











Towards Efficient Discovery of Partial Periodic Patterns in Columnar Temporal Databases

Penugonda Ravikumar^{1,4}(✉) , Venus Vikranth Raj⁴ , Palla Likhitha¹ ,
Rage Uday Kiran^{1,2,3} , Yutaka Watanobe¹ , Sadanori Ito² , Koji Zettsu² ,
and Masashi Toyoda³ 

¹ The University of Aizu, Fukushima, Japan
raviua138@gmail.com

² National Institute of Information and Communications Technology, Tokyo, Japan
{ito,zettsu}@nict.go.jp

³ The University of Tokyo, Tokyo, Japan
toyoda@tkl.iis.u-tokyo.ac.jp

⁴ IIIT-RK Valley, RGUKT-Andhar Pradesh, Vempalli, India

Abstract. Finding partial periodic patterns in temporal databases is a challenging problem of great importance in many real-world applications. Most previous studies focused on finding these patterns in row temporal databases. To the best of our knowledge, there exists no study that aims to find partial periodic patterns in columnar temporal databases. One cannot ignore the importance of the knowledge that exists in very large columnar temporal databases. It is because real-world big data is widely stored in columnar temporal databases. With this motivation, this paper proposes an efficient algorithm, Partial Periodic Pattern-Equivalence Class Transformation (3P-ECLAT), to find desired patterns in a columnar temporal database. Experimental results on synthetic and real-world databases demonstrate that 3P-ECLAT is not only memory and runtime efficient but also highly scalable. Finally, we present the usefulness of 3P-ECLAT with a case study on air pollution analytics.

Keywords: Pattern mining · Periodic patterns · Columnar databases

1 Introduction

The big data generated by real-world applications are naturally stored in row or columnar databases. Row databases help the user write the data quickly, while columnar databases facilitate the user to execute fast (aggregate) queries. Thus, row databases are suitable for Online Transaction Processing (OLTP), while columnar databases are suitable for Online Analytical Processing (OLAP). As the objective of knowledge discovery in databases falls under OLAP, this paper aims to find partial periodic patterns in columnar databases.

This research was funded by JSPS Kakenhi 21K12034.

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2022
N. T. Nguyen et al. (Eds.): ACIIDS 2022, LNAI 13758, pp. 141–154, 2022.
https://doi.org/10.1007/978-3-031-21967-2_12

Partial periodic pattern mining [4] is an important knowledge discovery technique in data mining. It involves discovering all patterns in a temporal database that satisfy the user-specified *minimum periodic-support* (*minPS*) and *periodicity* (*per*) constraints. The *minPS* controls the minimum number of periodic occurrences of a pattern in a database. The *per* controls the maximum inter-arrival time of a pattern in the database. A classical application is air pollution analytics. It involves identifying the geographical areas in which people were regularly exposed to harmful air pollutants, say PM2.5. A partial periodic pattern discovered in our air pollution database is as follows:

$$\{1591, 1266, 1250\} \quad [\textit{periodic-support} = 23, \textit{periodicity} = 3 \textit{ h}].$$

The above pattern indicates that the people living close to the sensors, 1591, 1266, and 1250, were frequently and regularly (i.e., at least once every 3 h) exposed to harmful levels of PM2.5. The produced information may help the users for various purposes, such as alerting local authorities and introducing new pollution control policies. (This application was further discussed as a case study in the latter parts of this paper.)

Uday et al. [4] described the Partial Periodic Pattern-growth (3P-growth) algorithm to find desired patterns in a temporal database. It is a depth-first search algorithm that can find partial periodic patterns only in a row database. In other words, this algorithm cannot find partial periodic patterns in a columnar database. One can find partial periodic patterns by transforming a columnar temporal database into a row database. However, we must avoid such naïve transformation process due to its high computational cost. With this motivation, this paper aims to develop an efficient algorithm that can find partial periodic patterns in a columnar database.

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

1. Zaki et al. [8] 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, we cannot directly use this algorithm to find periodic-frequent patterns in a columnar 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 raise 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.

This paper proposes a novel and generic ECLAT algorithm by addressing the above two issues. Our algorithm finds the desired patterns by taking into account the frequency and temporal occurrence information of the items in the data Fig. 1.

