

A DISSERTATION  
SUBMITTED IN FULFILLMENT OF THE REQUIREMENTS  
FOR THE DEGREE OF DOCTOR OF PHILOSOPHY  
IN COMPUTER SCIENCE AND ENGINEERING

**Research and Development of Fast Algorithms for  
Pattern Discovery in Binarized Spatiotemporal  
Databases**



by

Penugonda Ravikumar

*September 2023*

© Copyright by Penugonda Ravikumar , September 2023

All Rights Reserved.



The thesis titled

Contents

**Research and Development of Fast Algorithms for Pattern Discovery in  
Binarized Spatiotemporal Databases**

by

**Penugonda Ravikumar**

is reviewed and approved by:

---

**Chief referee**

**Associate Professor**

Date 11-8-2023

**Rage Uday Kiran**

*R. eudaykiran*



---

**Senior Associate Professor**

Date

2023/8/14

**Yutaka Watanobe**

*Yutaka Watanobe*



---

**Senior Associate Professor**

Date

**TRUONG Cong Thang**



---

**Professor**

Date

2023/8/15

**MARKOV Konstantin**

*K. Markov*



THE UNIVERSITY OF AIZU

September 2023

# Contents

<b>Chapter 1 Introduction</b>	<b>1</b>
1.1 Discovering interesting patterns from temporal databases . . . . .	3
1.1.1 Periodic-Frequent Pattern Model . . . . .	4
1.1.2 Contributions . . . . .	5
Discovering periodic-frequent patterns in columnar temporal databases	6
Discovering partial periodic patterns in columnar temporal databases	6
Discovering geo-referenced periodic frequent patterns in geo-referenced time series databases . . . . .	6
1.2 Related work . . . . .	7
1.2.1 Literature review on periodic-frequent pattern mining . . . . .	7
1.2.2 Literature review on partial periodic pattern mining . . . . .	9
1.2.3 Literature review on spatial co-occurrence pattern mining . . . . .	13
1.3 Organization . . . . .	14
<b>Chapter 2 Efficient Discovery of Periodic-Frequent Patterns in Columnar Temporal Databases</b>	<b>16</b>
2.1 Proposed Algorithm . . . . .	18
2.1.1 PF-ECLAT algorithm . . . . .	18
Finding one length periodic-frequent patterns . . . . .	18
Finding periodic-frequent patterns using PFP-list. . . . .	20
2.1.2 Time complexity analysis . . . . .	23
2.2 Experimental Results . . . . .	23
2.2.1 Experimental setup . . . . .	23
2.2.2 Evaluation of PFP-growth, PFP-growth++, PS-growth, and PF-ECLAT algorithms by varying <i>maxPer</i> constraint . . . . .	25
2.2.3 Evaluation of PFP-growth, PFP-growth++, PS-growth, and PF-ECLAT algorithms by varying <i>minSup</i> constraint . . . . .	27
2.2.4 Scalability test . . . . .	31
2.2.5 A case study 1: finding areas where people have been regularly exposed to hazardous levels of PM2.5 pollutant . . . . .	32
2.2.6 A Case Study on Traffic Congestion Analytics . . . . .	33
2.3 Discussion about PAMI package . . . . .	33
2.4 Conclusions . . . . .	33
<b>Chapter 3 Towards Efficient Discovery of Partial Periodic Patterns in Columnar Temporal Databases</b>	<b>36</b>
3.1 The Model of Partial Periodic Pattern . . . . .	37
3.2 Proposed Algorithm . . . . .	38
3.2.1 3P-ECLAT algorithm . . . . .	38

	Finding one length partial periodic patterns . . . . .	38
	Finding partial periodic patterns using the 3P-list. . . . .	40
3.3	Experimental Results . . . . .	41
3.3.1	Evaluation of algorithms by varying <i>minPS</i> . . . . .	41
3.3.2	Evaluation of algorithms by varying <i>per</i> . . . . .	42
3.3.3	Scalability test . . . . .	43
3.3.4	A Case Study: Finding Areas Where People Have Been Regularly Exposed to Hazardous Levels of PM2.5 Pollutant . . . . .	44
3.4	Discussion about PAMI package . . . . .	45
3.5	Conclusions . . . . .	45
<b>Chapter 4 Discovering Geo-referenced Periodic-Frequent Patterns in Geo-referenced Time Series Databases</b>		<b>46</b>
4.1	Proposed Model . . . . .	47
4.1.1	Location database and time series database . . . . .	47
4.1.2	Model of Geo-referenced Periodic-Frequent Patterns . . . . .	48
4.2	Proposed algorithm . . . . .	49
4.2.1	Naïve algorithm . . . . .	49
4.2.2	Basic idea . . . . .	50
4.2.3	GFPF-Miner . . . . .	51
	Finding one-length geo-referenced periodic-frequent patterns . . . . .	51
	Finding geo-referenced periodic-frequent patterns using the GFPF-list and neighboring lists. . . . .	52
4.3	Experimental results . . . . .	53
4.3.1	Experimental setup . . . . .	53
4.3.2	Evaluation of PF-ECLAT and GFPF-Miner algorithms by varying <i>minSup</i> constraint . . . . .	55
4.3.3	Evaluation of PF-ECLAT and GFPF-Miner algorithms by varying <i>maxPer</i> constraint . . . . .	56
4.3.4	A case study 1: air pollution analytics . . . . .	57
4.3.5	A case study 2: traffic congestion analytics . . . . .	58
4.4	Discussion about PAMI package . . . . .	58
4.5	Conclusions . . . . .	60
<b>Chapter 5 Conclusion</b>		<b>61</b>
5.1	Conclusion . . . . .	61
5.2	Future Research . . . . .	62

# List of Figures

Figure 2.1	Sample representation of itemset space. (a) representing the items $a$ , $b$ , and $c$ using a lattice structure (b) DFS-based evaluation of the itemsets . . . . .	17
Figure 2.2	Initially, the PFP-list is empty. After scanning the first transaction, we insert each item into the PFP-list, as seen in (a). After scanning the second transaction, we add a new item to the PFP-list and update the values of the existing items, as shown in (b). After scanning the third transaction, we update the values of the existing items, as shown in (c). After scanning the fourth transaction, we add a new item to the PFP-list and update the values of the existing items, as shown in (d). After scanning the whole database, we obtain the final 3P-list as shown in (e). Finally, we sort the one-length Periodic-Frequent Patterns ( $PFPs$ ) in descending order of their <i>support</i> , as shown in (h). . . . .	19
Figure 2.3	Mining $PFPs$ using Depth-First Search (DFS). . . . .	20
Figure 2.4	The runtime performance of PFP-growth, PFP-growth++, PS-growth, and PF-ECLAT algorithms, was compared across multiple databases with a fixed value of <i>maximum periodicity</i> ( $maxPer$ ) and varying values of <i>minimum support</i> ( $minSup$ ). . . . .	27
Figure 2.5	The memory usage of PFP-growth, PFP-growth++, PS-growth, and PF-ECLAT algorithms, was compared across multiple databases with a fixed value of $maxPer$ and varying values of $minSup$ . . . . .	27
Figure 2.6	The number of $PFPs$ generated by the PFP-growth, PFP-growth++, PS-growth, and PF-ECLAT algorithms, was compared across multiple databases with a fixed value of $maxPer$ and varying values of $minSup$ . . . . .	28
Figure 2.7	The runtime performance of PFP-growth, PFP-growth++, PS-growth, and PF-ECLAT algorithms, was compared across multiple databases with a fixed value of $minSup$ and varying values of $maxPer$ . . . . .	28
Figure 2.8	The memory usage of PFP-growth, PFP-growth++, PS-growth, and PF-ECLAT algorithms, was compared across multiple databases with a fixed value of $minSup$ and varying values of $maxPer$ . . . . .	29
Figure 2.9	The number of patterns generated by the PFP-growth, PFP-growth++, PS-growth, and PF-ECLAT algorithms, was compared across multiple databases with a fixed value of $minSup$ and varying values of $maxPer$ . . . . .	29
Figure 2.10	Scalability of PFP-growth, PFP-growth++, PS-growth, and PF-ECLAT . . . . .	30
Figure 2.11	Finding $PFPs$ in the Pollution database. The terms ' $s_1$ ,' ' $s_2$ ,' $\dots$ ' $s_n$ ' represents 'sensor identifiers' . . . . .	31

Figure 2.12	Finding <i>PFPs</i> in the Congestion database. The terms ‘ $s_1$ ,’ ‘ $s_2$ ,’ ... ‘ $s_n$ ’ represents ‘road sensor identifiers’	32
Figure 2.13	Explanation about the equal contribution statement in my journal paper	35
Figure 3.1	Finding Partial Periodic Patterns ( <i>3Ps</i> ). (a) after scanning the first transaction, (b) after scanning the second transaction, (c) after scanning the entire database, and (d) final list of <i>3Ps</i> sorted in descending order of their <i>period-support</i> ( <i>PS</i> ) (or the size of TS- list) with the constraint <i>minimum period-support</i> ( <i>minPS</i> )= 4 and <i>period</i> ( <i>per</i> ) = 2	39
Figure 3.2	Mining <i>3Ps</i> using DFS.	39
Figure 3.3	Patterns evaluation of 3P-growth and Partial Periodic pattern-Equivalence Class Transformation (3P-ECLAT) algorithms at constant <i>per</i>	42
Figure 3.4	Runtime evaluation of 3P-growth and 3P-ECLAT algorithms at constant <i>per</i>	42
Figure 3.5	Memory evaluation of 3P-growth and 3P-ECLAT algorithms at constant <i>per</i>	43
Figure 3.6	Evaluation of 3P-growth and 3P-ECLAT algorithms using Con- gestion database	43
Figure 3.7	Scalability of 3P-growth and 3P-ECLAT	44
Figure 3.8	Finding <i>3Ps</i> in Pollution data. The terms ‘ $s_1$ ,’ ‘ $s_2$ ,’ ... ‘ $s_n$ ’ rep- represents ‘sensor identifiers’ and ‘ <i>PS</i> ’ represents ‘period-support’	44
Figure 4.1	Air pollution analytics. The terms ‘SensorID’ and ‘ $PM_{2.5}$ ’ repre- sent ‘Sensor IDentifier’ and ‘value of the air pollutant,’ respectively	47
Figure 4.2	Finding one-length Geo-referenced Periodic-Frequent Patterns ( <i>GPFPS</i> ). (a) Content of the list after reading the 1 <sup>st</sup> transaction, (b) af- ter reading the 2 <sup>nd</sup> transaction, (c) Final content after reading the whole database, and (d) The complete list of one-length <i>GPFPS</i> sorted in increasing order of <i>support</i> (or the size of ts-list).	52
Figure 4.3	Mining of <i>GPFPS</i> .	53
Figure 4.4	Patterns evaluation of PF-ECLAT and GPFPS-Miner algorithms at constant <i>maxPer</i>	56
Figure 4.5	Runtime evaluation of PF-ECLAT and GPFPS-Miner algorithms at constant <i>maxPer</i>	56
Figure 4.6	Memory evaluation of PF-ECLAT and GPFPS-Miner algorithms at constant <i>maxPer</i>	57
Figure 4.7	Patterns evaluation of PF-ECLAT and GPFPS-Miner algorithms at constant <i>minSup</i>	57
Figure 4.8	Runtime evaluation of PF-ECLAT and GPFPS-Miner algorithms at constant <i>minSup</i>	58
Figure 4.9	Memory evaluation of PF-ECLAT and GPFPS-Miner algorithms at constant <i>minSup</i>	58
Figure 4.10	Interestingness of our patterns	59
Figure 4.11	Interestingness of our patterns	59

# List of Tables

Table 1.1	Chetan’s supermarket row database . . . . .	2
Table 1.2	Chetan’s supermarket binary columnar database . . . . .	2
Table 1.3	Chetan’s supermarket temporal database . . . . .	3
Table 1.4	Chetan’s columnar temporal database . . . . .	3
Table 1.5	Row database . . . . .	5
Table 1.6	Columnar database . . . . .	5
Table 2.1	Statistics of the databases . . . . .	24
Table 3.1	Row database . . . . .	37
Table 3.2	Columnar database . . . . .	37
Table 3.3	List of <i>ts</i> of an item . . . . .	37
Table 3.4	Statistics of the databases . . . . .	41
Table 4.1	Location (or Geo-referential) database . . . . .	47
Table 4.2	Time series database. The items whose values were equal to 0 at a particular timestamp were removed for brevity . . . . .	48
Table 4.3	Neighbors . . . . .	49

# List of Abbreviations

<i>3P</i>	Partial Periodic Pattern
3P-ECLAT	Partial Periodic pattern-Equivalence CLass Transformation
<i>3Ps</i>	Partial Periodic Patterns
AP analytics	Air Pollution analytics
CDB	Columnar database
CDBs	Columnar databases
CTDB	Columnar Temporal database
CTDBs	Columnar Temporal databases
DFS	Depth-First Search
FPM	Frequent Pattern Mining
<i>GPFP</i>	Geo-referenced Periodic-Frequent Pattern
GPFP-Miner	Geo-referenced Periodic-Frequent Pattern-Miner
<i>GPFPs</i>	Geo-referenced Periodic-Frequent Patterns
<i>GTSD</i>	Geo-referenced Time Series Database
<i>GTSDs</i>	Geo-referenced Time Series Databases
ID	identifier
JARTIC	Japan Road Traffic Information Center
LD	Location Database (or geo-referenced database)
<i>maxDist</i>	<i>maximum distance</i>
<i>maxPer</i>	<i>maximum periodicity</i>
MB analytics	Market Basket analytics
<i>minPS</i>	<i>minimum period-support</i>
<i>minSup</i>	<i>minimum support</i>
<i>per</i>	<i>period</i>
PF-ECLAT	Periodic Frequent-Equivalence CLass Transformation
<i>PF</i>	Periodic-Frequent Pattern
PFPM	Periodic-Frequent Pattern Mining
<i>PFs</i>	Periodic-Frequent Patterns
<i>PPFPs</i>	Partial Periodic-Frequent Patterns
PPs	Periodic (either full-periodic or partial periodic) patterns
<i>PS</i>	<i>period-support</i>
RDB	Row database
RDBs	Row databases
RTDB	Row Temporal database
RTDBs	Row Temporal databases
SPP	segment-wise periodic pattern
SPPs	segment-wise periodic patterns

ST	Spatiotemporal
<i>STD</i>	spatiotemporal database
<i>STDs</i>	spatiotemporal databases
TC analytics	Traffic Congestion analytics
TD	Transactional database
TDB	Temporal database (or time series database)
TDBs	Temporal databases (or time series databases)
TDs	Transactional databases
<i>ts</i>	timestamp
TSD	Time Series Database
TSDs	Time Series Databases



# List of Symbols

$a$	Bread
$b$	Milk
$c$	Books
$d$	Pen
$e$	Bat
$f$	Ball
$i_j$	Generalized symbolic representation of a $j^{th}$ item in set $I$
$Coor_{i_j}$	Spatial coordinates of the item $i_j$
$I$	Set of items
$m$	Represents the total size of the database
$ts_k^X$	Represents the $t_k$ of the pattern $X$ in a database
$p_a^X$	Represents the time difference (or inter-arrival time) between $ts_r^X$ and $ts_q^X$ of the pattern $X$ in TDB
$iat^X$	Represents the inter-arrival time between $ts_j^X$ and $ts_k^X$ of the pattern $X$ in TDB
$iat_k^X$	Represents the $k^{th}$ entry of the $iat^X$ from the set $IAT^X$
$n$	Represents the total number of items in a database
$p$	sensor1
$q$	sensor2
$\mathbb{R}+$	Set of real numbers
$r$	sensor3
$s$	sensor4
$t$	sensor5
$TS^{bc}$	The set of all timestamp ( $ts$ ) where the set of items $b$ and $c$ occurs in TDB
$IAT^{bc}$	The set of all inter-arrival times where the set of items $b$ and $c$ occurs in TDB
$\widehat{IAT^X}$	the set of all inter-arrival times in $IAT^X$ that have $iat^X \leq per$ .
$\widehat{IAT^{bc}}$	the set of all inter-arrival times in $IAT^{bc}$ that have $iat^{bc} \leq per$ .
$tid$	transaction identifier in a database
$t_k$	Represents the $k^{th}$ $tid$ in the case of TD (or represents the $k^{th}$ $ts$ in the case of TDB )
$TS^{rs}$	The set of all $ts$ where the set of items $r$ and $s$ occurs in TSD
$TS^Y$	The set of all $ts$ where $Y$ occurs in TDB
$TS^Z$	The set of all $ts$ where $Z$ occurs in TDB

$IAT^X$	The set of all $iat^X$ of a pattern $X$ in TDB
$TS^X$	The set of all $ts$ where $X$ occurs in TDB
$P^X$	The set of all <i>periods</i> of a pattern $X$
$u$	sensor6
$X$	pattern or itemset
$Y$	pattern or itemset
$Z$	pattern or itemset

# Acknowledgment

I am deeply grateful to my research supervisor, Prof. Rage Uday Kiran, for his invaluable mentorship throughout my research journey. Prof. Rage's systematic thinking, problem-solving methods, and drafting skills have been instrumental in shaping me as a researcher. I could not have completed this PhD thesis work without his love, affectionate guidance, and unwavering support. I am particularly thankful for his immense patience, which has allowed me to grow and learn at my own pace.

I would like to express my sincere gratitude to my research co-supervisor, Prof. Yutaka Watanobe, for his invaluable guidance, feedback, and support throughout my research journey. Without his contribution, this PhD thesis would not have been possible.

