	BMS-WebView-1	Real	Sparse	1	3	267	59,602
2	Pollution	Real	Dense	11	460	971	720
3	Drought	Real	Dense	6289	8341	10,122	766
4	Congestion	Real	Sparse	1	58	337	8928
5	BMS-WebView-2	Real	Sparse	2	5	161	77,512
6	T10I4D100K	Synthetic	Sparse	2	11	29	100,000
7	Kosarak	Real	Sparse	2	9	2,499	990,000

5.2. Evaluation of PFP-Growth, PFP-Growth++, PS-Growth, and PF-ECLAT Algorithms by Varying maxPer Constraint

In this experiment, we evaluate PFP-growth, PFP-growth++, PS-growth, and PF-ECLAT algorithms' performance by varying only the *maxPer* constraint in each of the databases. The *minSup* value in each of the databases will be set to a particular value. The *minSup* in BMS-WebView-1, Pollution, Drought, Congestion, BMS-WebView-2, and T10I4D100K databases has been set at 0.07(%), 51(%), 57(%), 30(%), 0.2(%), and 0.1(%), respectively.

First, the runtime of the PF-ECLAT algorithm is compared with PFP-growth, PFP-growth++, and PS-growth algorithms. Figure 4 shows that PF-ECLAT outperforms the compared state-of-the-art algorithms on all databases. The vertical and horizontal axes represent the runtime (in milliseconds) and *maxPer* threshold values in each subfigure, respectively. (*i*) It can be observed that the PF-ECLAT runs faster than the PS-growth algorithm. It means that periodic calculation of the PF-ECLAT algorithm is very effective and can prune many non-periodic patterns as fast as possible. In addition, the results show that the PF-ECLAT runs faster than the PFP-growth++ algorithm. (*ii*) In general, for all databases, when the *maxPer* threshold value is increased, the running time of the algorithms is also increased. In that case, PF-ECLAT can be much more efficient than the remaining algorithms, especially on BMS-WebView-1, Pollution, Drought, and Congestion databases. (*iii*) We observed a marginal runtime difference between PF-ECLAT and remaining algorithms in BMS-WebView-2 (sparse nature with short transactions) and T10I4D100K (sparse nature with short transactions) databases. Our investigation into the marginal runtime improvement cause has revealed that PS-growth summarised the database to quickly generate periodic-frequent patterns when the database base sparse nature with short transactions. (*iv*) Generally, the PFP-List structure used in the PF-ECLAT algorithm is more compact and efficient than the one used in all the other state-of-the-art algorithms.

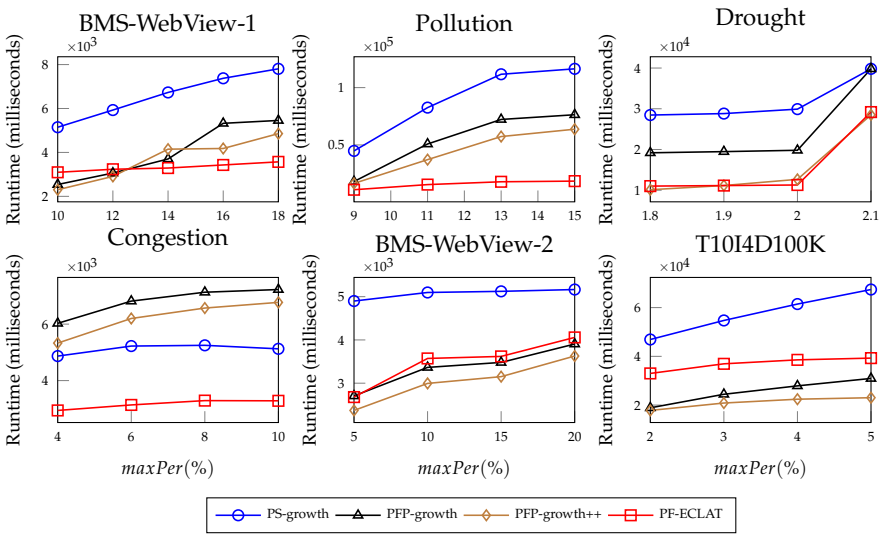

Figure 4. Runtime evaluation of algorithms at constant *minSup*.

Second, the memory consumption of the PF-ECLAT algorithm is compared with PFP-growth, PFP-growth++, and PS-growth algorithms. Figure 5 shows that PF-ECLAT outperforms the compared state-of-the-art algorithms on all databases. The vertical axis and horizontal axis represent the memory (in Kilobytes) and *maxPer* threshold values in each subfigure, respectively. The subsequent observations may be drawn from this figure: (*i*) Raise in *maxPer* increases PFP-growth, PFP-growth++, PS-growth, and PF-ECLAT algorithms' memory requirements. (*ii*) In every database (i.e., sparse or dense database containing either short or long transactions), PF-ECLAT consumed considerably

less memory over all other state-of-the-art algorithms at any given *maxPer* value. More importantly, the difference was significantly high at high *maxPer* values. (*iii*) In addition, the PF-ECLAT consumes less memory than PS-growth, although they are very close in some cases. Thus, the PFP-List structure used in the PF-ECLAT algorithm helps reduce the proposed algorithm's memory usage.