The contributions of this paper are as follows: (i) This paper proposes a novel algorithm to find partial periodic patterns in a columnar temporal database. We

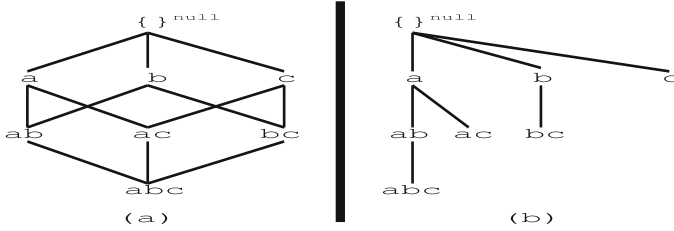


Fig. 1. Search space of the items a, b and c . (a) Itemset lattice and (b) Depth first search on the lattice

call our algorithm as Partial Periodic Pattern-Equivalence Class Transformation (3P-ECLAT). (ii) To the best of our knowledge, this is the first algorithm that aims to find partial periodic 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 partial periodic patterns in a horizontal database. (iii) Experimental results on synthetic and real-world databases demonstrate that our algorithm is not only memory and runtime efficient but also highly scalable. We will also show that 3P-ECLAT even outperforms the state-of-the-art algorithm while finding partial periodic patterns in a row database. (iv) Finally, we describe the usefulness of our algorithm with a case study on air pollution data.

The rest of the paper is organized as follows. Section 2 reviews the work related to our method. Section 3 introduces the model of the partial periodic 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

Agrawal et al. [1] introduced the concept of frequent pattern mining to extract useful information from the transactional databases. It has been used in many domains, and several other algorithms have been developed. Luna et al. [5] 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 database.

Tanbeer et al. [7] introduced the periodic-frequent pattern model to discover temporal regularities in a database. Amphawan et al. [2] extended this model to find top-K periodic-frequent patterns in a database. Amphawan et al. [3] have also discussed novel measuring technique named approximate periodicity to discover periodic-frequent patterns in a transactional database. Nofong et al. [6] have proposed a novel two-stage approach to discover periodic-frequent patterns efficiently. The widespread adoption and industrial application of this model have been hindered by the following limitation: “*Since the objective of $maxPer$ constraint to find all patterns that have maximum inter-arrival time no more than $maxPer$, the model discover only those patterns that have exhibited*

Table 1. Row database **Table 2.** Columnar database **Table 3.** List of ts of an item

ts	items	ts	items
1	ace	7	bcd
2	bc	8	bf
3	bdef	9	abcd
4	abef	10	cd
5	acdf	11	abcd
6	abcd	12	abcd

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

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

full periodic behavior in the database. In other words, this model fails to discover all those interesting patterns that have exhibited partial periodic behavior in the database.” When confronted with this problem in the real-world applications, researchers have tried to find partially occurring periodic-frequent patterns using constraints such as *periodic-ratio*, *standard deviation*, and *average period*. Unfortunately, these extended models require too many input parameters and are not practicable on large databases as the generated patterns do not satisfy the *downward closure property*.

Uday et al. [4] described a novel model to discover partial periodic patterns in a temporal database. Unlike the studies mentioned above, this model is easy to use as it requires only two input parameters, and the generated patterns satisfy the *downward closure property*. A pattern-growth algorithm, called 3P-growth, was also described to find desired patterns in a temporal database. Unfortunately, this algorithm can find partial periodic patterns only in row databases. In this context, this paper aims to advance the state-of-the-art by proposing an algorithm to find partial periodic patterns in a columnar database.

Overall, the proposed algorithm to find partial periodic patterns in a columnar database is novel and distinct from existing studies.

3 The Model of Partial Periodic Pattern

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., $TDB = \{t_1, \dots, t_m\}$, $m = |TDB|$, 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 $TS^X = \{ts_j^X, \dots, ts_k^X\}$, $j, k \in [1, m]$ and $j \leq k$, be an **ordered set of timestamps** where X has occurred in TDB . 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 1. 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 and which is also a binary columnar database. The temporal occurrences of each item in the entire database is 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 2-pattern. The temporal database contains 12 transactions. Therefore, $m = 12$. The minimum and maximum timestamps in this database are 1 and 12, respectively. Therefore, $ts_{min} = 1$ and $ts_{max} = 12$. The pattern ‘ bc ’ appears at the timestamps of 2, 6, 7, 9, 11, and 12. Therefore, the list of timestamps containing ‘ bc ’, i.e., $TS^{bc} = \{2, 6, 7, 9, 11, 12\}$. The support of ‘ bc ’, i.e., $sup(bc) = |TS^{bc}| = 6$.