I would also like to extend my gratitude to the members of my thesis committee, Prof. TRUONG Cong Thang and Prof. MARKOV Konstantin, for their constructive feedback and insightful comments on my work. Their input has been instrumental in improving the quality of my research and helping me refine my ideas.

I would like to thank the Japan Student Services Organization (JASSO) for their financial support and all the teaching and non-staff members of the University of AIZU for their valuable guidance during my stay in Japan.

I am grateful to my colleagues Dr. Md. Mostafizer Rahman and Mr. Raihan Kabir Riad, Miss Palla Likhitha, Miss Rasheeka, Mr. P. Sandeep, all my lab mates, and my friends Mr. Y. Sunil Chand and Mr. D.G. Naresh Babu for their constant support and encouragement and for providing a stimulating and enjoyable research environment. Their willingness to share their knowledge and expertise has been invaluable in broadening my horizons and deepening my understanding of the subject.

Last but not least, I would like to thank my father, Mr. P. Ramanjaneyulu (late), my mother, Mrs. P. Venkatalakshumma, my brother, Mr. P. Anjan Kumar, my brother's wife, Mrs. P. Chandrika, my brother's son, Mr. P. Hanuma Bhavit, my mother-in-law, Mrs. K. Padmavathi, my brother-in-law, K. Venkata Mahesh (late), my father, Mr. Their love, patience, and understanding have sustained me through the ups and downs of this challenging process, and I could not have done it without them. Lastly, to my wife, Mrs. Koneti Hema Latha, the love of my life and my constant source of inspiration during this journey: I am forever grateful to all of you for your love and for being my partner in this incredible adventure called life.

# Abstract

In the modern world, a huge amount of data comes from various sources, such as social media, financial systems, and scientific experiments. Data mining is introduced as a field that utilizes statistical and computational techniques to find patterns and relationships in these data. Research areas within data mining include classification, clustering, association rule mining, time series analysis, and big data analytics. These research areas in data mining are not exhaustive, but finding periodic patterns in large spatiotemporal databases (*STDs*) is one of the most significant areas in data mining. Discovering periodic patterns in *STDs* is important because it allows us to identify regularities and recurring trends in complex systems over time and space. Many researchers are dedicating their efforts to developing more effective methods for identifying periodic patterns in *STDs*. In this thesis, we have made an effort to propose novel and fast techniques to discover interesting patterns in large *STDs*. The initial two chapters of the thesis focused solely on the temporal component of the database while neglecting the spatial aspect. However, in the final chapter of the thesis, both the spatial and temporal components were taken into account for analysis and interpretation.

Frequent pattern mining is a data mining technique that aims to discover patterns that frequently occur together in a set of transactions based on a user-specified *minimum support* (*minSup*) constraint. The *minSup* determines the minimum number of transactions that a pattern must cover in the database. Frequent pattern mining has many applications, but its adoption and effectiveness are hindered by two primary challenges. The first challenge is that frequent pattern mining algorithms often generate a large number of patterns, many of which may not be useful or relevant to the user's needs or application requirements. The second challenge is that most frequent pattern mining algorithms do not consider the temporal behavior of patterns or the timestamp (*ts*) associated with transactions in the database. To address these challenges in real-world applications, researchers have generalized the frequent pattern model to enable the discovery of periodic patterns in *STDs*. This thesis proposes a generalized approach that aims to identify interesting patterns in an spatiotemporal database (*STD*) that meet the specific criteria set by the user in terms of *minSup* and *maximum periodicity* (*maxPer*). The *maxPer* constraint enables the regulation of the longest duration for which a pattern has to repeat in the data. Several novel and efficient algorithms were proposed to extract all interesting patterns from a *STD* at regular intervals. In addition, we have also explored the spatial information of the patterns to discover interesting patterns in a *STD* that meet the specific criteria set by the user in terms of *minSup*, *maxPer*, and *maximum distance* (*maxDist*). The *maxDist* controls how far apart the items in a pattern can be. Through our exploration, we have discovered that patterns where items are adjacent are more appealing to users than those where the items are not adjacent to each other. This indicates that adjacency or proximity is an important factor when considering the relevance and usefulness of patterns in *STDs*.

We have introduced a novel approach to extracting interesting patterns from *STDs*, which incorporates the spatial relationship between items.

In this thesis, we propose three approaches for discovering periodic patterns in *STDs*. The first two approaches are focused on identifying periodic (both full periodic and partial periodic) patterns in *STDs*, while the third approach takes into account the spatial information of the patterns and discovers *GPFPs*.

We have seen that row and columnar databases are two different types of database architectures for storing and organizing data. Both architectures have their own merits and demerits. As a result, there is no universally accepted best database architecture for any application. In this thesis, we have only used columnar database architecture, as it is well suited for massive databases. Firstly, we have developed a novel Periodic Frequent-Equivalence Class Transformation (PF-ECLAT), which is an ECLAT-like algorithm to discover *PFPs* in *STDs*. By utilizing depth-first search and applying the *downward closure property* of *PFPs*, the PF-ECLAT algorithm can efficiently reduce the extensive search space. Secondly, we claim that PF-ECLAT generates the full-periodic patterns (or *PFPs*), whereas in the real world there is a huge importance to the *3Ps* though they occur only during specific times. Therefore, we introduce a novel approach of *3Ps* and propose an algorithm named 3P-ECLAT to discover *3Ps* in *STDs*. Thirdly, we claim that earlier approaches overlooked the spatial (or geo-referenced) information of the patterns, and this information is also highly important. Therefore, we have considered the geo-referenced information of the patterns and introduced an algorithm named Geo-referenced Periodic-Frequent Pattern-Miner (GPFP-Miner) to discover *GPFPs* in *STDs*.

The efficiency of the proposed algorithms is shown by conducting extensive experiments on both synthetic and real-world databases. Finally, we demonstrate the practical applications of the proposed approaches through several case studies on Traffic Congestion analytics (TC analytics) and Air Pollution analytics (AP analytics). These case studies showcase the real-world applications of the proposed approaches, demonstrating how they can be used to analyze traffic congestion and air pollution data to provide insights into patterns, trends, and potential solutions.

Overall, it has been demonstrated that the proposed algorithms extract interesting patterns in large *STDs* more efficiently.

# Chapter 1

## Introduction

Technological advancements in computer science and information technologies have empowered real-world applications to produce massive data. This data is traditionally stored and processed in databases. Depending on the data storage format, these databases were broadly classified into row or columnar databases [1, 2]. The row databases are designed to store data in rows, with each row representing a single record. These databases are very good at handling small to medium-sized databases, making them ideal for applications with limited records. Furthermore, these databases prioritize ACID<sup>1</sup> properties, ensuring data reliability and consistency. Examples of row databases include MySQL [3] and Postgres [4]. On the other hand, columnar databases are designed to store data in columns instead of rows. This database architecture is well-suited for applications that process large amounts of data quickly, such as data warehousing and analytics. These databases are also very good at handling large databases, making them ideal for applications with a high volume of records. Furthermore, these databases rely on BASE<sup>2</sup> properties, prioritizing the availability and scalability of data. Examples of columnar databases include Snowflake [5] and BigQuery [6]. Since row and columnar databases offer unique advantages and disadvantages, selecting an appropriate database depends on user and/or application requirements. In general, row databases are better suited for Online Transaction Processing (OLTP) applications, while columnar databases are better suited for Online Analytical Processing (OLAP) applications. Since data mining is a part of OLAP, a supportive move has been made in this research to find interesting patterns hidden in a columnar database.

Useful information that can empower the users with competitive information to achieve socio-economic development lies hidden in these data. When confronted with this problem, researchers introduced the field of data mining to discover knowledge from big data. Data mining primarily consists of the following four techniques: *(i)* clustering [7] is a fundamental concept in unsupervised learning, where patterns and structures are automatically identified in data without the need for predefined labels or target outcomes, *(ii)* outlier detection [8] involve grouping similar data and identifying anomalies that do not belong to any of the groups, *(iii)* classification [9] is a fundamental concept in supervised learning, where labeled data is used to train models that can accurately classify new, unseen instances, and *(iv)* pattern mining [10, 11] discovers relationships or associations between different items in a database. This thesis, focuses on addressing the issues encountered by pattern mining techniques.

---

<sup>1</sup>ACID stands for Atomicity, Consistency, Isolation, and Duration

<sup>2</sup>BASE stands for Basically Available, Soft state, and Eventually consistent

Table 1.1: Chetan’s supermarket row database

<i>tid</i>	Items purchased
100	<i>bread, milk</i>
101	<i>books, pen</i>
102	<i>bat, ball</i>
103	<i>bread, butter</i>
104	<i>bread, milk</i>

Table 1.2: Chetan’s supermarket binary columnar database

<i>tid</i>	Items						
	<i>bread</i>	<i>milk</i>	<i>books</i>	<i>pen</i>	<i>bat</i>	<i>ball</i>	<i>butter</i>
100	1	1	0	0	0	0	0
101	0	0	1	1	0	0	0
102	0	0	0	0	1	1	0
103	1	0	0	0	0	0	1
104	1	1	0	0	0	0	0

A transactional database represents an unordered set of transactions. A transaction is an unordered set of items or objects. Agrawal et al. [12] introduced the frequent pattern model to discover regularities in a database. This model involves finding all patterns (or itemsets) in a database that satisfy the user-specified *minimum support* (*minSup*) threshold. The *minSup* regulates the threshold for the minimum occurrence of a pattern within a database, specifically in terms of the number of transactions it must be present in. A popular example of frequent pattern mining is customer purchase history analytics. It involves determining how frequently the customers have purchased items in an e-commerce store.

**Example 1.** Let Chetan be a supermarket selling *bread, milks, books, pen, bat, ball, and butter*. The purchase history of anonymous customers in row database format is shown in Table 1.1. The first transaction in this table informs that five anonymous customers whose identifier is equivalent to one have purchased the items *bread and milk* simultaneously. Similar statements can be drawn for the remaining transactions in this Table 1.1. Without loss of generality, this row database can be represented as a binary columnar database, as shown in Table 1.2. It can be observed that the purchases of items ‘*bread and milk*’ by the first customer were indicated with the value ‘1’ while other unpurchased items were indicated with the value ‘0’. For brevity, we call our binary columnar databases a columnar database. A frequent pattern generated from these databases is:

$$\{\textit{bread, milk}\} \quad \{\textit{support} = 2\} \tag{1.1}$$

The above pattern informs that most of the customers who buy *bread* have also purchased *milk*. Therefore, both *bread and milk* are frequently co-occurring items. By identifying these patterns, supermarket managers can make informed decisions about product placement, marketing strategies, and inventory management.

Several algorithms [13] were described in the literature to find interesting patterns in transactional databases [14, 15], sequence databases [16], uncertain databases [17],

Table 1.3: Chetan’s supermarket temporal database

<i>Timestamp</i>	Items purchased
9AM	<i>bread, milk, jam</i>
1PM	<i>books, pen</i>
4PM	<i>bat, ball</i>
7PM	<i>jam, butter</i>
8PM	<i>bread, milk</i>

Table 1.4: Chetan’s columnar temporal database

<i>Timestamp</i>	items						
	<i>bread</i>	<i>milk</i>	<i>books</i>	<i>pen</i>	<i>bat</i>	<i>ball</i>	<i>butter</i>
9AM	1	1	0	0	0	0	0
1PM	0	0	1	1	0	0	0
4PM	0	0	0	0	1	1	0
7PM	1	0	0	0	0	0	1
8PM	1	1	0	0	0	0	0

graphs [18], and streams [19]. However, the widespread adoption of frequent pattern mining models has been hindered by its inability to discover temporal regularities that may exist in temporal databases. When confronted with this problem in real-world applications, researchers tried to discover frequent patterns that have exhibited perfect (or full) periodic behavior in a temporal database [20–22].

## 1.1 Discovering interesting patterns from temporal databases

Periodic-Frequent Pattern Mining (PFPM) [20, 23] is a variant of the frequent pattern mining model that focuses on identifying patterns that occur periodically in given temporal (or time series) data. The mining process involves analyzing temporal data to identify interesting patterns that occur periodically and frequently, named as *PFPs*. This can be achieved by defining a *minSup* and *maxPer* thresholds. The *maxPer* determines the maximum time interval within which a pattern must repeat in the database. A classical application of PFPM is Market Basket analytics (MB analytics). It involves identifying frequent patterns that occur periodically in a TDB.

**Example 2.** *Continuing with the previous Example 1, we have already mentioned that bread and milk were frequently purchased items. However, looking at Table 1.3, we can get more extra information, i.e., both bread and milk were purchased during the early morning and late evening hours. Therefore, we can say that these items were PFPs. By identifying this PFPs, retailers can make informed decisions about product placement, marketing strategies, and inventory management.*

In this thesis, Table 1.3 represents the Row Temporal database (RTDB) format, and the same database can be represented as Columnar Temporal databases (CTDBs) as shown in Table 1.4. We may generalize the content of the Table 1.3 to Table 1.5, i.e., each item is generalized with a specific symbol such as *bread* as item *a* and *milk* as item *b* and so on. The following section discusses the formal model of PFPM.



---

### 1.1.1 Periodic-Frequent Pattern Model

**Definition 1. (Discovering set of timestamps of  $X$ .)** In Temporal databases (or time series databases) (TDBs), we are not worrying about the transaction identifier (ID), and instead, we must consider the temporal occurrence information of the patterns as timestamps denoted as  $ts$ . All the transactions were reconsidered as follows: each tuple  $t_k = (ts, Y)$  represents the occurrence of pattern  $Y$  at timestamp  $ts$ , where  $ts \in \mathbb{R}+$  signifies the timestamp associated with the pattern  $Y$ . A TDB over  $I$  constitutes a collection of transactions, denoted as  $TDB = \{t_1, \dots, t_m\}$ , with  $m = |TDB|$  representing the total number of transactions within TDB. Let's consider the scenario where  $t_k = (ts, Y)$ , and  $X \subseteq Y$ . In such cases, we state that  $X$  occurs within  $t_k$  (or  $t_k$  contains  $X$ ), and we represent this relationship as  $ts_k^X$ . The entire set of ordered timestamps in which  $X$  occurs within the TDB is denoted as  $TS^X = \{ts_j^X, \dots, ts_k^X\}$ , where  $j \leq k$ .

**Example 3.** Consider a given set of items  $I = \{a, b, c, d, e, f\}$ . Let us suppose that we have a hypothetical row temporal database (RTDB) derived from  $I$ , which is illustrated in Table 1.5. To enhance our analysis, we can transform this RTDB into a columnar temporal database schema (CTDBs), as presented in Table 1.6. Within this database, the combination of items 'b' and 'c' forms a pattern, which we shall refer to as 'bc' for brevity. Given that this pattern consists of two items, it is classified as a 2-pattern. Notably, the 'bc' pattern appears in transactions associated with  $ts$  1, 3, 4, 6, 9, and 10. Consequently, we obtain a list of timestamps ( $ts$ ) that contain the 'bc' pattern, denoted as  $TS^{bc} = \{1, 3, 4, 6, 9, 10\}$ .

**Definition 2. (The support of  $X$  [12].)** The support of a given pattern  $X$  in the TDB represents the number of transactions within the database that contain  $X$ . It is denoted as  $sup(X)$ , where  $sup(X) = |TS^X|$ . Here,  $TS^X$  refers to the set of timestamps in which the pattern  $X$  occurs.

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

**Definition 3. (Frequent pattern  $X$  [12].)** A pattern  $X$  is classified as a frequent pattern if its support, denoted as  $sup(X) \geq$  the user-defined minimum support threshold ( $minSup$ ).

**Example 5.** Given a user-specified  $minSup$  of 5, the pattern 'bc' is deemed a frequent pattern since its support,  $sup(bc)$ , satisfies the condition  $sup(bc) \geq minSup$ .

**Definition 4. (Periodicity of  $X$  [20].)** Let's consider two consecutive timestamps in the set  $TS^X$ :  $ts_q^X$  and  $ts_r^X$ , where  $j \leq q < r \leq k$ . The time difference, also known as the inter-arrival time, between  $ts_r^X$  and  $ts_q^X$  is defined as the period of pattern  $X$ . We denote this period as  $p_a^X$ , which can be calculated as  $p_a^X = ts_r^X - ts_q^X$ . Now, let  $P^X = \{p_1^X, p_2^X, \dots, p_r^X\}$ , where  $r > 1$ , represent the set of all periods for pattern  $X$ . The periodicity of  $X$ , denoted as  $per(X)$ , is defined as the maximum value among the periods in  $P^X$ . In other words,  $per(X) = \max(p_1^X, p_2^X, \dots, p_r^X)$ .

**Example 6.** For the pattern 'bc', the periods are calculated as follows:  $p_1^{bc} = 1$  ( $= 1 - ts_{initial}$ ),  $p_2^{bc} = 2$  ( $= 3 - 1$ ),  $p_3^{bc} = 1$  ( $= 4 - 3$ ),  $p_4^{bc} = 2$  ( $= 6 - 4$ ),  $p_5^{bc} = 3$  ( $= 9 - 6$ ),  $p_6^{bc} = 1$  ( $= 10 - 9$ ), and  $p_7^{bc} = 0$  ( $= ts_{final} - 10$ ). Here,  $ts_{initial} = 0$  represents the timestamp of the initial transaction, and  $ts_{final} = |TDB| = 10$  represents the timestamp of the final transaction in the TDB. To determine the periodicity of 'bc', denoted as  $per(bc)$ , we find the maximum value among the periods:  $per(bc) = \max(1, 2, 1, 2, 3, 1, 0) = 3$ .