Finally, the number of patterns was measured for various *maxPer* threshold values on each database. In Figure 6, vertical axes denote the number of patterns, and horizontal axes indicate the corresponding *maxPer* threshold values. In general, PFP-growth, PFP-growth++, PS-growth, and PF-ECLAT generate the same number of periodic-frequent patterns in each of the databases. It can be observed that an increase in *maxPer* has increased the number of periodic-frequent patterns. With an increase in the *maxPer* threshold, most of the patterns have become periodic patterns.

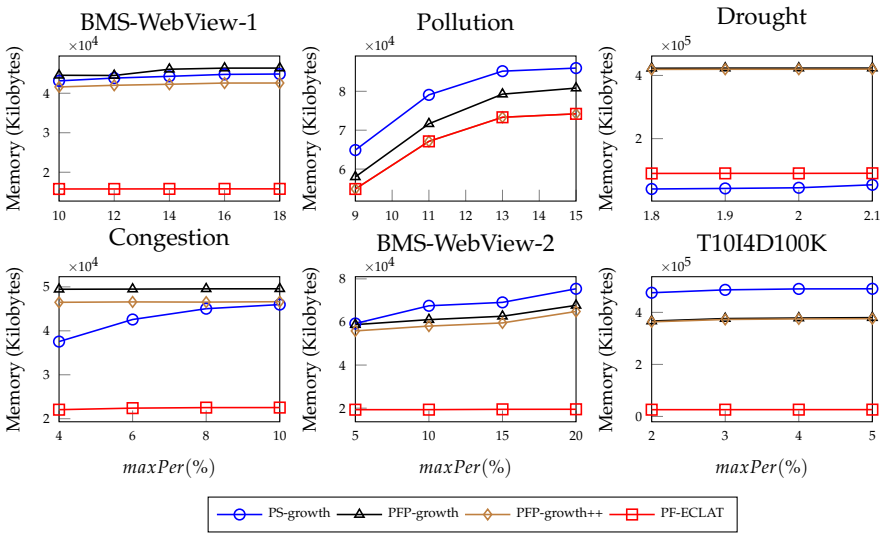

Figure 5. Memory evaluation of algorithms at constant *minSup*.

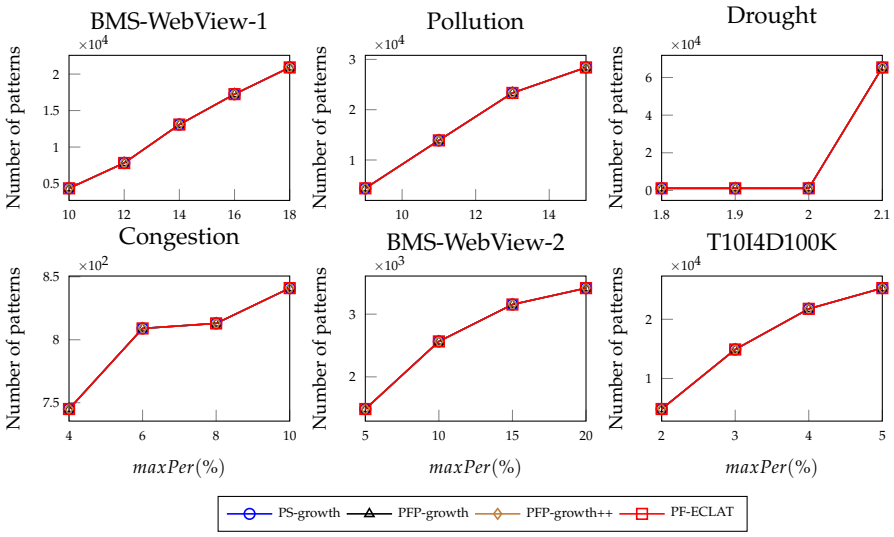

Figure 6. Patterns evaluation of algorithms at constant *minSup*.

5.3. Evaluation of PFP-Growth, PFP-Growth++, PS-Growth, and PF-ECLAT Algorithms by Varying minSup Constraint

We have evaluated PFP-growth, PFP-growth++, PS-growth, and PF-ECLAT algorithms' in the previous subsection by varying only the *maxPer* value. We now evaluate PFP-growth, PFP-growth++, PS-growth, and PF-ECLAT algorithms performance by vary-

ing only the *minSup* constraint in each of the databases. The *maxPer* value in each of the databases will be set to a particular value. The *maxPer* in BMS-WebView-1, Pollution, Drought, Congestion, BMS-WebView-2, and T10I4D100K databases has been set at 40%, 51%, 5%, 35%, 54%, and 20% respectively.