Definition 1 (Periodic appearance of pattern X). Let $ts_j^X, ts_k^X \in TS^X$, $1 \leq j < k \leq m$, denote any two consecutive timestamps in TS^X . An **inter-arrival time** of X denoted as $iat^X = (ts_k^X - ts_j^X)$. Let $IAT^X = \{iat_1^X, iat_2^X, \dots, iat_k^X\}$, $k = sup(X) - 1$, be the list of all inter-arrival times of X in TDB. An inter-arrival time of X is said to be **periodic** (or interesting) if it is no more than the user-specified period (per). That is, a $iat_i^X \in IAT^X$ is said to be **periodic** if $iat_i^X \leq per$.

Example 2. The pattern ‘ bc ’ has initially appeared at the timestamps of 2 and 6. Thus, the difference between these two timestamps gives an inter-arrival time of ‘ bc .’ That is, $iat_1^{bc} = 4 (= 6 - 2)$. Similarly, other inter-arrival times of ‘ bc ’ are $iat_2^{bc} = 1 (= 7 - 6)$, $iat_3^{bc} = 2 (= 9 - 7)$, $iat_4^{bc} = 2 (= 11 - 9)$, and $iat_5^{bc} = 1 (= 12 - 11)$. Therefore, the resultant $IAT^{ab} = \{4, 1, 2, 2, 1\}$. If the user-specified $per = 2$, then iat_2^{bc} , iat_3^{bc} , iat_4^{bc} and iat_5^{bc} are considered as the periodic occurrences of ‘ bc ’ in the data. In contrast, iat_1^{bc} is not considered as a periodic occurrence of ‘ bc ’ because $iat_1^{bc} \not\leq per$.

Definition 2 (Period-support of pattern X). Let $\widehat{IAT^X}$ be the set of all inter-arrival times in IAT^X that have $iat^X \leq per$. That is, $\widehat{IAT^X} \subseteq IAT^X$ such that if $\exists iat_k^X \in IAT^X : iat_k^X \leq per$, then $iat_k^X \in \widehat{IAT^X}$. The period-support of X , denoted as $PS(X) = |\widehat{IAT^X}|$.

Example 3. Continuing with the previous example, $\widehat{IAT^{bc}} = \{1, 2, 2, 1\}$. Therefore, the period-support of ‘ bc ’, i.e. $PS(bc) = |\widehat{IAT^{bc}}| = |\{1, 2, 2, 1\}| = 4$.

Definition 3 (Partial periodic pattern X). A pattern X is said to be a partial periodic pattern if $PS(X) \geq minPS$, where $minPS$ is the user-specified minimum period-support.

Example 4. Continuing with the previous example, if the user-specified $minPS = 4$, then ‘ bc ’ is a partial periodic pattern because $PS(bc) \geq minPS$. The complete set of partial periodic patterns discovered from Table 3 including 1-patterns (in Fig. 2(f)) are shown in Fig. 3 without “sample” (i.e., Strikethrough) mark on the text.

P	TS-list	PS	TS _i	P	TS-list	PS	TS _i	P	TS-list	PS	TS _i	P	TS-list
a	1	0	1	a	1	0	1	a	1,4,5,6,9,11,12	6	12	b	2,3,4,6,7,8,9,11,12
c	1	0	1	c	12	1	2	c	1,2,5,6,7,9,10,11,12	8	12	c	1,2,5,6,7,9,10,11,12
e	1	0	1	e	1	0	1	e	1,3,4	2	4	d	3,5,6,7,9,10,11,12
				b	2	0	2	b	2,3,4,6,7,8,9,11,12	8	12	a	1,4,5,6,9,11,12
				d				d	3,5,6,7,9,10,11,12	7	12		
				f				f	3,4,5,8	3	8		

Fig. 2. Finding partial periodic patterns. (a) after scanning the first transaction, (b) after scanning the second transaction, (c) after scanning the entire database, and (d) final list of partial periodic patterns sorted in descending order of their *PS* (or the size of TS-list) with the constraint $minPS = 4$ and $per = 2$

Definition 4 (Problem definition). Given a temporal database (*TDB*) and the user-specified period (*per*) and minimum period-support (*minPS*) constraints, find all partial periodic patterns in *TDB* that have period-support no less than *minPS*. The period-support of a pattern can be expressed in percentage of $(|TDB| - 1)$. The *per* can be expressed in percentage of $(ts_{max} - ts_{min})$. In this paper, we employ the above definitions of the period and period-support for brevity.