**Definition 5. (Periodic-frequent pattern  $X$  [20].)** A frequent pattern  $X$  is classified as a periodic-frequent pattern if its periodicity, denoted as  $per(X)$ , is less than or equal to the user-defined maximum periodicity threshold ( $maxPer$ ).

**Example 7.** If the user-defined  $maxPer$  is set to 3, then the frequent pattern ‘bc’ is considered a periodic-frequent pattern (Periodic-Frequent Pattern (PFP)) because its periodicity  $per(bc)$  satisfies the condition  $per(bc) \leq maxPer$ . Similarly, the patterns ‘bca’ and ‘ba’ are also classified as PFPs 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 PFPs discovered from Table 3.3 is presented in Figure 2.3(f), without the “sample” text being crossed out (i.e., strikethrough).

Table 1.5: Row database

$ts$	items
1	abcf
2	bd
3	abcd
4	abce
5	cef

Table 1.6: Columnar database

		items					
$ts$	$a$	$b$	$c$	$d$	$e$	$f$	
1	1	1	1	0	0	1	
2	0	1	0	1	0	0	
3	1	1	1	1	0	0	
4	1	1	1	0	1	0	
5	0	0	1	0	1	1	

Numerous algorithms were described in the literature using the above model to discover PFPs in Row Temporal databases (RTDBs). Some of them are as follows: Tanbeer et al. [20] proposed a periodic frequent pattern growth algorithm for discovering PFPs in Transactional databases (TDs). Uday et al. [24] introduced the PFP-growth++ algorithm, which efficiently mined PFPs using the PF-tree++ structure, leveraging the concept of local periodicity in large TDBs. Despite the existence of numerous algorithms in the literature [20, 23, 24] for addressing the issue of mining PFPs, all of them have employed the conventional RTDB format. This means multiple database scans are required, or the computation of PFPs cannot be performed asynchronously. Consequently, the algorithms for detecting PFPs become inefficient regarding both time and memory usage. The significance of mining data from a CTDBs format cannot be overlooked, given that large-scale real-world data is often stored in this format.

With this motivation, in this thesis we propose multiple efficient algorithms for identifying interesting patterns in CTDBs.

## 1.1.2 Contributions

In this thesis, we are discovering interesting patterns in spatiotemporal databases (STDs). The STD can be understood as a collection of Location Database (or geo-referenced database) (LD) considers spatial component and TDB considers the  $ts$  component. These two data types are integrated into a single database to allow for analysis and querying based on location and time. STDs find applications across various domains, including transportation planning, environmental monitoring, and emergency response. Examples of STDs include Oracle Spatial and Graph [25], PostGIS [26], and GeoServer [27]. Many researchers are dedicating their efforts to developing more effective methods for identifying periodic patterns in STDs. In this thesis, we have made an effort

---

to propose novel models and fast techniques to discover interesting patterns in large *STDs*. The initial two chapters of the thesis focused solely on the temporal component of the database while neglecting the spatial aspect. However, in the final chapter of the thesis, both the spatial and temporal components were considered for analysis and interpretation. In this thesis, even though we have used binary spatiotemporal databases, we call them spatiotemporal databases for brevity. The summary of the proposed approaches is as follows.

### **Discovering periodic-frequent patterns in columnar temporal databases**

The technique of identifying *PFPS*s can be utilized in RTDBs and CTDBs. However, current algorithms [20, 23, 28] in the literature only address RTDBs, making it challenging to apply the same approach to CTDBs. This study introduces a new algorithm, PF-ECLAT [29, 30], which is specifically designed to identify *PFPS*s in CTDBs. The PF-ECLAT is the first algorithm of its kind and can be used for CTDBs. The algorithm has been tested on synthetic and real-world databases, and the results demonstrate its efficiency in terms of memory and runtime, as well as its scalability. Additionally, two case studies are presented to illustrate the algorithm’s usefulness. In the first case study, the PF-ECLAT algorithm was used to identify regions in Japan (or sensor identities) where air pollution was regularly present. The second case study employed the algorithm to find regularly congested road segments in a transportation network.

### **Discovering partial periodic patterns in columnar temporal databases**

The distinction between full periodic patterns and *3Ps* is highlighted, with the former strictly adhering to cyclic behavior and disregarding uninteresting patterns based on a user-specific *maxPer* constraint. At the same time, the latter occurs regularly but only at specific times, such as on weekends or at a specific time of day. A new algorithm called 3P-ECLAT [31] has been proposed in this thesis to identify *3Ps* in CTDBs, which is a unique approach. The state-of-the-art algorithms cannot be directly applied to CTDBs. Extensive experiments on synthetic and real-world databases show that the 3P-ECLAT is efficient in terms of memory usage and runtime and is also scalable. Finally, we present a case study that showcases the usefulness of the 3P-ECLAT in analyzing air pollution data. This case study shows that the 3P-ECLAT can be used to identify patterns in air pollution data that occur regularly but only during specific time periods.

### **Discovering geo-referenced periodic frequent patterns in geo-referenced time series databases**

The fundamental *PFPS* model takes into account the time-related information of a pattern but ignores any spatial information associated with the pattern in a database. Thus, the model implicitly assumes that any geographical information, if present, would not affect the pattern’s significance in a database. However, in reality, spatial information plays a crucial role in determining a pattern’s interestingness, and users tend to find a pattern more appealing if the patterns are adjacent to each other. This thesis proposes a more flexible model, *GPFPs*, which can exist in a Geo-referenced Time Series Database (*GTSD*). A Geo-referenced Periodic-Frequent Pattern (*GPFP*) is a set of

frequently occurring neighboring items that appear regularly in the database. We introduce an efficient algorithm, GPPF-Miner [32], to discover all *GPFPs* in a *GTSD*. The algorithm uses a smart DFS approach to uncover the required patterns efficiently. The experimental results support the effectiveness of the proposed algorithm. Additionally, we present two case studies demonstrating how our approach was used to extract valuable information from databases related to air pollution and traffic congestion.

## 1.2 Related work

The focus of this section is on two related areas of research: the extraction of *PFPs* and *3Ps*. Next, we briefly discuss the efforts being made to discover spatial co-occurrence patterns.

### 1.2.1 Literature review on periodic-frequent pattern mining

Frequent pattern mining is a vital big data analytical technique with applications in various domains, including marketing, healthcare, and traffic congestion. It involves identifying all frequently occurring patterns in a given transactional database. Several algorithms, Apriori [12], FP-growth [14], and Eclat [15], have been developed for finding frequent patterns in a database. These algorithms differ in their approach to identifying frequent patterns and have varying levels of efficiency and scalability [13]. However, frequent pattern mining may not always be suitable for detecting consistent patterns over time. This is because the technique focuses on identifying patterns that occur frequently and does not consider temporal variations in pattern occurrence. To address this limitation, other techniques, such as periodic-frequent pattern mining, have been proposed [20].

Tanbeer et al. [20] developed a periodic frequent pattern growth algorithm to discover *PFPs* in TDs and introduced a tree-based data structure called PF-tree to store patterns. PF-tree has a tail-node to maintain a list of transaction identifiers for the pattern, which is moved to its parent when pruning. They used a *maxPer* and support-based measure to generate full-cyclic *PFPs* and claimed their mining process is efficient.

Amphawan et al. [21] developed the Mining Top-K *PFPs* (MTKPP) algorithm, which is a non-support metric-based method. The algorithm first identifies all *PFPs* and maintains them in a Top-K list structure. This list structure maintains the top-K *PFPs* and is sorted based on the frequency of the patterns. MTKPP uses a sliding window technique to identify *PFPs*. It slides a window over the database and counts the frequency of the items in the window. If the frequency of an item is above a certain threshold, it is considered a candidate *PFP*. The algorithm then checks if the pattern is periodic by comparing the frequency of the item in the current window with the frequency of the item in the previous window. The pattern is considered periodic if the difference is within a certain threshold. The algorithm uses a best-first strategy to identify the Top-K *PFPs*. It starts with the most frequent patterns and gradually prunes the patterns that are unlikely to be on the Top-K list. The algorithm stops when it has identified K patterns or when no more patterns can be pruned.

Uday et al. [33] developed an advanced model for identifying frequent and rare periodic patterns using a combination of multiple *minSup* and multiple *maxPer* con-

---

straints. The researchers employed two limitations, minimum item support and maximum item periodicity, to identify significant patterns. Each pattern must satisfy different criteria for  $minSup$  and  $maxPer$ , depending on the items included in it. Furthermore, the authors introduced a pattern-growth algorithm that uses a new and effective tree-based data structure, the Multi-Constraint Periodic-Frequent tree, to detect the complete set of frequent and rare items.

The PFP-growth++ algorithm proposed by Uday et al. [23, 24] can efficiently mine  $PFPs$  from large TDs. The PF-tree++ structure has also proved effective in storing the patterns and optimizing the mining process. Furthermore, the concept of local periodicity has significantly contributed to the algorithm, as it allows for early termination of the mining process when no new  $PFPs$  can be found. This feature saves time and resources by avoiding unnecessary computations. Finally, two pruning techniques are applied to further reduce the search space and improve the algorithm's efficiency. Overall, PFP-growth++ is an effective and scalable approach for discovering  $PFPs$  in TDs.

Amphawan et al. [34] discussed a novel approach for mining periodic-frequent patterns from TDs. The authors introduce a data structure called Interval Transaction-Ids List Tree (ITL-Tree), which efficiently stores and organizes interval transaction-ids lists, enabling the discovery of periodic-frequent patterns with approximate periodicity by identifying the intervals in which certain items frequently co-occur. The proposed method first builds the ITL-Tree from the input database, which involves grouping the transactions into intervals and constructing interval transaction-ids lists. The ITL-Tree is then used to identify approximate periods of transactions and generate candidate periodic-frequent patterns. Finally, the candidate patterns are pruned to obtain the final set of periodic-frequent patterns.

Uday et al. [35] proposed the minimum periodic ratio measure to discover  $PFPs$  from TDs, along with the concept of potential patterns consisting of a single item. They also introduced the Extended Periodic-frequent Pattern-tree (ExPF-tree) and Extended Periodic-frequent Pattern-growth (ExPF-growth) algorithms to mine the database, which includes ExPF-list and a prefix tree to preserve transactional identifiers of patterns. The ExPF-growth algorithm uses a DFS approach to construct the ExPF-tree and mine the database for potential patterns. The algorithm works by starting with a single item as a potential pattern and recursively expanding it by adding one item at a time, checking the minimum periodic ratio and support threshold at each level, and pruning any uninteresting patterns. The algorithm stops when no more potential patterns can be found or when all patterns have been mined.

Rashid et al. [36] describes an efficient approach for mining regularly frequent patterns, which is important in various data mining applications. The described approach utilizes the variance of interval time between pattern occurrences as a measure of temporal regularity. It uses a pattern-growth approach to find regularly frequent patterns based on user-given  $minSup$  and maximum variance thresholds.

Anirudh et al. [37] proposed a memory-efficient algorithm called Period Summary-growth (PS-growth) for mining  $PFPs$  in sparse databases. The PS-growth algorithm uses a vertical data format and a level-wise search approach similar to the Apriori algorithm [12]. However, it employs a novel period summary technique to reduce search space and memory usage. The period summary technique partitions the TD into multiple periodic summaries, each representing a subset of transactions that share the same periodicity. This reduces the number of candidate patterns generated at each level of the search tree, leading to faster mining and lower memory usage.

The EPF-growth algorithm proposed by Venkatesh et al. [38] is aimed at discovering rare *PFPS*s, which are patterns that occur frequently over a period of time but are rare in the overall database. The algorithm is designed to work with non-uniform TDs, which are databases where the length and content of transactions vary. The authors have also introduced a new constraint measure called periodic-all-confidence, which is used to extract interesting rare *PFPS*s. This measure takes into account both the periodicity and the rarity of a pattern and is shown to be effective in identifying patterns that are both frequent and rare.

The algorithms discussed above are designed to work with row-based databases, where the data is stored in rows, where each row represents a transaction, and the items are represented as columns. However, Columnar databases (CDBs) have become increasingly popular in recent years due to their superior performance in analytical workloads, especially for read-intensive queries. CDBs store data by column rather than by row, which makes them more efficient for certain types of queries. As a result, these algorithms cannot be directly applied to CDBs. In order to apply these algorithms to CDBs, some modifications may be required to take advantage of the unique properties of CDBs. On the other hand, we could make new algorithms that work well with CDBs.

In this thesis, we have made an effort to present a novel and efficient algorithm named, PF-ECLAT [30] for discovering *PFPS*s in CTDBs. The algorithm is an extension of the ECLAT algorithm, which is a well-known Frequent Pattern Mining (FPM) algorithm. The main idea behind PF-ECLAT is to exploit the temporal dimension of the data to identify patterns that occur frequently and periodically over time. To do this, the algorithm uses a list-based approach to efficiently generate candidate patterns and prune those that are not frequent. The algorithm also incorporates a period-based pruning strategy to further reduce the search space.

The PF-ECLAT algorithm was first presented as a preliminary version at a conference [29]. Later, we have thoroughly extended the experimental results section of the paper and presented these results in a journal [30]. Specifically, the extended experimental results included comparisons with several existing state-of-the-art algorithms on different databases. This suggests that we have further developed and evaluated the PF-ECLAT algorithm and provided additional evidence of its effectiveness compared to other approaches in the literature.

### 1.2.2 Literature review on partial periodic pattern mining

In the field of data mining and machine learning, time series analysis plays a crucial role in various domains such as finance, weather forecasting, and healthcare. Periodic pattern mining is a common problem in time series analysis and involves discovering patterns that exhibit cyclic repetitions. This problem has been widely studied in literature [39–42]. To do this, most studies use a two-step model. The two-step approach used to solve this problem involves partitioning the time series into distinct subsets or period segments of a fixed length or period and then discovering all Periodic (either full-periodic or partial periodic) patterns (PPs) that satisfy the user-defined *minSup*. The first step of partitioning the time series is critical to the success of the periodic pattern mining process. The length of the period or period segment should be chosen based on the characteristics of the time series being analyzed. If the period is too short, it may result in too many pattern occurrences, making it difficult to distinguish significant patterns from noise. On the other hand, if the period is too long, it may result in patterns

---

being missed entirely. Once the time series has been partitioned into distinct period segments, the second step is to discover all PPs that satisfy the user-defined *minSup*. The *minSup* is a threshold value that controls the minimum number of period segments in which a pattern must appear. Setting the *minSup* too low may result in many false positives, while setting it too high may result in missing important patterns.

Han et al. [39] proposed a method for discovering interesting patterns in time-related databases, with a focus on identifying segment-wise periodic patterns (SPPs). These are patterns that repeat in different segments of the time series data, rather than occurring throughout the entire database. For example, a segment-wise periodic pattern (SPP) could be a pattern that repeats every weekday in the morning but with a different pattern occurring in the afternoon or evening. To identify these patterns, the proposed method involves segmenting the time series data and then finding candidate patterns in each segment using a modified version of the Apriori algorithm [12]. The method then combines these candidate patterns across different segments to identify the complete set of SPPs. Overall, this approach offers a novel way to mine in time-related databases for interesting patterns, taking into account the segmented nature of the data.

Han et al. acknowledge that their previous work [39] on mining  $3Ps$  may provide imperfect  $3Ps$  i.e., the patterns exhibit some degree of repetition or periodicity, but with some variations or irregularities. In other words, these patterns exhibit repeating structures or motifs, but they may also contain variations or deviations from the expected pattern. These variations can be caused by a variety of factors, such as noise, randomness, or incomplete data. To address this issue, they presented an extended version [40] of their work that proposes efficient methods for mining  $3Ps$  in Time Series Databases (TSDs). The authors explored various properties related to partial periodicity and proposed several algorithms that can handle both single and multiple periods. One such property is the max-sub pattern hit set property, which enables the derivation of frequent pattern counts from a small subset of patterns mined from the time series. To further improve efficiency, the authors introduced a set of pruning techniques, such as the use of upper bounds on the frequency of candidate patterns, to reduce the search space for candidate patterns.

Yang et al. [42] proposes an algorithm for mining  $3Ps$  in sequential data. The algorithm extends the original InfoMiner [43] algorithm by allowing for gaps in the patterns. The authors introduce a gap penalty parameter that controls the maximum number of missing events allowed between two consecutive occurrences of a pattern. This allows the algorithm to detect  $3Ps$  with gaps between occurrences. The proposed method uses a generalized information-gain approach to identify patterns that occur periodically in the sequence data.

Raheed et al. [41] introduce a new algorithm called STNR for detecting periodic patterns in time series data. The STNR algorithm is indeed based on a suffix tree and designed to be resilient to noise, making it suitable for detecting periodic patterns in time series data even in the presence of significant noise. The algorithm can detect symbol, sequence, and segment periodicity while ignoring redundant and repeating periods. The authors' evaluation of the algorithm shows that it outperforms existing algorithms in terms of accuracy and efficiency and is able to maintain high accuracy under different levels of noise. Overall, STNR is a promising approach for detecting periodic patterns in time series data.

Berberidis et al. [44] propose a novel algorithm for discovering multiple and  $3Ps$  in TSDs. The authors note that existing algorithms for periodicity mining typically as-

sume that the data has a single global period and are not effective when the data has multiple or partial periodicities. To address this limitation, the authors propose a two-step algorithm based on the filter-refine paradigm that can identify multiple and  $3Ps$  in time series data. In this algorithm, during the filter step, the fast Fourier transform is applied to the time series data to compute its Fourier transform, which is then used to calculate the circular autocorrelation function. The resulting function provides a set of candidate period lengths for each letter in the alphabet of the time series. The set of candidate period lengths generated by this step is considered conservative because it is designed to include only the most significant periodicities in the time series while excluding noise or insignificant fluctuations. This conservative set of candidates is then further analyzed in the refine step to identify any  $3Ps$  that may exist within each candidate's period length.