First, the runtime of the PF-ECLAT algorithm is compared with PFP-growth, PFP-growth++, and PS-growth algorithms. Figure 7 shows that PF-ECLAT outperforms the compared state-of-the-art algorithms on all databases. The vertical axis and horizontal axis represent the runtime (in milliseconds) and *minSup* threshold values in each subfigure, respectively. The subsequent observations may be drawn from this figure: (*i*) Raise in *minSup* decreases the runtime requirements of all PFP-growth, PFP-growth++, PS-growth, and PF-ECLAT algorithms. However, PF-ECLAT requires considerably less runtime over PFP-growth, PFP-growth++, and PS-growth on any database at any given *minSup* value. (*ii*) In every database, PF-ECLAT completed the mining process much faster than the PFP-growth++ algorithm. More importantly, PF-ECLAT was several times faster than PFP-growth++, especially at high *minSup* values. (*iii*) We have observed a marginal runtime difference between PF-ECLAT and PS-growth algorithms in BMS-WebView-2 (sparse nature with short transactions) and T10I4D100K (sparse nature with short transactions) databases. Our investigation into the marginal runtime improvement cause has revealed that PS-growth summarised the database to quickly generate periodic-frequent patterns when the database base is sparse with short transactions. However, in BMS-WebView-1, Pollution, Drought, and Congestion databases, PF-ECLAT was an order of magnitude time faster than the PS-growth. More importantly, PF-ECLAT was several times faster than PS-growth, especially at high *minSup*.

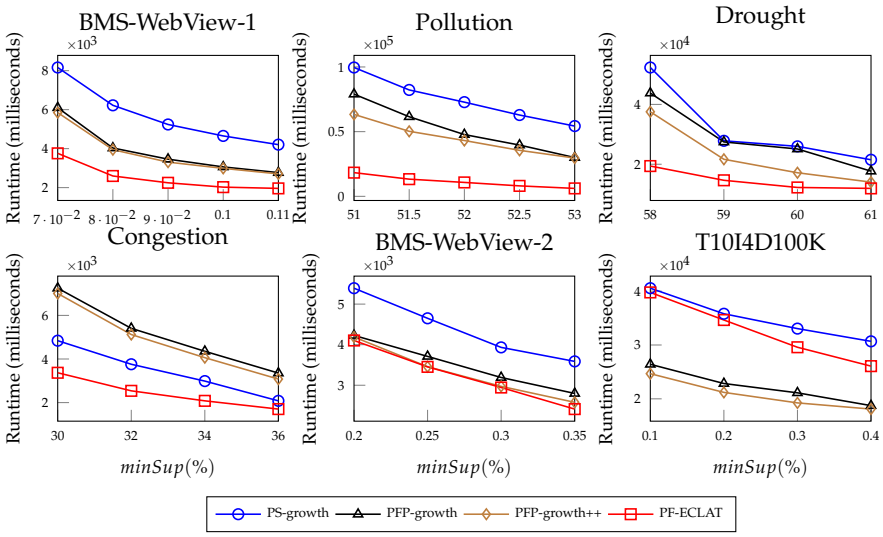

Figure 7. Runtime evaluation of algorithms at constant *maxPer*.

Second, the memory consumption of the PF-ECLAT algorithm is compared with PFP-growth, PFP-growth++, and PS-growth algorithms. Figure 8 shows that PF-ECLAT outperforms the compared state-of-the-art algorithms on all databases. The vertical axis and horizontal axis represent the memory (in Kilobytes) and *minSup* threshold values in each subfigure, respectively. The subsequent observations may be drawn from this figure: (*i*) Raise in *minSup* decreases PFP-growth, PFP-growth++, PS-growth, and PF-ECLAT algorithms' memory requirements. (*ii*) In every database (i.e., sparse or dense database containing either short or long transactions), PF-ECLAT consumed considerably less memory over all other state-of-the-art algorithms at any given *minSup* value. More importantly, the difference was significantly high at low *minSup* values. (*iii*) It is evident that the PFP-List structure used in the PF-ECLAT algorithm is very effective and able to reduce the memory usage of the proposed algorithm.

Finally, the number of patterns was measured for various *minSup* threshold values on each database. In Figure 9, vertical axes denote the number of patterns, and horizontal axes indicate the corresponding *minSup* threshold values. In general, PFP-growth, PFP-growth++, PS-growth, and PF-ECLAT generate the same number of periodic-frequent patterns in each of the databases. It can be observed that an increase in *minSup* has decreased the number of periodic-frequent patterns. It is because several patterns fail to fulfill the *minSup* constraint with an increase in the *minSup* value.

From the above two Sections 5.2 and 5.3, it is evident that the PFP-List structure used in the PF-ECLAT algorithm helps to eliminate many non-candidate patterns from the search space and thus reduce the runtime and memory usage of the PF-ECLAT algorithm.