4 Proposed Algorithm

This section first describes the procedure for finding one-length partial periodic patterns (or 1-patterns) and transforming row database to columnar database. Next, we will explain the 3P-ECLAT algorithm to discover a complete set of partial periodic patterns in columnar temporal databases. The 3P-ECLAT algorithm employs Depth-First Search (DFS) and the *downward closure property* (see Property 1) of partial periodic patterns to reduce the vast search space effectively.

Property 1 (The downward closure property [7]). If *Y* is a partial periodic pattern, then $\forall X \subset Y$ and $X \neq \emptyset$, *X* is also a partial periodic pattern.

4.1 3P-ECLAT Algorithm

Finding One Length Partial Periodic Patterns. Algorithm 1 describes the procedure to find 1-patterns using 3P-list, which is a dictionary. We now describe this algorithm’s working using the row database shown in Table 1. Let $minPS = 4$ and $per = 2$.

We will scan the complete database once to generate 1-patterns and transforming the row database to columnar database. The scan on the first transaction, “1 : ace”, with $ts_{cur} = 1$ inserts the items *a*, *c*, and *e* in the 3P-list. The timestamps of these items is set to 1 ($= ts_{cur}$). Similarly, *PS* and *TS_i* values of

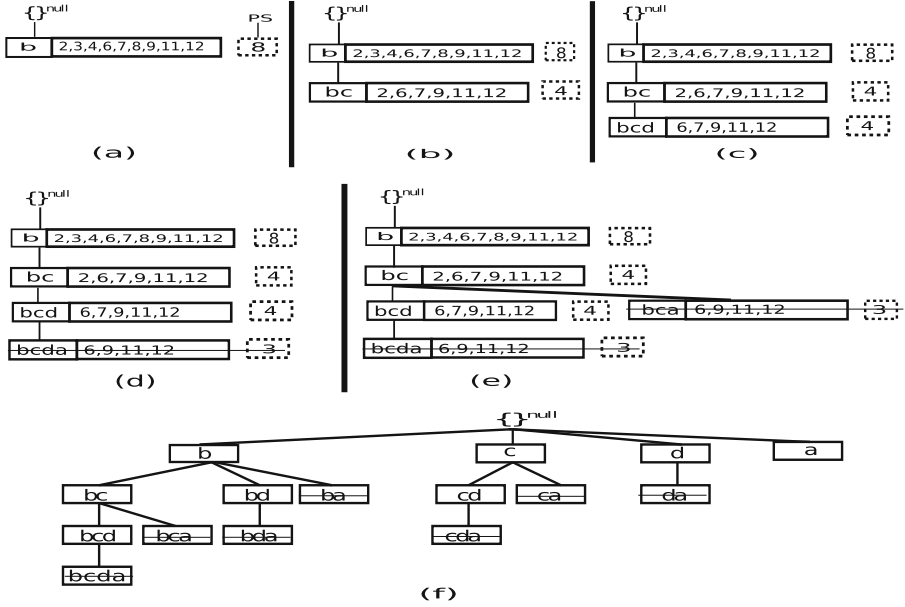


Fig. 3. Mining partial periodic patterns using DFS

these items were also set to 0 and 1, respectively (lines 5 and 6 in Algorithm 1). The 3P-list generated after scanning the first transaction is shown in Fig. 2(a). The scan on the second transaction, “2 : bc”, with $ts_{cur} = 2$ inserts the new item b into the 3P-list by adding 2 ($= ts_{cur}$) in their TS-list. Simultaneously, the PS and TS_l values were set to 0 and 2, respectively. On the other hand, 2 ($= ts_{cur}$) was added to the TS-list of already existing item c with PS and TS_l set to 1 and 2, respectively (lines 7 and 8 in Algorithm 1). The 3P-list generated after scanning the second transaction is shown in Fig. 2(b). A similar process is repeated for the remaining transactions in the database. The final 3P-list generated after scanning the entire database is shown in Fig. 2(c). The pattern e and f are pruned (using the Property 1) from the 3P-list as its PS value is less than the user-specified $minPS$ value (lines 10 and 11 in Algorithm 1). The remaining patterns in the 3P-list are considered partial periodic patterns and sorted in descending order of their PS values. The final 3P-list generated after sorting the partial periodic patterns is shown in Fig. 2(d).

Finding Partial Periodic Patterns Using 3P-list. Algorithm 2 describes the procedure for finding all partial periodic patterns in a database. We now describe the working of this algorithm using the newly generated 3P-list.