The above studies have a limitation in that they treat time series as a symbolic sequence and do not incorporate the temporal information of events within the time series. This can be a significant limitation, as the actual temporal information may contain valuable insights that can be missed by treating the time series as a symbolic sequence.

Ozden et al. [45] introduced a method to discover cyclic association rules by enhancing a TD with a time attribute and fragmenting the database into non-overlapping subsets based on time. They counted the number of subsets in which a pattern occurred to discover cyclic association rules that appear in at least a certain number of subsets. This simplifies the mining algorithm, but a limitation is that it cannot discover patterns or association rules that span multiple windows.

While several models have been proposed in the literature to identify  $PFPs$  in a TDBs without requiring data segmentation, it is important to note that periodic patterns in a TDBs can be classified as full periodic patterns ( $PFPs$ ), or  $3Ps$ . Full periodic patterns are monitored strictly within the database, and uninteresting patterns are discarded based on user-defined constraints such as the  $maxPer$ . However, this approach may be too strict, as it may exclude potentially interesting patterns if even one of their periods does not meet the threshold.  $3Ps$ , on the other hand, occur only during specific times and can be found in real-world scenarios. For example, in a supermarket database, customers may purchase dairy items frequently but other standard items only at the end of the month. In the traffic congestion database, traffic congestion is higher during peak hours of the day. As such, it is often useful to mine  $3Ps$  in TDBs, as they can provide valuable insights that full periodic patterns may not capture. It is worth noting that all the existing models (see Section 1.2.1) for identifying  $PFPs$  are designed to detect patterns that exhibit complete cyclic repetitions in RTDBs or CTDBs.

When confronted with this problem in real-world applications, researchers have tried to find partially occurring  $PFPs$  using several new constraint measures. These constraints may specify the time intervals during which the pattern should occur or the minimum and maximum number of occurrences required for a pattern to be considered periodic-frequent. By applying these constraints, researchers have been able to identify partially occurring  $PFPs$ .

Uday et al. [46] paper proposes a novel approach for discovering Partial Periodic-Frequent Patterns ( $PPFPs$ ) in a TD.  $PPFPs$  are sub-sequences that occur frequently and periodically within longer sequences, such as a specific purchasing pattern that occurs every week or month. The authors introduce a novel measure called periodic-ratio to identify the  $PPFPs$  that satisfy a minimum periodic-ratio constraint. However, these  $PPFPs$  do not satisfy the down ward closure property, so the authors propose



---

a new algorithm called Generalized Periodic-Frequent pattern-Growth (GPF-Growth). GPF-Growth scans the entire database twice and stores candidate  $PPFPs$  in GPF-List format, which is then used to compress the database into a novel data structure called GPF-Tree. The authors then use a recursive mining process to generate the complete set of  $PPFPs$ .

S. Nakamura et al. [47] proposes a novel algorithm for discovering  $PPFPs$  in TDBs, which considers both Row database (RDB) and Columnar database (CDB) architecture. The authors introduce a novel generalized dictionary-based data structure to efficiently identify one-length candidate patterns. The proposed algorithm uses a measure called probable maximum periodic-ratio to prune uninteresting patterns in a DFS manner, which reduces the search space and increases efficiency. To evaluate the effectiveness of the proposed algorithm, the authors conducted experiments on both synthetic and real-world databases and compared it with the existing GPF-Growth algorithm. The results show that the proposed algorithm outperforms the existing algorithm in terms of both runtime and memory usage.

Rashid et al. [36] introduces a new algorithm for mining regularly occurring frequent patterns in TDs. The proposed algorithm uses a novel data structure and pruning techniques to efficiently discover patterns that occur frequently in a regular manner. The algorithm works by first identifying the sets of items that occur in a transaction with a frequency that satisfies a user-specified  $minSup$  threshold, and then generating regularly frequent patterns by combining these sets of items using a novel maximum variance constraint. The algorithm uses a prefix tree-based data structure named RF-tree. The use of the maximum variance constraint and the RF-tree data structure are key contributions of the algorithm, as they help to reduce the computational complexity of the pattern mining process and improve the efficiency of the algorithm.

The extended models that have been proposed for partial PFP often have numerous input parameters and are impractical to use on large databases. This is because the patterns generated by these models do not satisfy the downward closure property, making them less efficient for analyzing and interpreting large amounts of data.

Uday et al. [48] propose a new model for discovering  $3Ps$  in TDBs. The authors note that traditional periodic pattern mining techniques may not be sufficient for discovering  $3Ps$  and thus propose a new interesting measure named period-support, which allows the partial periodic behavior of the pattern. All of the uninteresting patterns were discarded using a novel minimum period-support measure. The authors have also proposed a novel pattern growth algorithm named 3P-growth to discover the complete set of  $3Ps$  using a 3P-tree data structure. Unfortunately, this algorithm can find  $3Ps$  only in RTDBs.

In this thesis, we have made an effort to present a novel and efficient algorithm named 3P-ECLAT [31] for discovering  $3Ps$  in CTDBs. One of the significant advantages of 3P-ECLAT is its versatility, as it can also be applied to horizontal databases, making it more flexible than existing algorithms. One of the key contributions of this paper is the use of columnar storage to improve the efficiency of the pattern discovery process. Columnar storage allows for efficient scanning and filtering of the database, which is critical for discovering  $3Ps$  in large databases. We have also proposed a technique for compressing the database to further reduce the memory requirements of the algorithm.

### 1.2.3 Literature review on spatial co-occurrence pattern mining

FPM, which was introduced in [49], has been widely used for identifying frequent patterns in TDs. However, the extension of FPM to *STDs* presents several challenges, such as the need to consider both spatial and temporal dimensions in the analysis. To address these challenges, various methods have been proposed for identifying Spatiotemporal (ST) co-occurrence patterns or association rules in *STDs*.

Ding et al. [50] present a comprehensive framework for discovering interesting patterns and associations in spatial databases. By using a combination of clustering and association rule mining algorithms, the framework can identify regions of interest and uncover relationships between spatial objects that might not be apparent through other methods. The authors have identified regions of interest using a clustering algorithm, Supervised Clustering using Multi Resolution Grids (SCMRG). The use of the SCMRG algorithm is particularly noteworthy, as it allows for the identification of regions that are both geographically close and share similar attributes. This can be especially useful for understanding complex spatial patterns that might be difficult to detect using traditional clustering techniques. Each region was used by the regional association rule mining algorithm to discover interesting patterns. The regional association rule mining algorithm uses a modified version of the Apriori [12] algorithm, which takes into account the spatial relationships between the items in the database. The algorithm discovers rules that are both spatially and statistically significant, taking into account both the spatial proximity of the items and their frequency of occurrence.

Eick et al. [51] propose a new framework for mining regional co-location patterns in spatial databases with continuous variables without the need for discretization. The proposed approach uses a clustering algorithm and a correlation-based fitness function to identify regions with high densities of data points that share similar values for multiple variables. The fitness function allows researchers to specify their notion of interestingness, enabling them to rank regions according to their research priorities. The approach was tested in a case study involving chemical concentrations in Texas water wells centered on co-location patterns involving arsenic, and proved useful in identifying and suggesting promising hypotheses for future research. The proposed algorithm, named CLEVER, is able to identify known and unknown regional co-location sets. Overall, the proposed method provides a powerful tool for identifying co-location patterns in spatial databases with continuous variables and has a wide range of potential applications in fields such as ecology, meteorology, and urban planning.

Mohan et al. [52] proposed an approach to identifying regional co-location patterns (RCPs) in geographical data. The authors propose the use of a neighborhood graph to represent the spatial relationships between different types of features in the data, such as businesses and residential areas. They then use this graph to identify regions of high co-location, where certain types of features are clustered together more than would be expected by chance. The authors introduce two novel interest measures to quantify the regional prevalence of RCPs. The effectiveness of the proposed approach is evaluated on real crime databases, which demonstrates that it is capable of identifying significant RCPs and providing insights into the underlying causes of spatial phenomena. Overall, the paper presents a novel approach for identifying regional co-location patterns in spatial databases and introduces two novel interest measures and a novel pruning algorithm.

Hung et al. [53] discussed a method to analyze ST data to identify co-occurring

---

events. The authors focus on the challenge of analyzing data that is unevenly distributed in time and space, which is a common problem in many real-world applications. The authors used the DBSCAN [54] clustering algorithm and repeated the process to define ST neighborhoods and quantify co-occurrence patterns between events. They also propose a new interest measure of co-occurrence patterns for prediction purposes. The authors evaluated the proposed method on a real-world database of traffic congestion during disasters in Kansai, Japan. They show that their method is able to identify meaningful co-occurrence patterns between traffic congestion and typhoons, which can be useful for improving traffic safety. Overall, the paper provides a novel method for analyzing ST data that is unevenly distributed, which has the potential to be applied to a wide range of applications beyond traffic safety.

Uday et al. [55] introduced a method for discovering frequent spatial patterns in large spatiotemporal databases. The authors first define the problem of frequent spatial pattern mining and then present their approach, which is based on the idea of a “pattern growth” method. This method involves identifying frequent sub-patterns and then growing them into larger patterns by adding new spatial locations. Finally, the authors evaluate their approach in several real-world, and the results demonstrate the effectiveness of their approach in discovering frequent spatial patterns in large databases with high efficiency and scalability.

Unfortunately, all the above-mentioned approaches have only considered the *support* and *spatial information* of patterns while disregarding their *temporal information*. This is a significant limitation of the existing approaches.

However, the research presented in this thesis aims to address the limitations of existing approaches to ST co-occurrence pattern mining by developing a new and efficient algorithm, GPFM-Miner [32], to discover all *GPFMs* in a *GTSD* that considers not only the *support* and *spatial information* of a pattern but also its *temporal information*. The algorithm uses a smart DFS approach to uncover the required patterns efficiently. The experimental results support the effectiveness of the proposed algorithm. Additionally, we present two case studies demonstrating how our approach was used to extract valuable information from databases related to air pollution and traffic congestion.

## 1.3 Organization

In Chapter 2, we discuss the limitations of existing algorithms for discovering *PFPs* in CTDBs and propose a new algorithm that leverages the temporal dimension of the data to identify such patterns efficiently. The proposed algorithm utilizes a list-based approach to generate candidate patterns and incorporates period-based pruning to reduce the search space further. The chapter presents experimental results that demonstrate the proposed algorithm’s effectiveness compared to the state-of-the-art algorithms. Overall, Chapter 2 provides an overview of the proposed methodology and its advantages in discovering *PFPs* in CTDBs.

In Chapter 3, we discuss the limitations of existing algorithms for discovering *3Ps* in CTDBs and propose a new algorithm that addresses these limitations. The chapter presents the partial periodic pattern discovery mechanism, allowing the algorithm to identify patterns that occur only at certain period intervals. Experimental results demonstrate that the proposed algorithm outperforms the existing algorithm in terms of efficiency and scalability. Overall, Chapter 3 presents a novel algorithm to discover

3Ps in CTDBs and addresses a significant problem in data mining.

In Chapter 4, we present a model for *GFPF* and propose a smart DFS algorithm to uncover interesting patterns efficiently. The proposed algorithm can handle variations in the data's *support* and *spatial* aspects and detect patterns that occur at different *time scales*. The results of applying the algorithm to real-world traffic congestion and air pollution databases demonstrate its ability to uncover meaningful patterns that offer insights into the underlying dynamics of the data. Overall, the chapter contributes to the field of spatiotemporal data mining by providing a new approach for discovering *GFPFs* in Geo-referenced Time Series Databases (*GTSDs*).

The conclusion and future research directions of our thesis are presented in Chapter 5.

## Chapter 2

# Efficient Discovery of Periodic-Frequent Patterns in Columnar Temporal Databases

*GTSD* is one of the key forms of a *STD* [56]. Our research is focused on discovering interesting patterns in *STDs*, but during the early stages, we did not consider the spatial characteristics of these databases, i.e., we did not consider LD. Specifically, we only focused on the temporal characteristics of the patterns and aimed to discover interesting patterns based on temporal (or time series) databases only. In this chapter, we have discovered *PFPs* in CTDBs

The discovery of *PFPs* can be utilized in both RTDBs and CTDBs. There have been numerous algorithms proposed in the literature, including PFP-growth [20], PFP-growth++ [23], and PS-growth [37], which aim to identify *PFPs* in RTDBs. However, there is no algorithm available to identify *PFPs* in a CTDBs. One approach to finding *PFPs* in a Columnar Temporal database (CTDB) involves converting it into a RTDB using existing methods. However, such a transformation process can be computationally expensive and should be avoided. Therefore, the objective of our research is to identify *PFPs* in a CTDB more efficiently.

Finding *PFPs* in CTDBs is non-trivial and challenging due to the following reasons:

1. Zaki et al. [15] made a significant contribution by highlighting the significance of identifying frequent patterns in Row databases (RDBs). They introduced the Equivalence Class Transformation (ECLAT) algorithm, which utilizes a DFS approach to discover frequent patterns in a RDB. However, it is important to note that the ECLAT algorithm is not directly applicable for finding *PFPs* in CTDBs. This limitation arises from the fact that the ECLAT algorithm disregards the temporal occurrence information associated with each item in the database. As a result, it cannot capture the temporal characteristics such as periodicities required to identify *PFPs* effectively.
2. The space of items within a database forms an itemset lattice, which represents the entire search space for discovering interesting patterns. The size of this lattice is determined by the number of items present in the database and is given by  $2^n - 1$ , where  $n$  denotes the total number of items. Consequently, the itemset lattice can quickly become enormous, resulting in a challenging task for pattern

mining. One of the primary challenges in pattern mining is reducing this vast search space to a manageable size. It involves developing efficient techniques and algorithms that can effectively navigate and explore the itemset lattice. By employing various pruning strategies, optimization techniques, and heuristics, researchers aim to reduce the search space and focus on discovering patterns that are deemed interesting or significant based on certain criteria, such as *support*, *periodicity*, or other measures.

**Example 8.** *The itemset lattice, as depicted in Figure 2.1(a), represents the search space for patterns consisting of the three items  $a$ ,  $b$ , and  $c$ . In this case, the size of the itemset lattice is  $2^3 - 1 = 7$ . This means that there are a total of 7 possible combinations of itemsets that need to be explored by the mining algorithm to discover the desired PFPs in a CTDB. Effectively searching this extensive lattice poses a significant challenge due to the large number of possible itemset combinations. Mining algorithms need to be designed with efficient search strategies and pruning techniques to navigate and explore the itemset lattice in an optimized manner. This involves considering various factors, such as pattern length, temporal constraints, and periodicities, to identify and prioritize the most relevant and interesting PFPs within the given CTDB. Addressing the complexity of searching the itemset lattice is crucial for efficient and effective mining of PFPs in CTDBs, particularly when dealing with larger databases with a higher number of items.*

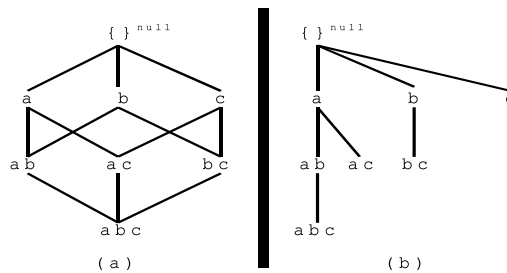


Figure 2.1: Sample representation of itemset space. (a) representing the items  $a$ ,  $b$ , and  $c$  using a lattice structure (b) DFS-based evaluation of the itemsets

In this chapter, we first define the problem of discovering *PFPs* in CTDB. Next, we discuss about the PF-ECLAT [30] algorithm<sup>1</sup>, which is designed to identify the complete set of *PFPs* in CTDB. After that, we check the performance of the PF-ECLAT algorithm against several state-of-the-art algorithms by considering both synthetic and real-world databases. Finally, we conclude the chapter by summarizing our findings.

**Definition 6. (Problem definition.)** *The objective is to identify the comprehensive collection of PFPs from a given TDB, while satisfying the user-specified  $minSup$  and  $maxPer$  constraints.*

<sup>1</sup>Initial version of this algorithm is presented at [29]

---

## 2.1 Proposed Algorithm

Although our proposed algorithm can generate interesting patterns using either RTDB or CTDB architecture, existing state-of-the-art algorithms are only capable of generating such patterns in RTDB architecture. As a result, we used RTDB architecture for evaluation purposes only, even though this required additional effort to transform the data from row to columnar architecture.

In this section, we will outline the procedure for identifying one-length *PFPs* (or 1-patterns) and transforming a RTDB into a CTDB. Subsequently, we will introduce the PF-ECLAT algorithm, which facilitates the discovery of a comprehensive set of *PFPs* in a CTDB. The PF-ECLAT algorithm utilizes a DFS approach and leverages the *downward closure property* (see Property 1) of *PFPs*. This property helps in effectively reducing the immense search space during mining process. By utilizing the DFS approach and leveraging the downward closure property, the PF-ECLAT algorithm enables efficient exploration of the search space and identifies a complete set of *PFPs* within the CTDB.

**Property 1.** (*The downward closure property* [20].) *if a pattern  $Y$  is classified as a PFP, then for any non-empty subset  $X$  of  $Y$ , where  $X$  is not equal to the empty set,  $X$  is also considered a PFP.*

### 2.1.1 PF-ECLAT algorithm

#### Finding one length periodic-frequent patterns

By employing Algorithm 1 in conjunction with the specified RTDB shown in Table 1.5, we can effectively identify and extract the set of 1-patterns. The algorithm leverages the PFP-list dictionary to maintain relevant occurrence information and support counts for each item. Subsequently, it filters out items that fail to meet the *minSup* threshold. Additionally, the periodicity of each remaining item is assessed by calculating the periods between consecutive occurrences within transactions. Any item exceeding the *maxPer* constraint is excluded from the final list of 1-patterns. Throughout the evaluation of the algorithm, we set the *minSup* to 5 and the *maxPer* to 3.