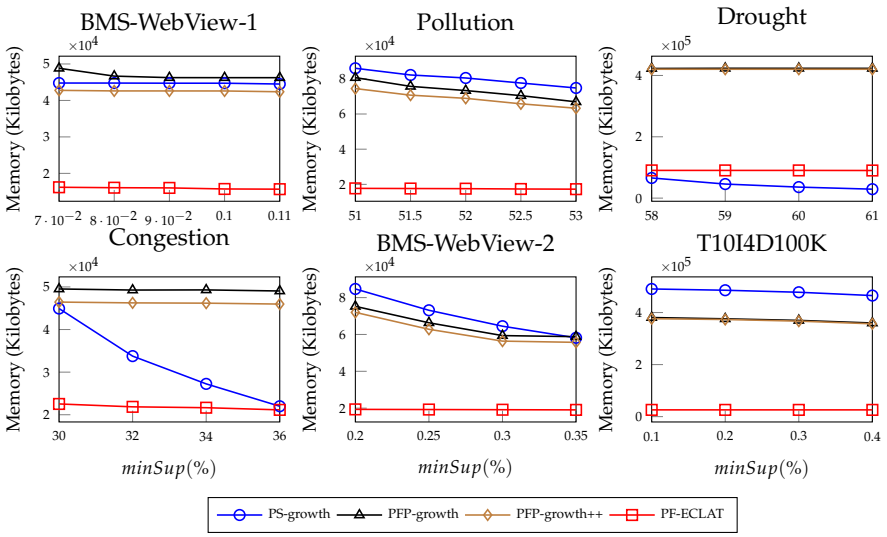

Figure 8. Memory evaluation of algorithms at constant *maxPer*.

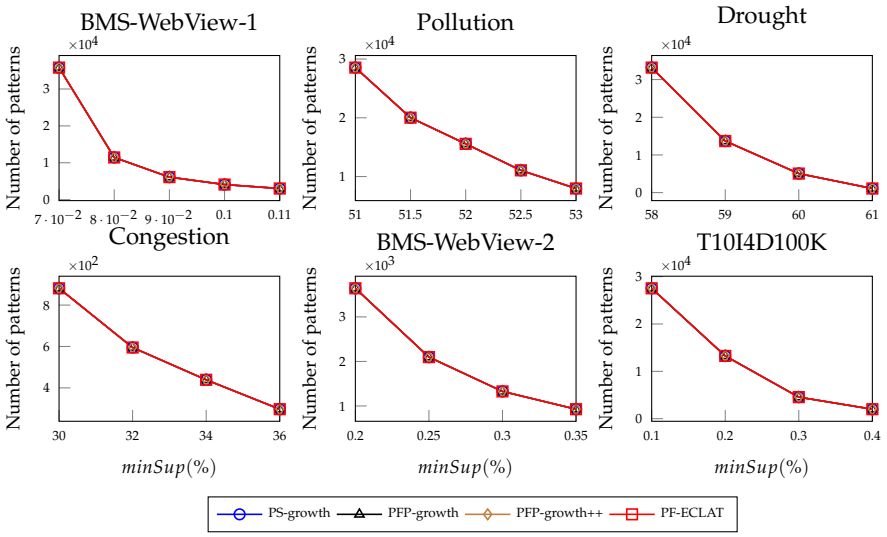

Figure 9. Patterns evaluation of algorithms at constant *maxPer*.

5.4. Scalability Test

The Kosarak database was divided into five portions of 0.2 million transactions in each part. Then we investigated the performance of PFP-growth, PFP-growth++, PS-growth, and PF-ECLAT algorithms after accumulating each portion with previous parts. Figure 10 show the runtime and memory requirements of all algorithms at different database sizes when $minSup = 1$ (%) and $maxPer = 0.1$ (%). The following two observations can be drawn from these figures: (*i*) Runtime and memory requirements of PFP-growth, PFP-growth++, PS-growth, and PF-ECLAT algorithms' increase almost linearly with the increase in database

size. (*ii*) At any given database size, PF-ECLAT consumes less runtime and memory as compared to the remaining algorithms.

5.5. A Case Study 1: Finding Areas Where People Have Been Regularly Exposed to Hazardous Levels of PM2.5 Pollutant

The Ministry of Environment, Japan has set up a sensor network system, called SORA-MAME [28], to monitor air pollution throughout Japan, as shown in Figure 11a. The raw data produced by these sensors, i.e., quantitative columnar database (see Figure 11b) can be transformed into a binary columnar database, if the raw data value is $\geq$15 (see Figure 11c). The transformed data is provided to the PF-ECLAT algorithm (see Figure 11d) to identify all sets of sensor identifiers (patterns) in which pollution levels are high (see Figure 11e). The spatial locations of interesting patterns generated from the **Pollution** database are visualized in Figure 11f. It can be observed that most of the sensors in this figure are situated in the southeast of Japan. Thus, it can be inferred that people working or living in the southeast parts of Japan were periodically exposed to high levels of PM2.5. Such information may be useful to the Ecologists in devising policies to control pollution and improve public health. Please note that more in-depth studies, such as finding high polluted areas on weekends or particular time intervals of a day, can also be carried out with our algorithm efficiently.