We start with item b , which is the first pattern in the 3P-list (line 2 in Algorithm 2). We record its PS , as shown in Fig. 3(a). Since b is a partial periodic 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 1. PartialPeriodicItems(TDB : temporal database, $minPS$: period-Support and per : period)

- 1: Let 3P-list= $(X, TS\text{-list}(X))$ be a dictionary that records the temporal occurrence information of a pattern in a TDB . Let TS_i be a temporary list to record the *timestamp* of the last occurrence of an item in the database. Let PS be a temporary list to record the *periodic-support* of an item in the database. Let i is an item in any transaction $t \in TDB$ and ts_{cur} is current time stamp of any item $i \in t$.
 - 2: **for** each transaction $t \in TDB$ **do**
 - 3: **if** ts_{cur} is i 's first occurrence **then**
 - 4: Insert i and its timestamp into the 3P-list.
 - 5: Set $TS_i[i] = ts_{cur}$ and $PS^i = 0$.
 - 6: **else**
 - 7: Add i 's timestamp in the 3P-list.
 - 8: **if** $(ts_{cur} - TS_i[i]) \leq per$ **then**
 - 9: Set $PS^i ++$.
 - 10: Set $TS_i[i] = ts_{cur}$.
 - 11: **for** each item i in 3P-list **do**
 - 12: **if** $(PS^i < minPS)$ **then**
 - 13: Remove i from 3P-list.
 - 14: Consider the remaining items in 3P-list as partial periodic items. Sort these items in *support* descending order. Let L denote this sorted list of partial periodic items.
-

Algorithm 2). We record PS of bc , as shown in Fig. 3(b). We verify whether bc is partial periodic or uninteresting pattern (line 5 in Algorithm 2). Since bc is partial periodic pattern, we move to its child node bcd and generate its TS-list by performing intersection of TS-lists of bc and d , i.e., $TS^{bcd} = TS^{bc} \cap TS^d$. We record PS bcd , as shown in Fig. 3(c) and identified it as a partial periodic pattern. We once again, move to its child node $bcda$ and generate its TS-list by performing intersection of TS-lists of bcd and a , i.e., $TS^{bcda} = TS^{bcd} \cap TS^a$. As PS of $bcda$ is less than the user-specified $minPS$, we will prune the pattern $bcda$ from the partial periodic patterns list as shown in Fig. 3(d). A similar process is repeated for remaining nodes in the set-enumeration tree to find all partial periodic patterns. The final list of partial periodic patterns generated from Table 1 are shown in Fig. 3(e). The above approach of finding partial periodic patterns using the downward closure property is efficient because it effectively reduces the search space and the computational cost.

5 Experimental Results

In this section, we first compare the 3P-ECLAT against the state-of-the-art algorithm 3P-growth [4] 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 a case study on air pollution data. Please note that 3P-growth ran out of memory on this database. The algorithms 3P-growth and 3P-ECLAT were developed in Python 3.7 and executed on an Intel i5 2.6 GHz,

Table 4. Statistics of the databases

S.No	Database	Type	Nature	Transaction Length			Total transactions
				min	avg	max	
1	Kosarak	Real	Sparse	2	9	2499	990000
2	T20I6d100k	Synthetic	Sparse	1	20	47	199844
3	Congestion	Real	Sparse	1	58	337	17856
4	Pollution	Real	Dense	11	460	971	1438

Algorithm 2. 3P-ECLAT(3P-List)

```

1: for each item  $i$  in 3P-List do
2:   Set  $pi = \emptyset$  and  $X = i$ ;
3:   for each item  $j$  that comes after  $i$  in the 3P-list do
4:     Set  $Y = X \cup j$  and  $Tid^Y = Tid^X \cap Tid^j$ ;
5:     Calculate Period-support of  $Y$ ;
6:     if Period-support  $\geq minPS$  then
7:       Add  $Y$  to  $pi$  and  $Y$  is considered as partial periodic;
8:       Store the Period-support of a partial periodic pattern  $Y$ ;
9:   3P-ECLAT( $pi$ )

```

8GB RAM machine running Ubuntu 18.04 operating system. The experiments have been conducted using synthetic (T20I6d100K) and real-world (Congestion and Pollution) databases. The statistics of all the above databases were shown in Table 4. **The complete evaluation results, databases, and algorithms have been provided through GitHub¹ to verify the repeatability of our experiments.** We are not providing the Congestion databases on GitHub due to confidential reasons.