The algorithm begins by scanning the first transaction, “1:abcf”, with the current timestamp  $ts_{cur} = 1$ . This results in the insertion of items  $a$ ,  $b$ ,  $c$ , and  $f$  into the PFP-list. The timestamps of these items are set to 1 ( $= ts_{cur}$ ), and their corresponding *Per* and  $TS_l$  values are also initialized to 1. This process is executed in lines 5 and 6 of Algorithm 1. The generated PFP-list after scanning the first transaction is depicted in Figure 2.2(a). The second transaction, “2:bd”, is then scanned with  $ts_{cur} = 2$ . This leads to the insertion of the new item  $d$  into the PFP-list, with its timestamp set to 2. Simultaneously, the *Per* and  $TS_l$  values of item  $d$  are updated to 2. Additionally, the timestamp 2 is added to the TS-list of the existing item  $b$ , and its *Per* and  $TS_l$  values are adjusted accordingly (lines 7 and 8 in Algorithm 1). The resulting PFP-list after scanning the second transaction is illustrated in Figure 2.2(b). The third transaction, “3:bcd”, is processed, leading to the update of the TS-lists, *Per*, and  $TS_l$  values of items  $b$ ,  $c$ , and  $d$  in the PFP-list. The updated PFP-list after scanning the third transaction is shown in Figure 2.2(c). The fourth transaction, “4:abce”, is scanned with  $ts_{cur} = 4$ . As a result, the new item  $e$  is inserted into the PFP-list, with its timestamp set to 4. Simultaneously, the *Per* and  $TS_l$  values of item  $e$  are initialized to 4. Ad-

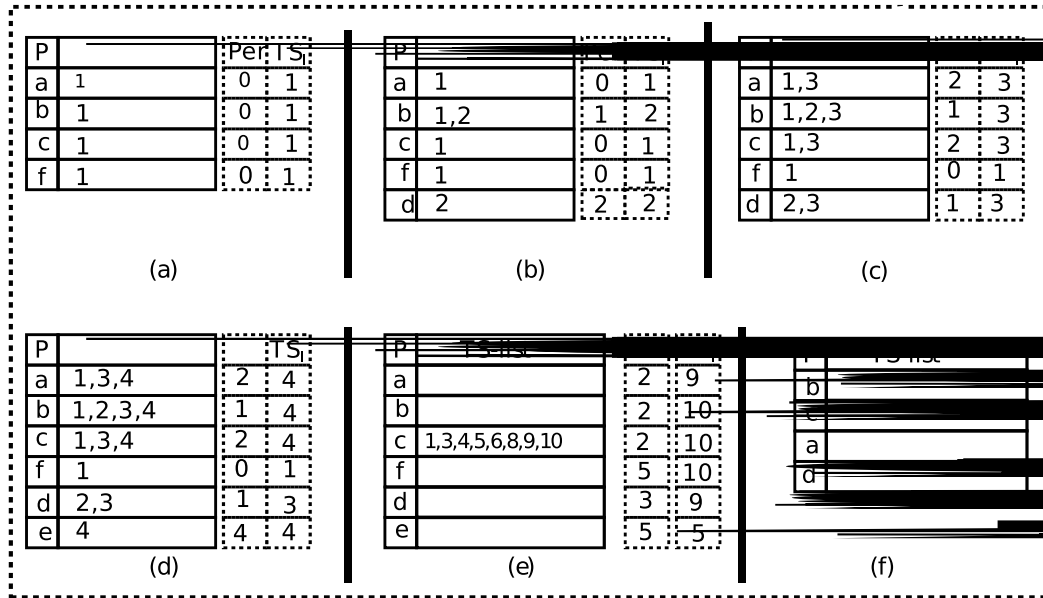


Figure 2.2: Initially, the PFP-list is empty. After scanning the first transaction, we insert each item into the PFP-list, as seen in (a). After scanning the second transaction, we add a new item to the PFP-list and update the values of the existing items, as shown in (b). After scanning the third transaction, we update the values of the existing items, as shown in (c). After scanning the fourth transaction, we add a new item to the PFP-list and update the values of the existing items, as shown in (d). After scanning the whole database, we obtain the final 3P-list as shown in (e). Finally, we sort the one-length PFPs in descending order of their *support*, as shown in (h).



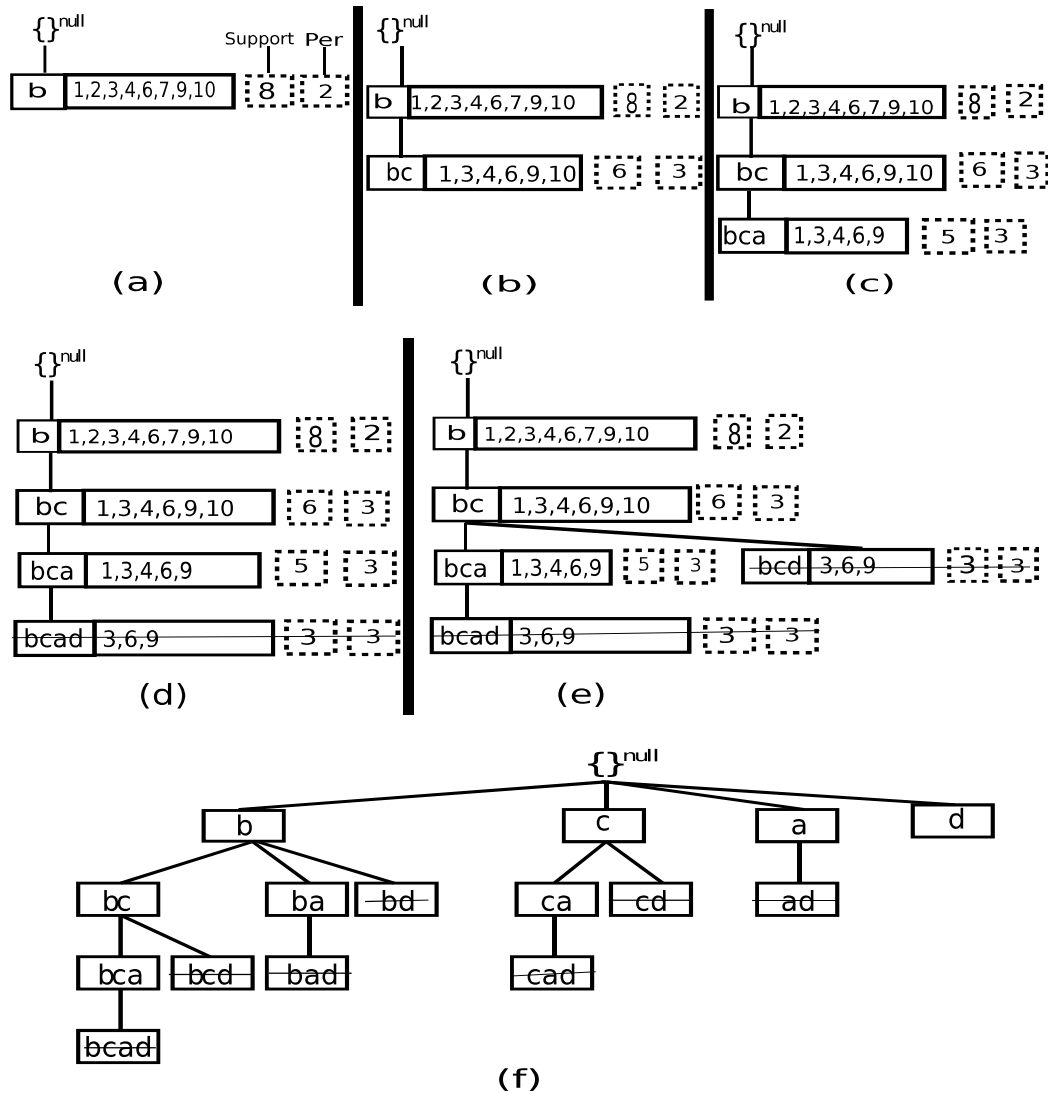


Figure 2.3: Mining *PFPs* using DFS.

ditionally, the TS-lists,  $Per$ , and  $TS_i$  values of the existing items  $a$ ,  $b$ ,  $c$ , and  $e$  in the PFP-list are updated accordingly. The PFP-list after scanning the fourth transaction is depicted in Figure 2.2(d). This process is repeated for the remaining transactions in the database. The final PFP-list generated after scanning the entire database is presented in Figure 2.2(e). Utilizing Property 1, the patterns  $e$  and  $f$  are pruned from the PFP-list as their support values are less than the user-specified  $minSup$ . This pruning process is carried out in lines 10 to 15 of Algorithm 1. The remaining patterns in the PFP-list are considered *PFPs* and are sorted in descending order based on their *support* values. The final PFP-list, obtained after sorting the *PFPs*, is displayed in Figure 2.2(f).

### Finding periodic-frequent patterns using PFP-list.

Algorithm 2 outlines the step-by-step procedure for discovering all *PFPs* in a given database. We will now elaborate on the workings of this algorithm using the above generated PFP-list.

The algorithm commences by selecting the first pattern in the PFP-list, which is item ‘ $b$ ’ (line 2 in Algorithm 2). The *support* and *periodicity* values of ‘ $b$ ’ are recorded and displayed in Figure 2.3(a). Since ‘ $b$ ’ satisfies the criteria to be classified as a *PFP*, we proceed to its child node ‘ $bc$ ’. To generate the TS-list for ‘ $bc$ ’, we perform an intersection operation between the TS-lists of ‘ $b$ ’ and ‘ $c$ ’, denoted as  $TS^{bc} = TS^b \cap TS^c$  (lines 3 and 4 in Algorithm 2). The *support* and *periodicity* values of ‘ $bc$ ’ are recorded and displayed in Figure 2.3(b). Next, we verify whether ‘ $bc$ ’ qualifies as a *PFP* or is deemed an uninteresting pattern (line 5 in Algorithm 2). As ‘ $bc$ ’ satisfies the requirements to be classified as a *PFP*, we proceed to its child node ‘ $bca$ ’. Similarly, we generate the TS-list for ‘ $bca$ ’ by intersecting the TS-lists of ‘ $bc$ ’ and ‘ $a$ ’, denoted as  $TS^{bca} = TS^{bc} \cap TS^a$ . The *support* and *periodicity* values of ‘ $bca$ ’ are recorded and displayed in Figure 2.3(c), confirming its classification as a *PFP*. Subsequently, we move to the child node ‘ $bcad$ ’ and generate its TS-list by performing an intersection between the TS-lists of ‘ $bca$ ’ and ‘ $d$ ’, denoted as  $TS^{bcad} = TS^{bca} \cap TS^d$ . However, as the *support* of ‘ $bcad$ ’ is less than the user-specified *minSup*, we prune this pattern from the list of *PFPs* (Figure 2.3(d)). Since ‘ $bcad$ ’ is the leaf node in the set-enumeration tree (or there exists no superset of ‘ $bcad$ ’), we construct ‘ $bcd$ ’ by intersecting the TS-lists of ‘ $bc$ ’ and ‘ $d$ ’, denoted as  $TS^{bcd} = TS^{bc} \cap TS^d$ . However, as the *support* of ‘ $bcd$ ’ is less than the user-specified *minSup*, we prune this pattern from the list of *PFPs* (Figure 2.3(e)). This process is repeated for the remaining nodes in the set-enumeration tree to identify all *PFPs*. The final list of *PFPs* generated from Table 1.5 is displayed in Figure 2.3(f). Utilizing the *downward closure property* for the detection of *PFPs* greatly optimizes the process due to its impactful role in diminishing the explored space and computational burdens. The algorithm’s reliability is validated by its adherence to Properties 2, 3, and 4, and this is further substantiated through the practical illustration provided by Lemma 1.

**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 *ts* at which patterns  $X$  and  $Y$  have respectively occurred in the database, then the set of *ts* at which  $Z$  has appeared in the database, i.e.,  $TS^Z = TS^X \cap TS^Y$ .

**Property 3.** In the case where the  $minSup > |TS^X|$ , it can be concluded that  $X$  does not meet the criteria to be classified as a *PFP*. Additionally, for any itemset  $Z$  that is a superset of  $X$ , it is similarly impossible for  $Z$  to qualify as a *PFP*.

*Proof.* The relationship between patterns  $X$  and  $Z$  can be established by observing that if  $X$  is a subset of  $Z$ , then the set of *ts*  $TS^X$  is a superset of  $TS^Z$ . Consequently, considering the definition of *minSup*, it follows that *minSup* must be greater than the cardinality of  $TS^X$ , which in turn is greater than or equal to the cardinality of  $TS^Z$ . Based on this reasoning, it can be concluded that  $Z$  cannot be classified as a *PFP*. Thus, the property is proven.  $\square$

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

*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 *PFP*. Hence proved.  $\square$

**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 *PFPs*, then  $Z$  cannot be a *PFP*. In other words, we do not need to check whether  $Z$  is a *PFP* if any one of its supersets is not a *PFP*.

---

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

---

- 1: We define the dictionary  $PFP-list = (X, TS-list(X))$  as a data structure that stores the temporal occurrence information of a pattern in a TDB. To facilitate the storage and retrieval of this information, we utilize the temporary lists  $TS_l$  and  $Per$ . The list  $TS_l$  maintains the timestamp of the last occurrence of each item in the database, while the list  $Per$  records the *periodicity* of the items. Additionally, we employ another temporary list, denoted as *support*, to keep track of the support values associated with patterns during the mining process
  - 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 exist in PFP-list **then**
  - 6:       The item  $i$  and its corresponding  $ts$  are inserted into the PFP-list. Subsequently, the  $ts$   $ts_{cur}$  is recorded as the last occurrence of item  $i$  by setting  $TS_l[i] = ts_{cur}$ . Additionally, the periodicity  $Per[i]$  of item  $i$  is calculated as  $Per[i] = (ts_{cur} - ts_{initial})$ ;
  - 7:     **else**
  - 8:       The item  $i$  and its corresponding  $ts$  are inserted into the PFP-list. The last occurrence of item  $i$  is recorded as  $ts_{cur}$  by setting  $TS_l[i] = ts_{cur}$ . The *periodicity* of item  $i$  is calculated as  $Per[i] = (ts_{cur} - ts_{initial})$ , where  $ts_{initial}$  represents the initial  $ts$  of the database.
  - 9:   **for** each item  $i$  in PFP-list **do**
  - 10:      $support[i] = length(TS-list(i))$
  - 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: The remaining items in the PFP-list are sorted either in ascending or descending order based on their *support* values. Afterwards, the PF-ECLAT algorithm is invoked, passing the sorted PFP-List as input.
- 

**Algorithm 2** PF-ECLAT(PFP-List)

---

- 1: **for** each item  $i$  in PFP-List **do**
  - 2:   Set  $pi = \emptyset$  and  $X = i$ ;
  - 3:   **for** each item  $j$  that comes after  $i$  in the PFP-list **do**
  - 4:     Set  $Y = X \cup j$  and  $TS^Y = TS^X \cap TS^j$ ;
  - 5:     **if**  $sup(TS^Y) \geq minSup$  and  $per(TS^Y) \leq maxPer$  **then**
  - 6:       The itemset  $Y$  is added to the pattern  $pi$ , and as a result,  $Y$  is classified as a *PFP*.
  - 7:   PF-ECLAT( $pi$ )
-

*Proof.* The correctness of the above statement is straightforward to prove from Properties 2, 3, and 4. Hence proved.  $\square$

### 2.1.2 Time complexity analysis

Suppose we are examining a database that stores temporal information. This database contains a total of  $a$  transactions, each of which corresponds to a specific point in time. Across all of these transactions, there are  $c$  unique items that have been recorded. Furthermore, the average transaction length is equal to  $b$ . In this database, all items are deemed to be of interest and are therefore included in the analysis. Understanding the characteristics of the database, including the number of transactions, unique items count, and the length of transactions, is crucial for performing the complexity analysis.

The PF-ECLAT algorithm provides significant contributions to the field of PFP, as demonstrated by its efficient computation and identification of PFPs. The Algorithm 1 begins by scanning the entire database and calculating the *support*, and *periodicity* of each item. A list of items that meet the *minSup* and *maxPer* constraints is then created and sorted in descending order based on their *support*. The complexity of this initial Algorithm is  $O(ab)$ , where  $a$  is the number of transactions and  $b$  is the average transaction length.

Once the one-length PFPs have been identified, then we proceed to generate combinations of items to form larger periodic-frequent itemsets. This is accomplished using procedures outlined in Algorithm 2. Algorithm 2 follows two steps procedure. The first step involves accessing two items and comparing their  $(d - 1)$  itemset timestamp lists to generate a  $d$ -itemset timestamp list with a complexity of  $O(c^2)$ . The second step involves calculating the *periodicity* and *support* of each itemset and discarding the uninteresting patterns based on the user-specified Overall, the complexity of this Algorithm 2 is  $O(c^2)$ .

Finally, the entire complexity of finding all the PFPs using PF-ECLAT is  $O(c^2)$ , which makes the PF-ECLAT a highly efficient method for PFP in columnar temporal databases.