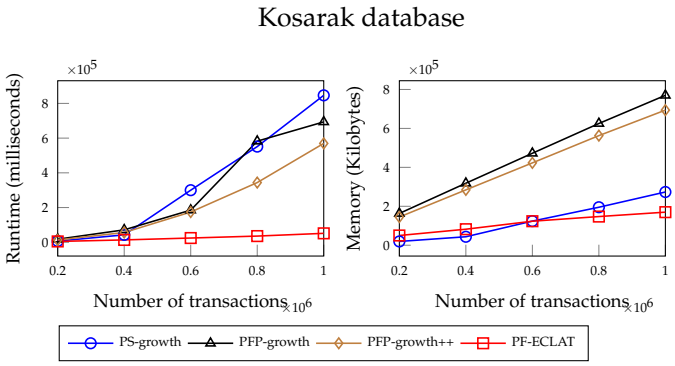

Figure 10. Scalability of PFP-growth, PFP-growth++, PS-growth, and PF-ECLAT.

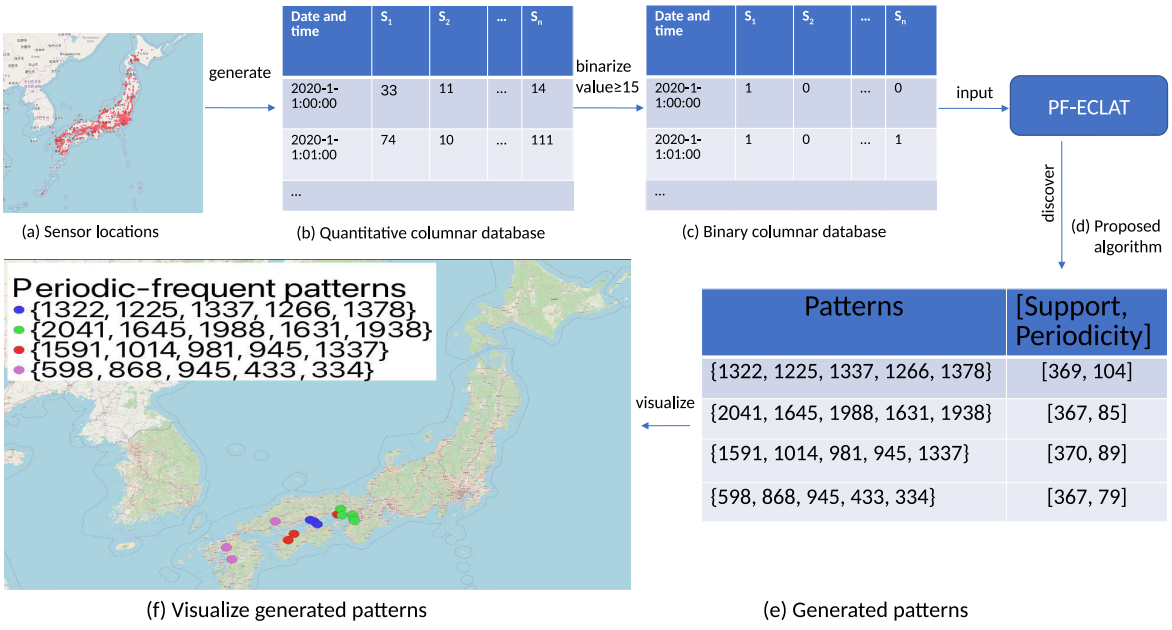

Figure 11. Finding periodic-frequent patterns in the Pollution database. The terms 's_1', 's_2', $\cdots$ 's_n' represents 'sensor identifiers'.

5.6. A Case Study 2: Traffic Congestion Analytics

Typhoon Nangka struck Kobe, Japan, on 17-July-2005. This typhoon dropped 29 inches of rainfall, causing floods. Almost 350,000 people were asked to flee to high-level areas to protect themselves from surges. Thus, causing significant congestion in the traffic network. To monitor the traffic congestion, JApan Road Traffic Information Center (JARTIC) [26] had deployed a sensor network to monitor congestion throughout Japan. The road network covered by the traffic congestion measuring sensors in Kobe, Japan, is shown in Figure 12a. The raw data produced by these sensors, i.e., quantitative columnar database (see Figure 12b) can be transformed into a binary columnar database, if the raw data value is $\geq$15 (see Figure 12c). The transformed data are provided to the PF-ECLAT algorithm (see Figure 11d) to discover all sets of sensor identifiers (patterns) in which traffic congestion is very high. The spatial locations of interesting patterns generated from the **Congestion** database are visualized in Figure 12e. In this case study, we have also demonstrated the usefulness of the discovered patterns using Nangka's rainfall data. When the rainfall data of the typhoon in the respective hour is overlaid on the discovered patterns as shown in Figure 12f, it can be observed that the generated information may additionally determined to be extremely useful to the users in the traffic control room to take effective decisions, such as diverting the traffic and suggesting police patrol routes to the users. In Figure 12f, road segments that need considerable attention are indicated with a black circle. It can be observed that the black circle moved from left to right in 4 h. Such information can be found to be very useful in traffic management.