5.1 Evaluation of Algorithms by Varying $minPS$

In this experiment, we evaluate 3P-growth and 3P-ECLAT algorithms performance by varying only the $minPS$ constraint in each of the databases. The *Per* value in each of the databases will be set to a particular value. The $minPS$ in T20I6d100K, Congestion and Pollution databases has been set at 60%, 50%, and 50%, respectively.

Figure 4 shows the number of partial periodic patterns generated in T20I6d100K, Congestion, and Pollution databases at different $minPS$ values. It can be observed that an increase in $minPS$ has a negative effect on the generation of partial periodic patterns. It is because many patterns fail to satisfy the increased $minPS$.

Figure 5 shows the runtime requirements of 3P-growth and 3P-ECLAT algorithms in T20I6d100K, Congestion, and Pollution databases at different $minPS$ values. It can be observed that even though the runtime requirements of both

¹ https://github.com/udayRage/pykit_old/tree/master/traditional/3peclat.

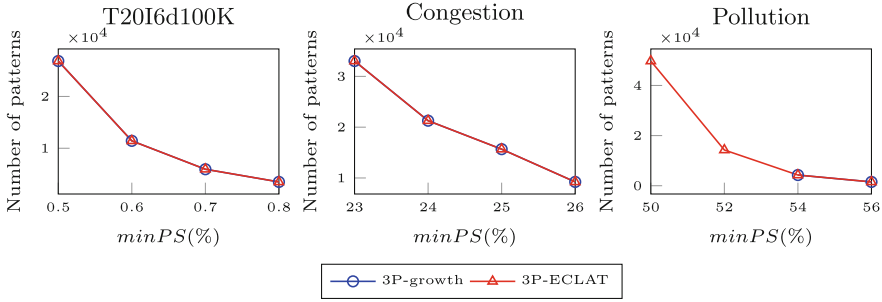


Fig. 4. Patterns evaluation of 3P-growth and 3P-ECLAT algorithms at constant *Per*

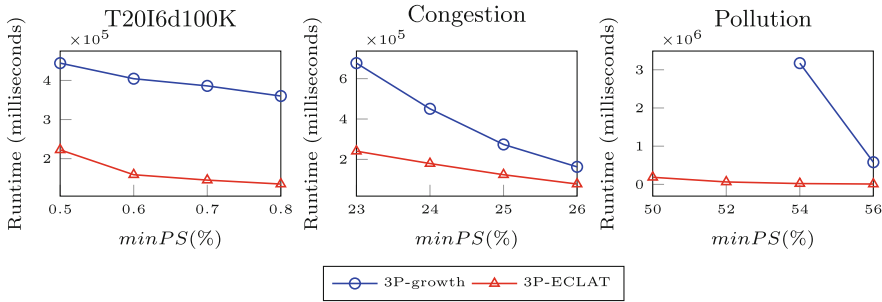


Fig. 5. Runtime evaluation of 3P-growth and 3P-ECLAT algorithms at constant *Per*

the algorithms decrease with the increase in *minPS*, the 3P-ECLAT algorithm completed the mining process much faster than the 3P-growth algorithm in both sparse and dense databases at any given *minPS*. More importantly, the 3P-ECLAT algorithm was several times faster than the 3P-growth algorithm, especially at low *minPS* values.

Figure 6 shows the memory requirements of 3P-growth and 3P-ECLAT algorithms in T20I6d100K, Congestion, and Pollution databases at different *minPS* values. It can be observed that though an increase in *minPS* resulted in the decrease of memory requirements for both the algorithms, the 3P-ECLAT algorithm has consumed relatively very little memory in all databases at different *minPS* values. More importantly, 3P-growth has taken a huge amount of memory, especially at low *minPS* values in all of the databases, and ran out of memory in the Pollution database.

5.2 Evaluation of Algorithms by Varying *Per*

Figure 7 first graph shows the number of partial periodic patterns generated in Congestion database at different *Per* values. It can be observed that an increase in *Per* has increased the number of partial periodic patterns in both of the algorithms.