## 2.2 Experimental Results

In this section, we commence by conducting a comparative analysis of the PF-ECLAT algorithm against state-of-the-art algorithms such as PFP-growth [20], PFP-growth++ [23], and PS-growth [37]. Our objective is to demonstrate that our algorithm outperforms these existing approaches in terms of memory utilization, runtime efficiency, and scalability. Furthermore, we showcase the practical significance and applicability of our algorithm through two case studies: TC analytics and AP analytics. These case studies highlight the effectiveness of PF-ECLAT in real-world scenarios, providing insights into its utility and potential impact in diverse domains.

### 2.2.1 Experimental setup

The Python implementations of the algorithms, namely PFP-growth, PFP-growth++, PS-growth, and PF-ECLAT, were developed specifically for Python version 3.7. The experiments were conducted on a machine equipped with an Intel(R) Core i5-3230M

Table 2.1: 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	6,289	8,341	10,122	766
4	Congestion	Real	Sparse	1	58	337	8,928
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

CPU operating at a base frequency of 2.6GHz and a maximum Turbo Boost frequency of 3.2GHz. The machine had 4GB of RAM and was running the Ubuntu 18.04 operating system. To evaluate the performance of the algorithms, experiments were conducted on both real-world and synthetic databases. The real-world databases used in the experiments were BMS-WebView-1, Pollution, Drought, Congestion, BMS-WebView-2, and Kosarak. Additionally, the synthetic database T10I4D100K was also employed for evaluation purposes.

The T10I4D100K database is a synthetic database generated following the procedure outlined in [12]. It has been widely utilized as a benchmark for evaluating various pattern-mining algorithms. The BMS-WebView-1 and BMS-WebView-2 databases are real-world sparse databases that contain clickstream data from an anonymous eCommerce company. These databases were utilized in the KDD Cup 2000 competition. Notably, both databases feature short transactions. The Kosarak database is a massive real-world sparse database that has been widely used for evaluating the scalability of pattern-mining algorithms. Its inclusion in this chapter enables an assessment of the scalability of the PFP-growth, PFP-growth++, PS-growth, and PF-ECLAT algorithms. To ensure the reproducibility of our experiments, comprehensive evaluation results, including the databases and algorithms used, have been made available through GitHub [57]. These databases were obtained from the Sequence Pattern Mining Framework (SPMF) [58] repository. Lastly, the Drought database [59] is a dense real-world database characterized by a high-dimensional structure. It has been selected to provide a unique and challenging dataset for evaluation purposes.

The monitoring of traffic congestion in smart cities poses a significant challenge in the field of Intelligent Transportation Systems. In addressing this challenge, the Japan Road Traffic Information Center (JARTIC) [60] project has established a comprehensive sensor network spanning across Japan. This nationwide network aims to monitor and analyze traffic congestion levels in real-time. The sensor network deployed by JARTIC operates by collecting data from individual sensors placed at various locations. Each sensor provides information on the congestion level of a specific road segment at regular 5-minute intervals. The collected data from this network forms a substantial volume of big data, representing a quantitative (non-binary) columnar temporal database. To facilitate the analysis and mining of this extensive dataset, we have converted it into a binary form, resulting in a columnar temporal database (CTDB). This conversion was achieved by setting a threshold value of 200 meters. Specifically, congestion lengths shorter than 200 meters, which are often attributed to waiting times at traffic signals, were excluded from the dataset. For our expertise and analysis, we focus on the binary

columnar traffic database named Congestion, which was derived from the JARTIC network’s data collected in Kobe, the fifth-largest city in Japan. This database serves as a valuable resource for investigating traffic congestion patterns and developing effective strategies for congestion management in urban areas.

Air pollution is a significant contributor to cardio-respiratory health issues reported in Japan, resulting in an alarming annual death toll of approximately 60,000 individuals [61]. In response to this pressing concern, the Ministry of Environment in Japan has implemented a sensor network system known as SORAMAME [62]. This system is designed to monitor air pollution levels across the entire country. Within the SORAMAME network, each sensor is responsible for collecting data on pollution levels of various air pollutants at hourly intervals. For this particular experiment, we have utilized a three-month dataset focusing on the PM2.5 pollutants. This dataset comprises readings collected from sensors positioned throughout Japan. The Pollution database used in this experiment is characterized by its dense nature and high dimensionality. It contains numerous long transactions, capturing a comprehensive representation of air pollution levels across various regions and time periods in Japan. The database serves as a valuable resource for conducting analyses and investigations related to air pollution patterns, contributing to the development of effective strategies and policies for mitigating the impact of air pollution on public health.

Table 2.1 presents the statistics of all the aforementioned databases, including their key characteristics and relevant details. To ensure the reproducibility of our experiments, we have made the complete evaluation results, along with the databases and algorithms used, available through GitHub [57]. This enables independent verification and replication of our experimental findings. However, please note that the Congestion and Drought databases are not included in the GitHub repository due to confidentiality reasons. Nonetheless, all other databases, algorithms, and evaluation results are accessible for comprehensive analysis and validation purposes.

### 2.2.2 Evaluation of PFP-growth, PFP-growth++, PS-growth, and PF-ECLAT algorithms by varying $maxPer$ constraint

In this experiment, the performance of the PFP-growth, PFP-growth++, PS-growth, and PF-ECLAT algorithms was evaluated by varying the  $maxPer$  constraint while keeping the  $minSup$  value fixed for each of the databases. The fixed  $minSup$  values ensured consistency in the  $support$  threshold across all databases, while the varying  $maxPer$  constraints provided insights into the algorithms’ behavior and performance in relation to different levels of  $periodicity$  present in the databases. The specific  $minSup$  values chosen for the BMS-WebView-1, Pollution, Drought, Congestion, BMS-WebView-2, and T10I4D100K databases were 0.07%, 51%, 57%, 30%, 0.2%, and 0.1% respectively.

The runtime performance of the PF-ECLAT algorithm was compared to that of the PFP-growth, PFP-growth++, and PS-growth algorithms. Figure 2.4 illustrates the results of this comparison, demonstrating that PF-ECLAT outperforms the other state-of-the-art algorithms across all evaluated databases. In each subfigure of Figure 2.4, the vertical axis represents the runtime in milliseconds, while the horizontal axis represents the  $maxPer$  threshold values. Several key observations can be made from the results: (i) The PF-ECLAT algorithm exhibits faster execution times compared to the PS-growth algorithm. This indicates that the periodic calculation approach employed

---

in PF-ECLAT is highly effective, allowing for the efficient pruning of non-periodic patterns. Furthermore, the results indicate that PF-ECLAT outperforms the PFP-growth++ algorithm in terms of runtime. (ii) Generally, for all databases, increasing the *maxPer* threshold value leads to an increase in the runtime of the algorithms. However, in such cases, PF-ECLAT demonstrates superior efficiency compared to the other algorithms, particularly in the BMS-WebView-1, Pollution, Drought, and Congestion databases. (iii) Marginal differences in runtime were observed between PF-ECLAT and the other algorithms in the BMS-WebView-2 and T10I4D100K databases, both of which exhibit a sparse nature with short transactions. Further investigation revealed that the PS-growth algorithm generates *PFPs* quickly by summarizing the database when it has a sparse nature with short transactions, resulting in the marginal runtime improvement. (iv) Overall, the PFP-List structure utilized in the PF-ECLAT algorithm proves to be more compact and efficient compared to the corresponding structure used in the other state-of-the-art algorithms.

The memory consumption of the PF-ECLAT algorithm was compared to that of the PFP-growth, PFP-growth++, and PS-growth algorithms. Figure 2.5 presents the results of this comparison, demonstrating that PF-ECLAT outperforms the other state-of-the-art algorithms across all evaluated databases. In each subfigure of Figure 2.5, the vertical axis represents the memory consumption in kilobytes, while the horizontal axis represents the *maxPer* threshold values. Several important observations can be made from the results: (i) As the *maxPer* threshold increases, the memory requirements of the PFP-growth, PFP-growth++, PS-growth, and PF-ECLAT algorithms also increase. (ii) Across all databases, including those with sparse or dense characteristics and containing short or long transactions, PF-ECLAT consistently exhibits significantly lower memory consumption compared to the other state-of-the-art algorithms at any given *maxPer* value. This difference is particularly prominent at high *maxPer* values. (iii) Furthermore, PF-ECLAT consumes less memory than the PS-growth algorithm, although they are relatively close in some cases. This highlights the efficiency of the PFP-List structure utilized in the PF-ECLAT algorithm, which helps reduce memory usage. Overall, the evaluation results confirm that the PF-ECLAT algorithm offers superior performance in terms of memory consumption when compared to the PFP-growth, PFP-growth++, and PS-growth algorithms.

The number of patterns was measured for different *maxPer* threshold values on each database, and the results are presented in Figure 2.6. The vertical axes of the subfigures represent the number of patterns, while the horizontal axes correspond to the *maxPer* threshold values. It was observed that PFP-growth, PFP-growth++, PS-growth, and PF-ECLAT algorithms generated the same number of *PFPs* in each of the databases. However, the number of patterns increased as the *maxPer* threshold increased. This implies that a higher *maxPer* threshold led to a greater number of patterns being identified as periodic patterns. The results indicate that the PF-ECLAT algorithm effectively captures the periodic nature of patterns in the data, producing a significant number of periodic frequent patterns. These findings demonstrate the usefulness of the *maxPer* threshold in controlling the identification of periodic patterns and further highlight the potential of the PF-ECLAT algorithm in pattern mining tasks.

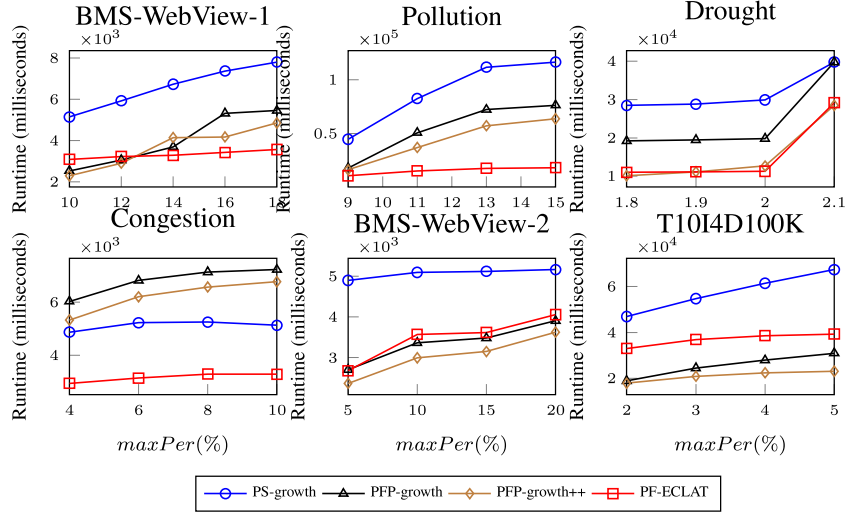


Figure 2.4: The runtime performance of PFP-growth, PFP-growth++, PS-growth, and PF-ECLAT algorithms, was compared across multiple databases with a fixed value of  $maxPer$  and varying values of  $minSup$ .

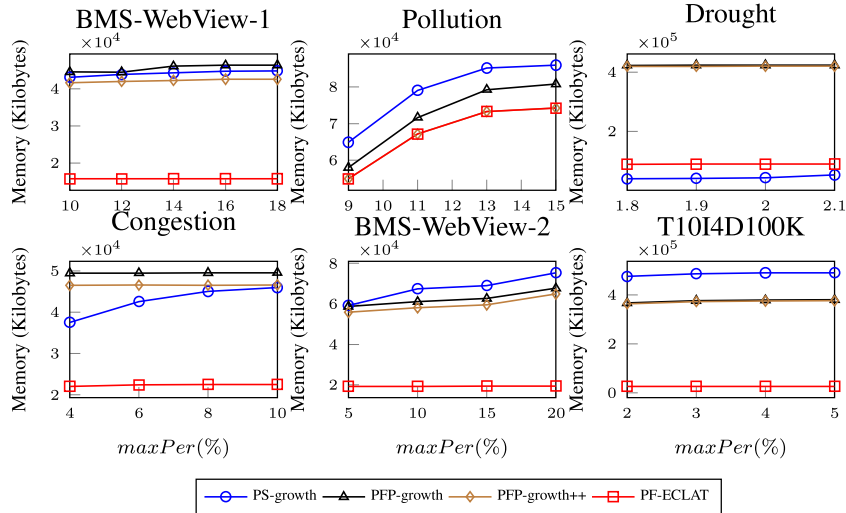


Figure 2.5: The memory usage of PFP-growth, PFP-growth++, PS-growth, and PF-ECLAT algorithms, was compared across multiple databases with a fixed value of  $maxPer$  and varying values of  $minSup$ .

### 2.2.3 Evaluation of PFP-growth, PFP-growth++, PS-growth, and PF-ECLAT algorithms by varying $minSup$ constraint

In this subsection, we evaluate the performance of PFP-growth, PFP-growth++, PS-growth, and PF-ECLAT algorithms by varying only the  $minSup$  constraint in each of the databases. The  $maxPer$  value in each database is set to a specific value, while the  $minSup$  is varied. For the BMS-WebView-1, Pollution, Drought, Congestion, BMS-WebView-2, and T10I4D100K databases, the  $maxPer$  values are set at 40%, 51%, 5%, 35%, 54%, and 20% respectively. By varying the  $minSup$  constraint, we can assess the



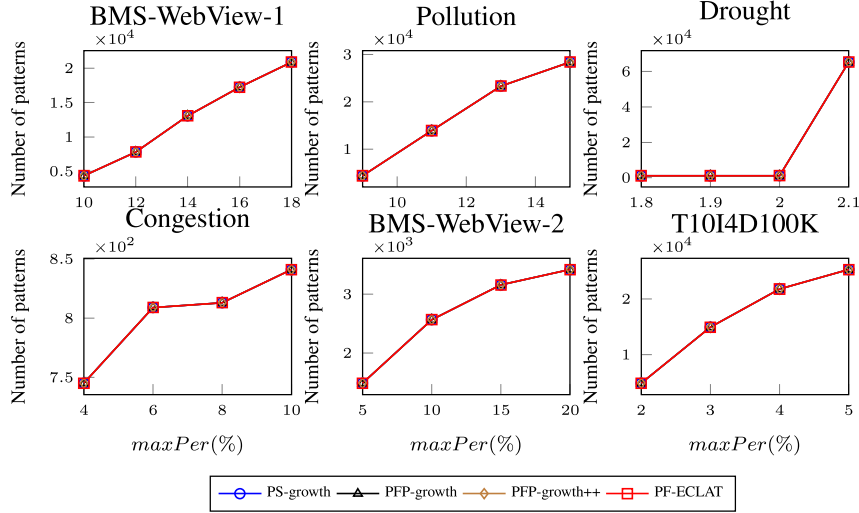


Figure 2.6: The number of *PFPs* generated by the PFP-growth, PFP-growth++, PS-growth, and PF-ECLAT algorithms, was compared across multiple databases with a fixed value of *maxPer* and varying values of *minSup*.

impact of different *support* thresholds on the performance of the algorithms. Through this evaluation, we aim to analyze and compare the runtime, memory consumption, and number of patterns generated by each algorithm for different *minSup* values. The results will provide insights into the scalability, efficiency, and effectiveness of the algorithms in handling varying *support* constraints across different databases.

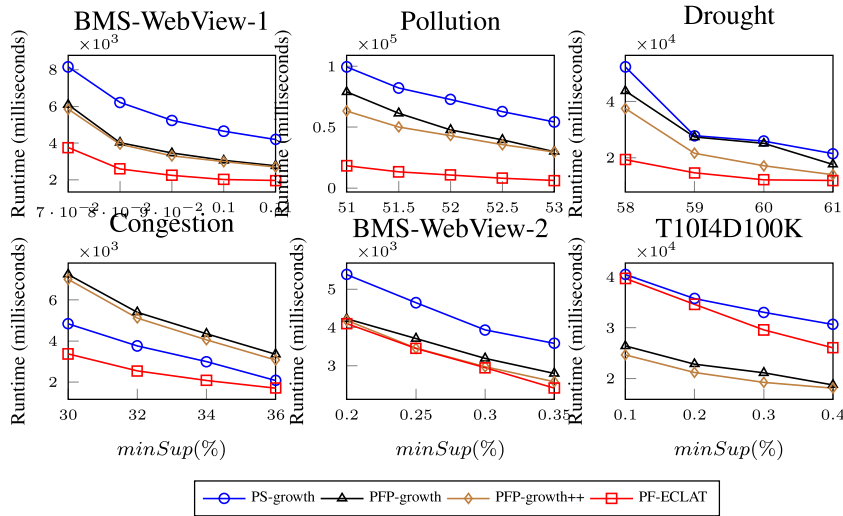


Figure 2.7: The runtime performance of PFP-growth, PFP-growth++, PS-growth, and PF-ECLAT algorithms, was compared across multiple databases with a fixed value of *minSup* and varying values of *maxPer*.