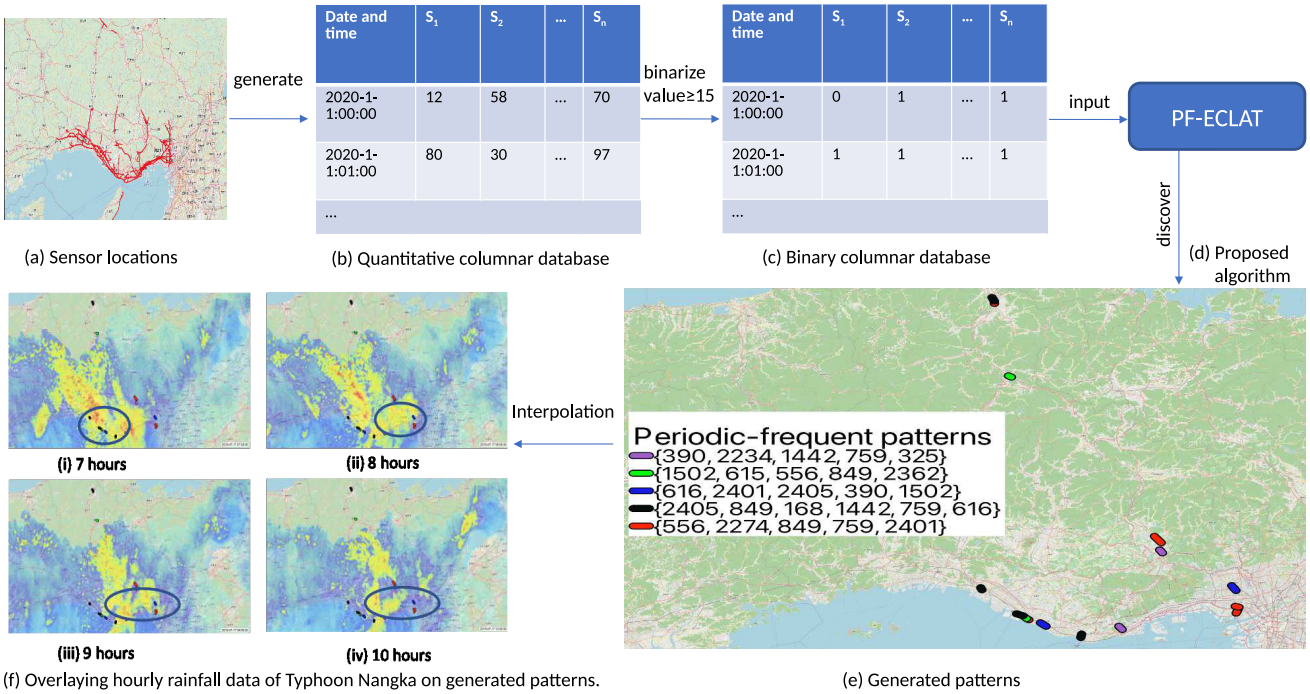

Figure 12. Finding periodic-frequent patterns in the Congestion database. The terms 's_1', 's_2', $\cdots$ 's_n' represents 'road sensor identifiers'.

6. Conclusions and Future Work

This paper has proposed an efficient algorithm named Periodic Frequent-Equivalence CLass Transformation (PF-ECLAT) to find periodic-frequent patterns in columnar temporal databases. Two constraints, *minimum support* and *maximum periodicity*, were utilized to discard uninteresting patterns. The PFP-List structure used in the PF-ECLAT algorithm helps to eliminate many non-candidate patterns from the search space and thus reduces the runtime and memory usage of the PF-ECLAT algorithm. The performance of the PF-ECLAT is verified by comparing it with other algorithms on different real-world and

synthetic databases. Experimental analysis shows that PF-ECLAT exhibits high performance in periodic-frequent pattern mining and can obtain periodic-frequent patterns faster and with less memory usage against the state-of-the-art algorithms. Finally, we have presented our model's usefulness with two case studies: air pollution analytics and traffic congestion analytics.

Future work may be expanded as follows, but the scope is not limited: We would like to extend our algorithm to the distributed environment to find periodic-, partial- and fuzzy periodic-frequent patterns in very large temporal databases. In addition, we would like to investigate novel measures or techniques to reduce further the computational cost of mining the periodic-frequent patterns.

Author Contributions: Conceptualization, P.R., P.L., B.V.V.R. and R.U.K.; Data curation, P.R., P.L., B.V.V.R. and R.U.K.; Formal analysis, P.R., P.L., B.V.V.R. and R.U.K.; Funding acquisition, R.U.K.; Investigation, P.R., P.L., B.V.V.R. and R.U.K.; Methodology, P.R., P.L., B.V.V.R. and R.U.K.; Project administration, P.R., P.L., B.V.V.R. and R.U.K.; Resources, P.R., P.L., B.V.V.R. and R.U.K., Y.W. and K.Z.; Software, P.R., P.L., B.V.V.R. and R.U.K.; Supervision, P.R., P.L., B.V.V.R. and R.U.K.; Validation, P.R., P.L., B.V.V.R. and R.U.K. Y.W.; Visualization, P.R., P.L., B.V.V.R. and R.U.K.; Writing—original draft, P.R., P.L., B.V.V.R. and R.U.K.; Writing—review & editing, P.R., P.L., B.V.V.R. and R.U.K. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by JSPS Kakenhi 21K12034.