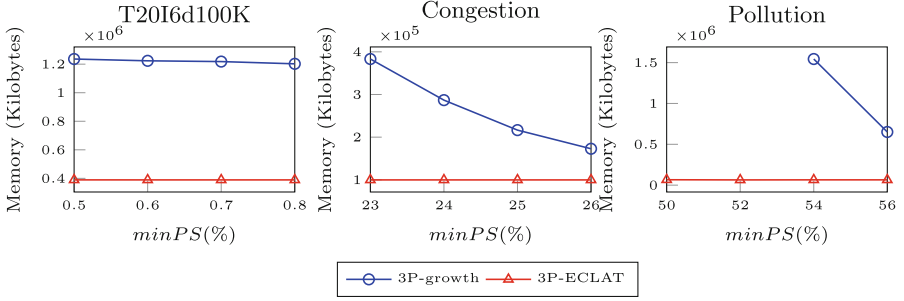


Fig. 6. Memory evaluation of 3P-growth and 3P-ECLAT algorithms at constant Per

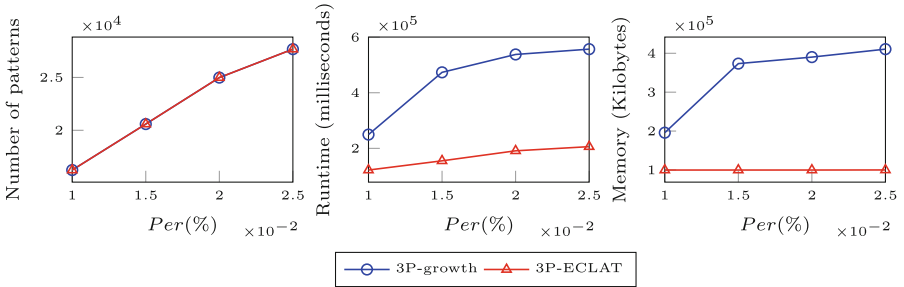


Fig. 7. Evaluation of 3P-growth and 3P-ECLAT algorithms using Congestion database

Figure 7 second graph shows the runtime requirements of 3P-growth and 3P-ECLAT algorithms in Congestion database at different Per values. It can be observed that though the runtime requirements of both the algorithms increase with the increase in Per value, the 3P-ECLAT algorithm consumes relatively less runtime than the 3P-growth algorithm.

Figure 7 third graph shows the memory requirements of 3P-growth and 3P-ECLAT algorithms in Congestion database at different Per values. It can be observed that though the memory requirements of both the algorithms increase with Per , the 3P-ECLAT algorithm consumes very less memory than the 3P-growth algorithm.

The $minPS$ is set at 23% during the above evaluation. Similar results were obtained during the experimentation on remaining databases. However, we have confined this experiment to the Congestion database due to page limitations.

5.3 Scalability Test

The Kosarak database was divided into five portions of 0.2 million transactions in each part, in order to check the performance of 3P-ECLAT against 3P-growth. We have investigated the performance of 3P-growth and 3P-ECLAT algorithms after accumulating each portion with previous parts. Figure 8 shows the runtime

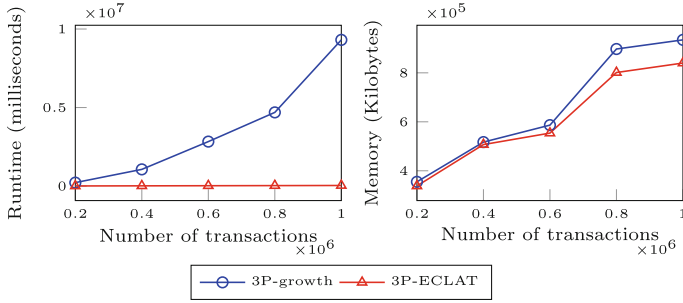


Fig. 8. Scalability of 3P-growth and 3P-ECLAT

and memory requirements of both algorithms at different database sizes (i.e., increasing order of the size) when $minPS = 1$ (%) and $Per = 1$ (%). The following two observations can be drawn from these figures: (i) Runtime and memory requirements of 3P-growth and 3P-ECLAT algorithms increase almost linearly with the increase in database size. (ii) At any given database size, 3P-ECLAT consumes less runtime and memory as compared against the 3P-growth algorithm.

5.4 A Case Study: 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 SORAMAME, to monitor air pollution throughout Japan, is shown in Fig. 9(a). The raw data produced by these sensors i.e., quantitative columnar database (see Fig. 9(b)) can be transformed into a binary columnar database, if the raw data value is ≥ 15 (see Fig. 9(c)). The transformed data is provided to 3P-ECLAT algorithm (see Fig. 9(d)) to identify all sets of sensor identifiers in which pollution levels are high (see Fig. 9(e)). The spatial locations of interesting patterns generated from the **Pollution** database are visualized in Fig. 9(f). 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.