In this experiment, we compare the runtime of the PF-ECLAT algorithm with that of PFP-growth, PFP-growth++, and PS-growth algorithms by varying only the *minSup* constraint in each of the databases. The runtime performance is evaluated by measuring the execution time (in milliseconds) required by each algorithm for different

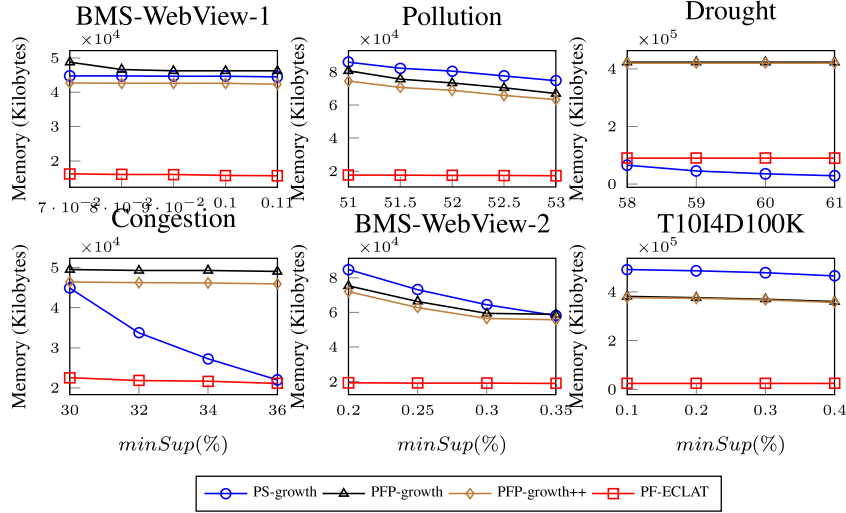


Figure 2.8: The memory usage of PFP-growth, PFP-growth++, PS-growth, and PF-ECLAT algorithms, was compared across multiple databases with a fixed value of  $minSup$  and varying values of  $maxPer$ .

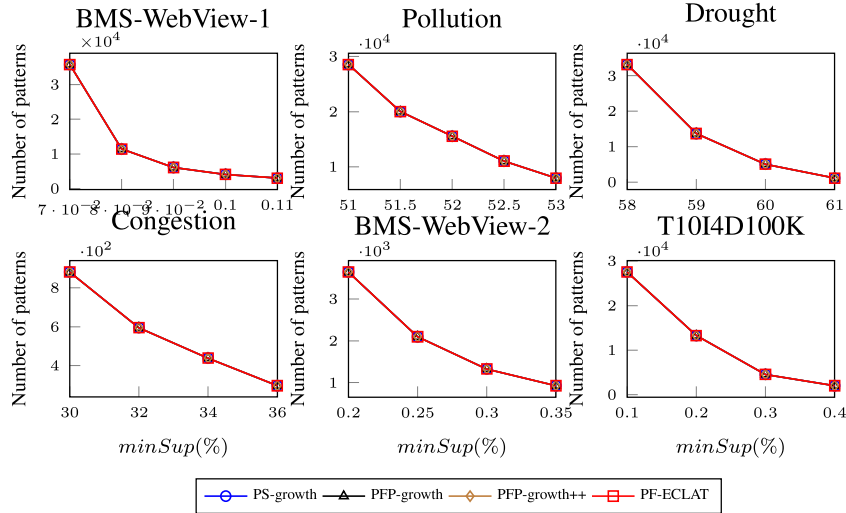


Figure 2.9: The number of patterns generated by the PFP-growth, PFP-growth++, PS-growth, and PF-ECLAT algorithms, was compared across multiple databases with a fixed value of  $minSup$  and varying values of  $maxPer$ .

$minSup$  threshold values. Figure 2.7 illustrates the runtime comparison results, where the vertical axis represents the runtime and the horizontal axis represents the  $minSup$  threshold values in each subfigure. The following observations can be made from this figure: (i) Increasing the  $minSup$  value generally decreases the runtime requirements of all algorithms. This is because a higher  $minSup$  threshold reduces the number of patterns to be generated and, thus, the computation time. (ii) Among the compared algorithms, PF-ECLAT consistently exhibits superior runtime performance across all databases. It significantly outperforms PFP-growth, PFP-growth++, and PS-growth algorithms in terms of runtime efficiency. Even at low  $minSup$  values, PF-ECLAT

### Kosarak database

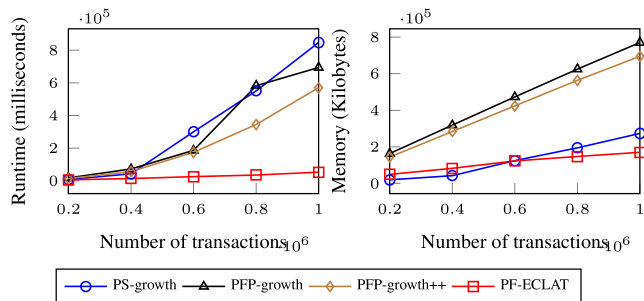


Figure 2.10: Scalability of PFP-growth, PFP-growth++, PS-growth, and PF-ECLAT

demonstrates remarkable speed and efficiency. (iii) Particularly in databases such as BMS-WebView-1, Pollution, Drought, and Congestion, PF-ECLAT achieves substantial runtime advantages over the PS-growth algorithm. The margin of improvement becomes even more significant at higher  $minSup$  values. However, in sparse databases with short transactions like BMS-WebView-2 and T10I4D100K, the runtime difference between PF-ECLAT and PS-growth is relatively small.

In this experiment, we compare the memory consumption of the PF-ECLAT algorithm with that of the PFP-growth, PFP-growth++, and PS-growth algorithms by varying only the  $minSup$  constraint in each of the databases. The memory performance is evaluated by measuring the memory usage (in kilobytes) of each algorithm for different  $minSup$  threshold values. Figure 2.8 illustrates the memory comparison results, where the vertical axis represents the memory usage in kilobytes and the horizontal axis represents the  $minSup$  threshold values in each subfigure. The following observations can be made from this figure: (i) As the  $minSup$  value increases, the memory requirements of all algorithms generally decrease. This is because a higher  $minSup$  threshold leads to a smaller number of patterns, resulting in reduced memory usage. (ii) Across all databases, PF-ECLAT consistently exhibits superior memory performance compared to the other state-of-the-art algorithms. Regardless of whether the database is sparse or dense, and whether it contains short or long transactions, PF-ECLAT consistently consumes significantly less memory than the other algorithms at any given  $minSup$  value. The difference in memory usage is particularly high at low  $minSup$  values. (iii) The effectiveness of the PFP-List structure used in the PF-ECLAT algorithm is evident in the memory comparison results. The compact and efficient representation of patterns in the PFP-List significantly reduces the memory requirements of the algorithm. Overall, these findings demonstrate that the PF-ECLAT algorithm excels in terms of memory efficiency compared to the other state-of-the-art algorithms. It consistently consumes less memory across various databases and  $minSup$  values, making it a preferable choice for memory-constrained environments.

In this evaluation, we analyze the number of patterns generated by the PFP-growth, PFP-growth++, PS-growth, and PF-ECLAT algorithms by varying only the  $minSup$  constraint in each of the databases. Figure 2.9 depicts the results, where the vertical axes represent the number of patterns and the horizontal axes indicate the corresponding  $minSup$  threshold values. From the figure, it can be observed that all four algorithms generate the same number of  $PFPs$  in each of the databases. This indicates that the algorithms are consistent in their ability to discover frequent patterns. Furthermore, as

the  $minSup$  threshold increases, the number of patterns decreases. This is because a higher  $minSup$  value sets a stricter criterion for pattern frequency, requiring patterns to appear more frequently in the database to be considered frequent. As a result, patterns that fail to meet the increased  $minSup$  constraint are filtered out, leading to a reduced number of  $PFPS$ .

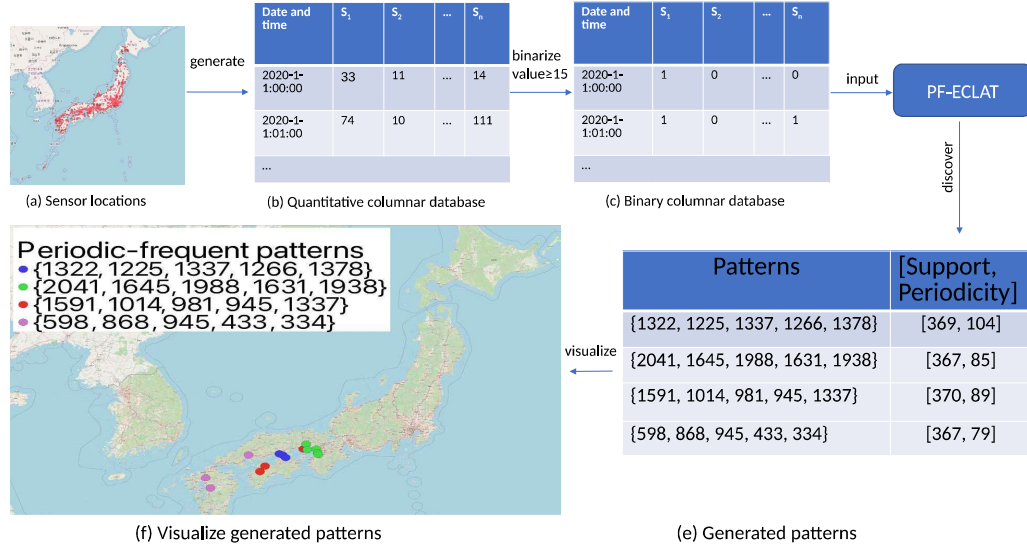


Figure 2.11: Finding  $PFPS$  in the Pollution database. The terms ' $s_1$ ', ' $s_2$ ',  $\dots$ ' ' $s_n$ ' represents 'sensor identifiers'

## 2.2.4 Scalability test

In order to assess the scalability of the PFP-growth, PFP-growth++, PS-growth, and PF-ECLAT algorithms, the Kosarak database was divided into five portions, each containing 0.2 million transactions. The performance of these algorithms was evaluated as each portion was accumulated with the previous portions. Figure 2.10 illustrates the runtime and memory requirements of all algorithms at different database sizes, considering a  $minSup$  value of 1% and a  $maxPer$  value of 0.1%. From the analysis of these figures, the following two observations can be made: (i) The runtime and memory requirements of the PFP-growth, PFP-growth++, PS-growth, and PF-ECLAT algorithms exhibit an almost linear increase with the growth of the database size. This indicates that these algorithms scale well as the size of the database increases. (ii) At any given database size, the PF-ECLAT algorithm demonstrates superior performance in terms of both runtime and memory consumption compared to the other algorithms (i.e., PFP-growth, PFP-growth++, and PS-growth). This highlights the efficiency and scalability of the PF-ECLAT algorithm, making it a favorable choice for mining periodic frequent patterns in large databases.

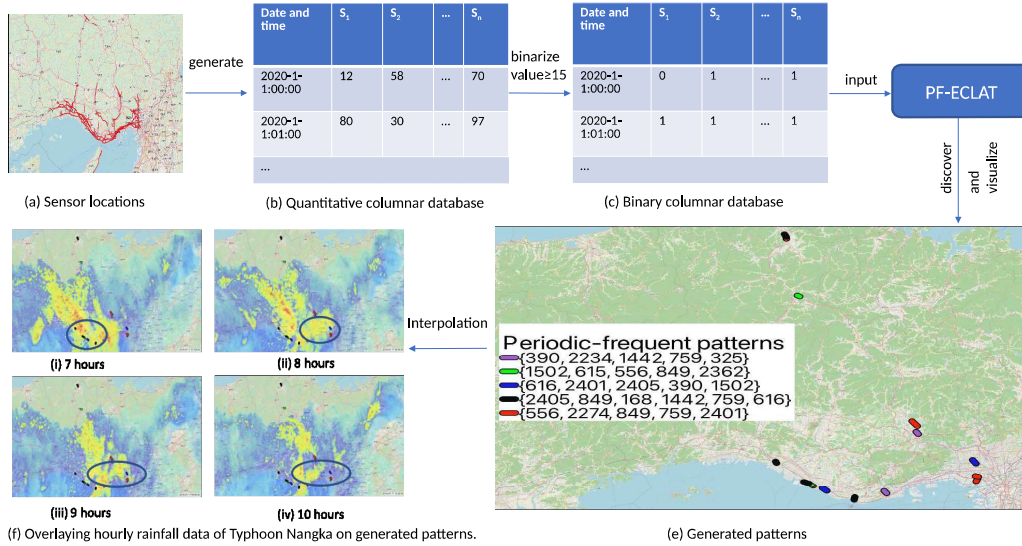


Figure 2.12: Finding *PFPs* in the Congestion database. The terms ‘ $s_1$ ,’ ‘ $s_2$ ,’  $\dots$  ‘ $s_n$ ’ represents ‘road sensor identifiers’

## 2.2.5 A case study 1: finding areas where people have been regularly exposed to hazardous levels of PM2.5 pollutant

The Ministry of Environment in Japan has established the SORAMAME [62], sensor network system to monitor air pollution across the country (shown in Figure. 2.11 (a)). This sensor network generates raw data, which represents quantitative values of pollution levels collected by individual sensors at various locations. The raw data, in the form of a quantitative CTDB (shown in Figure. 2.11 (b)), is transformed into a binary CTDB by considering a threshold value (in this case,  $\geq 15$ ) as shown in Figure. 2.11 (c). This transformation helps identify instances where pollution levels are high. The transformed binary CTDB is then input into the PF-ECLAT algorithm, which employs the downward closure property and efficient mining techniques to identify sets of sensor identifiers (patterns) associated with high pollution levels. These patterns represent areas or time intervals where pollution levels consistently exceeded the threshold. The identified patterns, representing areas of high pollution (shown in Figure. 2.11 (e)), can be further analyzed and visualized geographically. In Figure 2.11 (f), the spatial locations of these interesting patterns are visualized. It can be observed that the majority of the sensors in the southeast region of Japan exhibit high pollution levels periodically. This information provides valuable insights to ecologists and policymakers, enabling them to devise targeted policies and measures to control pollution and improve public health in the identified areas. It’s worth noting that the PF-ECLAT algorithm allows for more in-depth studies, such as analyzing high-polluted areas during specific time intervals or on weekends. This flexibility enables researchers and stakeholders to gain a comprehensive understanding of pollution patterns and their temporal variations, supporting more informed decision-making processes.

### 2.2.6 A Case Study on Traffic Congestion Analytics

In July 2005, Typhoon Nangka caused heavy rainfall and subsequent flooding in Kobe, Japan. To monitor traffic congestion resulting from the evacuation efforts, the JARTIC [60] deployed a sensor network across Japan. The road network in Kobe, Japan, covered by the congestion monitoring sensors, is shown in Figure 2.12(a). The raw data collected by these sensors, representing quantitative values of congestion levels (shown in Figure 2.12(b)), can be transformed into a binary CTDB by applying a threshold value (in this case, greater than or equal to 15). The transformed binary CTDB as show in Figure 2.12(c) is then processed by the PF-ECLAT algorithm to identify patterns consisting of sensor identifiers where traffic congestion is particularly high. The spatial locations of these interesting patterns, generated from the **Congestion** database, are visualized in Figure 2.12(e). Additionally, the rainfall data from Typhoon Nangka during the respective hour can be overlaid on the discovered patterns, as shown in Figure 2.12(f). By analyzing the combined information of the congestion patterns and rainfall data, valuable insights can be gained for effective decision-making in traffic control. For example, traffic control room operators can use this information to divert traffic and suggest alternative routes to users. In Figure 2.12(f), road segments requiring attention are indicated with a black circle, which moves from left to right over a span of 4 hours. This information is highly beneficial for traffic management.

## 2.3 Discussoin about PAMI package

In this section, to ensure the practical applicability and accessibility of PF-ECLAT, we have developed a dedicated Python package, referred to as PAttern MIning (PAMI). The PAMI serves as a comprehensive Python library designed to facilitate efficient and scalable pattern mining. The package encompasses the implementation of the PF-ECLAT algorithm [63], providing researchers and practitioners with a powerful tool for discovering periodic-frequent patterns in large databases. The PAMI package encapsulates key functionalities necessary for effective application of the PF-ECLAT algorithm. These include optimized data structures and algorithms that expedite the mining process, ensuring computational efficiency even for databases of considerable size. Moreover, PAMI offers a user-friendly Github [64] guide that streamlines the integration of PF-ECLAT into existing data mining workflows. Its well-documented comprehensive examples guide users through the installation, configuration, and utilization of the package, ensuring a smooth and intuitive experience.

In addition to the technical aspects, PAMI encourages community engagement and collaboration. The package is open-source and hosted on a publicly accessible repository, facilitating contributions, bug reports, and feature requests from the wider data mining community. This collaborative approach fosters continual improvements, as users can benefit from the expertise and diverse perspectives of fellow researchers and practitioners.

## 2.4 Conclusions

In summary, this chapter introduces the PF-ECLAT algorithm, which efficiently discovers *PFPS* in CTDBs. By incorporating the *minSup* and *maxPer* constraints,

---

the algorithm filters out uninteresting patterns. The utilization of the PFP-List structure further enhances the algorithm's performance by reducing both runtime and memory requirements. Experimental evaluations conducted on various real-world and synthetic databases demonstrate that PF-ECLAT outperforms existing algorithms in terms of PFPM, achieving faster pattern discovery with lower memory usage. The chapter concludes by presenting two case studies, namely AP analytics and TC analytics, that highlight the practical utility of the proposed algorithm.

In the next chapter, we discuss the importance of  $3Ps$  and introduce a model of Partial Periodic Pattern ( $3P$ ) mining. Additionally, we propose a novel ECLAT-based algorithm named 3P-ECLAT to discover  $3Ps$  in CTDBs.



To  
The Graduate School Academic Affairs Committee,  
The University of AIZU, Japan.

Sub: Explanation about the equal contribution statement in my journal paper.