Data Availability Statement: Data available in a publicly accessible repository that does not issue DOIs. Publicly available datasets were analyzed in this study. This data can be found here: [http://www.philippe-fournier-viger.com/spmf/index.php?link=datasets.php] (accessed on 4 June 2020).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. MySQL. Available online: https://www.mysql.com/ (accessed on 10 March 2021).
2. PostGres. Available online: https://www.postgresql.org/ (accessed on 10 March 2021).
3. SnowFlake. Available online: https://www.snowflake.com/ (accessed on 10 March 2021).
4. BigQuery. Available online: https://cloud.google.com/bigquery (accessed on 10 March 2021).
5. Brijs, T.; Swinnen, G.; Vanhoof, K.; Wets, G. Using Association Rules for Product Assortment Decisions: A Case Study. In Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Diego, CA, USA, 15–18 August 1999; pp. 254–260.
6. Kiran, R.U.; Shrivastava, S.; Fournier-Viger, P.; Zettsu, K.; Toyoda, M.; Kitsuregawa, M. Discovering Frequent Spatial Patterns in Very Large Spatiotemporal Databases. In Proceedings of the 28th International Conference on Advances in Geographic Information Systems (SIGSPATIAL '20), Seattle, WA, USA, 3–6 November 2020; Lu, C., Wang, F., Trajcevski, G., Huang, Y., Newsam, S.D., Xiong, L., Eds.; ACM: New York, NY, USA, 2020; pp. 445–448.
7. Tran-The, H.; Zettsu, K. Discovering co-occurrence patterns of heterogeneous events from unevenly-distributed spatiotemporal data. In Proceedings of the 2017 IEEE International Conference on Big Data (BigData 2017), Boston, MA, USA, 11–14 December 2017; pp. 1006–1011.
8. Agrawal, R.; Imieliński, T.; Swami, A. Mining association rules between sets of items in large databases. In Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, Washington, DC, USA, 26–28 May 1993; pp. 207–216.
9. Han, J.; Cheng, H.; Xin, D.; Yan, X. Frequent Pattern Mining: Current Status and Future Directions. *Data Min. Knowl. Discov.* **2007**, *15*, 55–86. [CrossRef]
10. Aggarwal, C.C. Applications of Frequent Pattern Mining. In *Frequent Pattern Mining*; Springer International Publishing: Berlin/Heidelberg, Germany, 2014; pp. 443–467.
11. Fournier-Viger, P.; Lin, J.C.W.; Kiran, R.U.; Koh, Y.S. A Survey of Sequential Pattern Mining. *Data Sci. Pattern Recognit.* **2017**, *1*, 54–77.
12. Luna, J.M.; Fournier-Viger, P.; Ventura, S. Frequent itemset mining: A 25 years review. *Wiley Interdiscip. Rev. Data Min. Knowl. Discov.* **2019**, *9*, e1329. [CrossRef]
13. Tanbeer, S.K.; Ahmed, C.F.; Jeong, B.S.; Lee, Y.K. Discovering Periodic-Frequent Patterns in Transactional Databases. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*; Springer: Berlin/Heidelberg, Germany, 2009; pp. 242–253.
14. Kiran, R.U.; Kitsuregawa, M. Novel Techniques to Reduce Search Space in Periodic-Frequent Pattern Mining. In *Database Systems for Advanced Applications*; Bhowmick, S.S., Dyreson, C.E., Jensen, C.S., Lee, M.L., Muliantara, A., Thalheim, B., Eds.; Springer International Publishing: Berlin/Heidelberg, Germany, 2014; pp. 377–391.