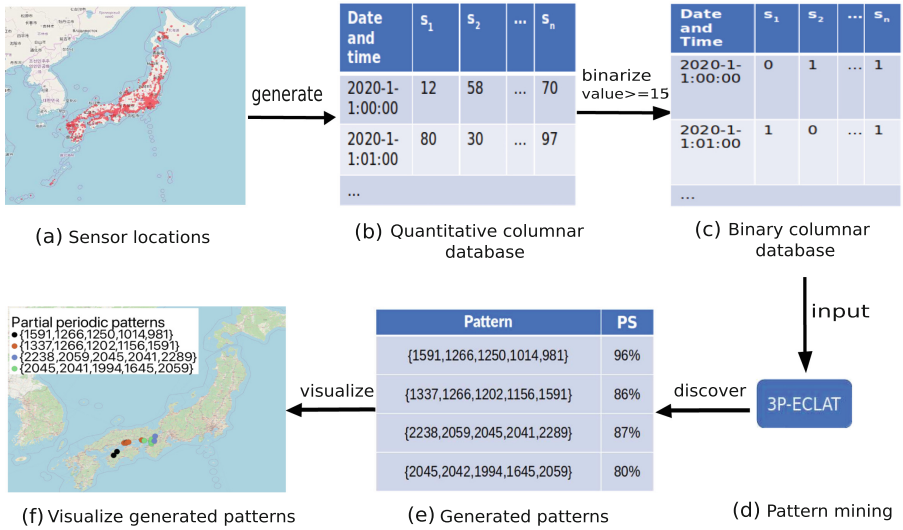


Fig. 9. Finding partial periodic patterns in Pollution data. The terms ‘ s_1 ,’ ‘ s_2 ,’ ... ‘ s_n ’ represents ‘sensor identifiers’ and ‘ PS ’ represents ‘periodic-support’

6 Conclusions and Future Work

This paper has proposed an efficient algorithm named Partial Periodic Pattern-Equivalence Class Transformation (3P-ECLAT) to find partial periodic patterns in columnar temporal databases. The performance of the 3P-ECLAT is verified by comparing it with a 3P-growth algorithm on different real-world and synthetic databases. Experimental analysis shows that 3P-ECLAT exhibits high performance in partial periodic pattern mining and can obtain all partial periodic patterns faster and with less memory usage against the state-of-the-art algorithm. We have also presented a case study to illustrate the usefulness of generated patterns in a real-world application.

As part of future work, we would like to investigate parallel algorithms to find periodic and fuzzy partial periodic patterns in very large temporal databases. We will try to extend our model to the distributed environment and develop novel pruning techniques to reduce the computation cost further.

References

1. Agrawal, R., Imieliński, T., Swami, A.: Mining association rules between sets of items in large databases. In: SIGMOD, pp. 207–216 (1993)
2. Amphawan, K., Lenca, P., Surarerks, A.: Mining top-k periodic-frequent pattern from transactional databases without support threshold. In: Papasratorn, B., Chutimaskul, W., Porkaew, K., Vanijja, V. (eds.) IAIT 2009. CCIS, vol. 55, pp. 18–29. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-10392-6_3

3. Amphawan, K., Surarerks, A., Lenca, P.: Mining periodic-frequent itemsets with approximate periodicity using interval transaction-ids list tree. In: 2010 Third International Conference on Knowledge Discovery and Data Mining, pp. 245–248 (2010)
4. Kiran, R.U., Shang, H., Toyoda, M., Kitsuregawa, M.: Discovering partial periodic itemsets in temporal databases. In: Proceedings of the 29th International Conference on Scientific and Statistical Database Management. SSDBM '17 (2017)
5. Luna, J.M., Fournier-Viger, P., Ventura, S.: Frequent itemset mining: a 25 years review. *Wiley Interdiscip. Rev. Data Min. Knowl. Discov.* **9**(6), e1329 (2019)
6. Nofong, V.M., Wondoh, J.: Towards fast and memory efficient discovery of periodic frequent patterns. *J. Inf. Telecommun.* **3**(4), 480–493 (2019)
7. Tanbeer, S.K., Ahmed, C.F., Jeong, B.-S., Lee, Y.-K.: Discovering periodic-frequent patterns in transactional databases. In: Theeramunkong, T., Kijssirikul, B., Cercone, N., Ho, T.-B. (eds.) PAKDD 2009. LNCS (LNAI), vol. 5476, pp. 242–253. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-01307-2_24
8. Zaki, M.J.: Scalable algorithms for association mining. *IEEE Trans. Knowl. Data Eng.* **12**(3), 372–390 (2000)