15. Anirudh, A.; Kiran, R.U.; Reddy, P.K.; Kitsuregawa, M. Memory efficient mining of periodic-frequent patterns in transactional databases. In Proceedings of the 2016 IEEE Symposium Series on Computational Intelligence, Athens, Greece, 6–9 December 2016; pp. 1–8.
16. Zaki, M.J. Scalable algorithms for association mining. *IEEE Trans. Knowl. Data Eng.* **2000**, *12*, 372–390. [CrossRef]
17. Ravikumar, P.; Likitha, P.; Kiran, R.U.; Watanobe, Y.; Zettsu, K. Towards Efficient Discovery of Periodic-Frequent Patterns in Columnar Temporal Databases. In Proceedings of the 2021 International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems(IEA/AIE), Kuala Lumpur, Malaysia, 26–29 July 2021; accepted and to be presented.
18. Han, J.; Pei, J.; Yin, Y.; Mao, R. Mining Frequent Patterns without Candidate Generation: A Frequent-Pattern Tree Approach. *Data Min. Knowl. Discov.* **2004**, *8*, 53–87. [CrossRef]
19. Amphawan, K.; Lenca, P.; Surarerks, A. Mining Top-K Periodic-Frequent Pattern from Transactional Databases without Support Threshold. In *International Conference on Advances in Information Technology*; Springer: Berlin/Heidelberg, Germany, 2009; pp. 18–29.
20. Kiran, R.U.; Reddy, P.K. Towards efficient mining of periodic-frequent patterns in transactional databases. In *International Conference on Database and Expert Systems Applications*; Springer: Berlin/Heidelberg, Germany, 2010; pp. 194–208.
21. Amphawan, K.; Surarerks, A.; Lenca, P. Mining Periodic-Frequent Itemsets with Approximate Periodicity Using Interval Transaction-Ids List Tree. In Proceedings of the 2010 Third International Conference on Knowledge Discovery and Data Mining, Phuket, Thailand, 9–10 January 2010; pp. 245–248. [CrossRef]
22. Kiran, R.U.; Reddy, P.K. An Alternative Interestingness Measure for Mining Periodic-Frequent Patterns. In *Proceedings of the 16th International Conference on Database Systems for Advanced Applications—Volume Part I (DASFAA'11)*; Springer: Berlin/Heidelberg, Germany, 2011; pp. 183–192.
23. Rashid, M.M.; Karim, M.R.; Jeong, B.S.; Choi, H.J. Efficient mining regularly frequent patterns in transactional databases. In *International Conference on Database Systems for Advanced Applications*; Springer: Berlin/Heidelberg, Germany, 2012; pp. 258–271.
24. Fournier-Viger, P. SPMF: A Java Open-Source Data Mining Library. 2020. Available online: http://www.philippe-fournier-viger.com/spmf/index.php?link=datasets.php (accessed on 4 June 2020).
25. National Center for Atmospheric Research, University Corporation for Atmospheric Research. *Standardized Precipitation Index (SPI) for Global Land Surface (1949–2012)*; National Center for Atmospheric Research, University Corporation for Atmospheric Research: Boulder, CO, USA, 2013.
26. JARTIC. JApan Road Traffic Information Center. 2020. Available online: https://www.jartic.or.jp (accessed on 11 November 2020).
27. Times, T.J. Air Pollution Deaths in Japan. 2019. Available online: https://www.japantimes.co.jp/life/2019/05/11/environment/reading-air-tokyo-still-work-air-pollution (accessed on 12 December 2020).
28. The Ministry of Environment, J. SORAMAME. Available online: http://soramame.taiki.go.jp/ (accessed on 12 December 2020).
29. Kiran, R.U. PAttern MIning-Python Kit (PAMI-PyKit). 2020. Available online: https://github.com/udayRage/pami_pykit/tree/master/traditional/Eclat-pfp (accessed on 4 March 2021).

Short Biography of Authors

Penugonda Ravikumar is currently pursuing a Ph.D. in Computer and information systems at the University of Aizu, Aizu Wakamatsu, Fukushima, Japan on a deputation basis. He is an Assistant Professor in computer science and engineering at the IIIT – RK Valley, Rajiv Gandhi University of Knowledge Technologies, Andhra Pradesh, India. He received his Master of Engineering degree in computer science from the Indian Institute of Science, Bangalore, Karnataka, 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 International Conference on Fuzzy Systems (FUZZ-IEEE), IEEE International Conference on Big Data(IEEE BigData), IEEE Symposium on Computational Intelligence and Data Mining (CIDM), International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems(IEA/AIE), International Conference on Soft Computing and Machine Intelligence (ISCMI).

Palla Likhitha is pursuing B.Tech in Computer science and engineering at the IIIT – RK Valley, Rajiv Gandhi University of Knowledge Technologies, Andhra Pradesh, India. She published papers in IEEE BIG DATA 2020 and International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems(IEA/AIE).

Bathala Venus Vikranth Raj is pursuing B.Tech in Computer science and engineering at the IIIT – RK Valley, Rajiv Gandhi University of Knowledge Technologies, Andhra Pradesh, India. At present, he is working on periodic-frequent pattern mining and spatial pattern mining.

Rage Uday Kiran is currently working as an Associate Professor at the University of Aizu, Aizu Wakamatsu, Fukushima, Japan. He also works as a researcher at the University of Tokyo, Tokyo, Japan. He received his PhD degree in computer science from International Institute of Information Technology, Hyderabad, Telangana, India. His current research interests include data mining, parallel computation, air pollution data analytics, traffic congestion data analytics, recommender systems and ICTs for Agriculture. He has published over 50 papers in refereed journals and international conferences, such as The Conference on Information and Knowledge Management (CIKM), International Conference on Extending Database Technology (EDBT), International Conference on Scientific and Statistical Database Management (SSDBM), The Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD), Database Systems for Advanced Applications (DASFAA), and International Conference on Database and Expert Systems Applications (DEXA).

Yutaka Watanobe is currently a senior associate professor in the School of Computer Science and Engineering, University of Aizu, Japan. His research interests include visual programming language, data mining, and cloud robotics.

Koji Zettsu is a Director General of Big Data Integration Research Center of National Institute of Information and Communications Technology (NICT). He has been doing research and development of data analytics technology in NICT, and now leading Real Space Information Analytics Project since 2016 to implement smart data platform based on data mining and AI. For promoting industry-academia-government collaboration on the platform, he is also a leader of Cross-Data Collaboration Project of Smart IoT Acceleration Forum in Japan. He received Ph.D. in Informatics from Kyoto University in 2005. His research interests are database systems, data mining, information retrieval and software engineering. He has serviced on numerous academic societies, conference committees and working groups.