Respected committee members,



I am Penugonda Ravikumar, a doctoral student at the University of AIZU, Japan. My paper "Efficient Discovery of Periodic-Frequent Patterns in Columnar Temporal Databases" was published in the Journal of Electronics 2021. In this paper, I have shared my first authorship with three other co-authors, even though I have proposed the idea, developed the algorithm, evaluated the algorithm on various datasets, and drafted the paper. An explanation for the same is as follows:

The second author, Miss Pala Likhitha, provided me with the air pollution database, which I have used as a case study to demonstrate the usefulness of discovered patterns. Miss Likhitha developed several crawlers to download air pollution data from Japanese websites, wrote ETL (Extraction, Transformation, and Loading) plugins to store the data in a Postgres database, and developed a program to convert non-binary air pollution database into a binary air pollution database useful for my experimentation. I am deeply grateful for her contributions, which were critical to the success of my research project.

The third author, Mr. Bathala Venus Vikranth Raj, was not part of the earlier version of my paper published at the IEA/AIE 2021 conference. However, when converting my conference paper into a journal paper, Mr. Vikranth helped me identify the relevant articles for my research. Furthermore, he assisted me in discussing the literature, where some papers needed help to understand.

The fourth author, Professor Rage Uday Kiran, gave me a congestion database for my research project. The database was critical to my research, which aimed to understand traffic congestion patterns and develop strategies to improve transportation systems. Prof. Rage collected the information from the sensors and transformed it to form a real-world database using a threshold value. In the case study, we have overlaid typhoon Nangka data on generated patterns, and the professor did this as he had access to NICT servers. He also gave me feedback and suggestions to improve my draft, which helped refine my research questions and methodology. His guidance and support throughout the project were invaluable, and I am deeply grateful for his contributions. Without his help, my research project would not have been possible.

However, I now declare that other than me, the remaining three authors will not claim the first authorship position anywhere else. Therefore, please accept me as the paper's first author (Primary contributor). Undersigned by all four authors of the paper:

  
P. Likhitha  
B. Venus Vikranth Raj  


---

Figure 2.13: Explanation about the equal contribution statement in my journal paper



## Chapter 3

# Towards Efficient Discovery of Partial Periodic Patterns in Columnar Temporal Databases

In this chapter, too, the spatial characteristics of the *STDs*, specifically LD, were not taken into account. Instead, the research focused solely on discovering interesting patterns based on the temporal characteristics of the patterns. The main objective of this chapter is to identify *3Ps* in CTDBs.

Periodic patterns in a TDB can be classified as full periodic patterns or *3Ps*. Full periodic patterns are monitored strictly within the database, and uninteresting patterns are discarded based on user-defined constraints such as the *maxPer*. *3Ps*, on the other hand, occur only during specific times and can be found in real-world scenarios. However, these events will happen regularly.

A classical application of *3P* mining is MB analytics. It is used to discover patterns that occur partially regularly in a TDB. *3Ps* reveal that certain items are frequently purchased together on certain days of the week or at certain times of the day.

**Example 9.** *If we consider the department store database (see Table 1.3), most of the customers may purchase bat and ball during the evening hours only and other standard items during the remaining time of the day. However, these might have happened regularly. Therefore, we call these types of purchases made only during this particular time of the day as 3Ps. By identifying these 3Ps, retailers can make informed decisions about when to stock these items and how much inventory to keep on hand. They may also offer special discounts or promotions during these times to encourage customers to purchase more of these items.*

It is often useful to mine *3Ps* in TDBs, as they can provide valuable insights that full periodic patterns may not capture.

In this chapter, we first discuss the model of *3Ps* in TDBs. Next, we discuss the 3P-ECLAT algorithm, which is designed to identify the complete set of *3Ps* in CTDB. After that, we check the performance of the 3P-ECLAT [31] algorithm against the state-of-the-art algorithm by considering both synthetic and real-world databases. Finally, we conclude the chapter by summarizing our findings. This chapter shows the importance of discovering partial periodic patterns over patterns generated in chapter 2. We propose a novel model and introduce an algorithm to efficiently discover these patterns. Finally, we show the importance of the .

Table 3.1: Row database

<i>ts</i>	items
1	ace
2	bc
3	bdef
4	abef
5	acdf
6	abcd

<i>ts</i>	items
7	bcd
8	bf
9	abcd
10	cd
11	abcd
12	abcd

Table 3.2: Columnar database

	items					
<i>ts</i>	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>e</i>	<i>f</i>
1	1	0	1	0	1	0
2	0	1	1	0	0	0
3	0	1	0	1	1	1
4	1	1	0	0	1	1
5	1	0	1	1	0	1
6	1	1	1	1	0	0

	items					
<i>ts</i>	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>e</i>	<i>f</i>
7	0	1	1	1	0	0
8	0	1	0	0	0	1
9	1	1	1	1	0	0
10	0	0	1	1	0	0
11	1	1	1	1	0	0
12	1	1	1	1	0	0

 Table 3.3: List of *ts* of an item

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

### 3.1 The Model of Partial Periodic Pattern

In this section, we first use the few definitions discussed in the previous chapter. We use the Definition 1 and 2 to discover the  $sup(X)$ .

**Example 10.** Consider a set of items  $I = \{a, b, c, d, e, f\}$ , and suppose we have a hypothetical RTDB generated from  $I$ , as depicted in Table 3.1. Without loss of generality, this RTDB can be represented as a CTDBs as shown in Table 3.2. The *ts* of each item in the entire database are shown in Table 3.3. The TDB contains 12 transactions. Therefore,  $m = 12$ . The minimum and maximum *ts* in this database are 1 and 12, respectively. Therefore,  $ts_{min} = 1$  and  $ts_{max} = 12$ . In this database, the set of items ‘*b*’ and ‘*c*’ forms a pattern, which we represent as ‘*bc*’ for brevity. Since this pattern contains two items, it is classified as a 2-pattern. The ‘*bc*’ pattern appears in transactions with *ts*’s : 2, 6, 7, 9, 11, and 12, which yields a list of *ts* containing ‘*bc*’, denoted by  $TS^{bc} = \{2, 6, 7, 9, 11, 12\}$ . The support of ‘*bc*’, i.e.,  $sup(bc) = |TS^{bc}| = 6$ .

**Definition 7. (Periodic appearance of pattern  $X$  [48].)** Let  $ts_j^X$  and  $ts_k^X$ ,  $1 \leq j < k \leq m$ , be the two consecutive *ts* 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 per. That is, a  $iat_k^X \in IAT^X$  is said to be **periodic** if  $iat_k^X \leq per$ .

**Example 11.** The pattern ‘*bc*’ has initially appeared at the *ts* of 2 and 6. Thus, the difference between these two *ts* 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  $IAT^{bc} = \{4, 1, 2, 2, 1\}$ . If the user-specified per = 2, then  $iat_2^{bc}$ ,  $iat_3^{bc}$ ,  $iat_4^{bc}$

---

and  $iat_5^{bc}$  are considered the periodic occurrences of 'bc' in the data. In contrast,  $iat_1^{bc}$  is not considered a periodic occurrence of 'bc' because  $iat_1^{bc} \not\leq per$ .

**Definition 8. (Period-support of pattern  $X$  [48].)** 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 PS of  $X$ , denoted as  $PS(X) = |\widehat{IAT^X}|$ .

**Example 12.** Continuing with the previous example,  $\widehat{IAT^{bc}} = \{1, 2, 2, 1\}$ . Therefore, the PS of 'bc', i.e.  $PS(bc) = |\widehat{IAT^{bc}}| = |\{1, 2, 2, 1\}| = 4$ .

**Definition 9. (Partial periodic pattern  $X$  [48].)** A pattern  $X$  is said to be a 3P if  $PS(X) \geq minPS$ , where  $minPS$  is the user-specified minimum period-support.

**Example 13.** Continuing with the previous example, if the user-specified  $minPS = 4$ , then 'bc' is a 3P because  $PS(bc) \geq minPS$ . The complete set of 3Ps discovered from Table 3.3 including 1-patterns' (in Figure 3.1(f)) are shown in Figure 3.2 without "~~sample~~" (i.e., Strikethrough) mark on the text.

**Definition 10. (Problem definition.)** Given a TDB and the user-specified  $per$  and  $minPS$  constraints, find all 3Ps in TDB that have PS no less than  $minPS$ . The PS of a pattern can be expressed as a percentage of  $(|TDB| - 1)$ . The  $per$  can be expressed as a percentage of  $(ts_{max} - ts_{min})$ . In this chapter, we employ the above definitions of the period and period-support for brevity.

## 3.2 Proposed Algorithm

Although our proposed algorithm can generate interesting patterns using either RTDB or CTDB architecture, existing state-of-the-art algorithms are only capable of generating such patterns in RTDB architecture. As a result, we used RTDB architecture only for evaluation purposes, even though this required additional effort to transform the data from a row to a columnar architecture.

This section first describes the procedure for finding one-length 3Ps (or 1-patterns) and transforming RTDB to CTDB. Next, we will explain the 3P-ECLAT algorithm to discover a complete set of 3Ps in CTDBs. The 3P-ECLAT algorithm employs DFS and the *downward closure property* (see Property 5) of 3Ps to reduce the vast search space effectively.

**Property 5. (The downward closure property [20].)** If  $Y$  is a 3P, then  $\forall X \subset Y$  and  $X \neq \emptyset$ ,  $X$  is also a 3P.

### 3.2.1 3P-ECLAT algorithm

#### Finding one length partial periodic patterns

The Algorithm 3 describes the procedure to find 1-patterns using 3P-list, which is a dictionary. We now describe this algorithm's working using the RTDB shown in Table 3.1. Let  $minPS = 4$  and  $per = 2$ .

P	TS-list	PS	TS <sub>i</sub>
a	1	0	1
c	1	0	1
e	1	0	1

P	TS-list	PS	TS <sub>i</sub>
a	1	0	1
c	1 2	1	2
e	1	0	1
b	2	0	2

P	TS-list	PS	TS <sub>i</sub>
a	1,4,5,6,9,11,12	6	12
c	1,2,5,6,7,9,10,11,12	8	12
e	1,3,4	2	4
b	2,3,4,6,7,8,9,11,12	8	12
d	3,5,6,7,9,10,11,12	7	12
f	3,4,5,8	3	8

P	TS-list
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
a	1,4,5,6,9,11,12

Figure 3.1: Finding 3Ps. (a) after scanning the first transaction, (b) after scanning the second transaction, (c) after scanning the entire database, and (d) final list of 3Ps sorted in descending order of their  $PS$  (or the size of TS-list) with the constraint  $minPS = 4$  and  $per = 2$

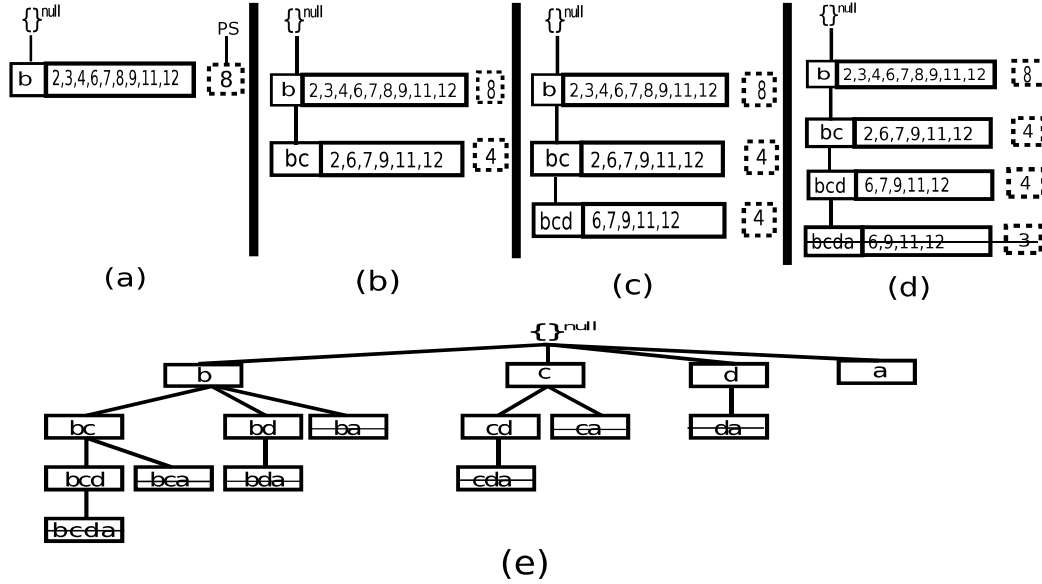


Figure 3.2: Mining 3Ps using DFS.

We will scan the complete database once to generate 1-3Ps and transforming the RTDB to CTDB. The scan on the first transaction, “1 : ace”, with  $ts_{cur} = 1$  inserts the items  $a$ ,  $c$ , and  $e$  in the 3P-list. The  $ts$  of these items is set to 1 ( $= ts_{cur}$ ). Similarly,  $PS$  and  $TS_i$  values of these items were also set to 0 and 1, respectively (lines 5 and 6 in Algorithm 3). The 3P-list generated after scanning the first transaction is shown in Figure 3.1(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  $TS_i$  values were set to 0 and 2, respectively. On the other hand, 2 ( $= ts_{cur}$ ) was added to the TS-list of an already existing item,  $c$  with  $PS$  and  $TS_i$  set to 1 and 2, respectively (lines 7 and 8 in Algorithm 3). The 3P-list generated after scanning the second transaction is shown in Figure 3.1(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 Figure 3.1(c). The patterns  $e$  and  $f$  are pruned (using the Property 5) from the 3P-list as its  $PS$  value is less than the user-specified  $minPS$  value (lines 10 and 11 in Algorithm 3). The remaining patterns in the 3P-list are considered 3Ps and sorted in descending order of their  $PS$  values. The final 3P-list generated after sorting the 3Ps is shown in Figure 3.1(d).

---

**Algorithm 3** 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  $i$  be 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 the 3P-list as partial periodic items. Sort these items in *support* descending order. Let  $L$  denote this sorted list of partial periodic items.
- 

### Finding partial periodic patterns using the 3P-list.

Algorithm 4 describes the procedure for finding all  $3Ps$  in a CTDB. We now describe the workings 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 4). We record its  $PS$ , as shown in Figure 3.2(a). Since  $b$  is a  $3P$ , 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 4). We record  $PS$  of ‘ $bc$ ’, as shown in Figure 3.2(b). We verify whether ‘ $bc$ ’ is  $3P$  or uninteresting pattern (line 5 in Algorithm 4). Since ‘ $bc$ ’ is  $3P$ , 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 Figure 3.2(c) and identified it as a  $3P$ . 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  $3Ps$  list as shown in Figure 3.2(d).

A similar process is repeated for remaining nodes in the set-enumeration tree to find all  $3Ps$ . The final list of  $3Ps$  generated from Table 3.1 is shown in Figure 3.2(e). The above approach to finding  $3Ps$  using the downward closure property is efficient because it effectively reduces the search space and the computational cost.

Table 3.4: 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 4** 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 3P;
  - 8:   3P-ECLAT( $pi$ )
- 

### 3.3 Experimental Results

In this section, we first compare the 3P-ECLAT against the state-of-the-art algorithm 3P-growth [48,65] 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, 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 3.4. **The complete evaluation results, databases, and algorithms have been provided through GitHub<sup>1</sup> to verify the repeatability of our experiments.** We are not providing the Congestion databases on GitHub due to confidentiality reasons.

#### 3.3.1 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 the T20I6d100K, Congestion, and Pollution databases have been set at 60%, 50%, and 50%, respectively.

Figure 3.3 shows the number of 3Ps generated in the 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 3Ps. It is because many patterns fail to satisfy the increased  $minPS$ .

---

<sup>1</sup><https://github.com/udayRage/PAMI/tree/main/PAMI/partialPeriodicPattern/basic>

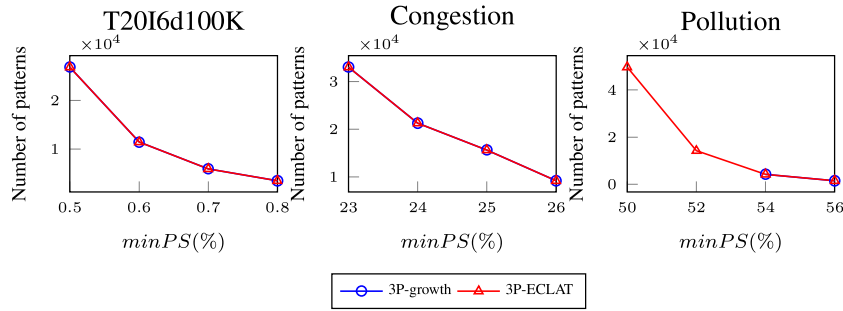


Figure 3.3: Patterns evaluation of 3P-growth and 3P-ECLAT algorithms at constant  $per$

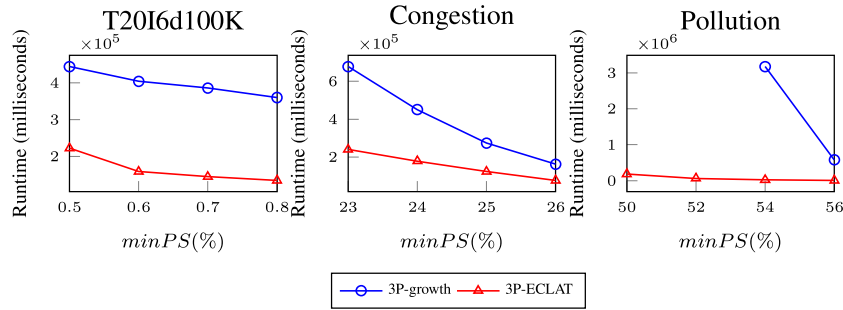


Figure 3.4: Runtime evaluation of 3P-growth and 3P-ECLAT algorithms at constant  $per$

Figure 3.4 shows the runtime requirements of 3P-growth and 3P-ECLAT algorithms in the T20I6d100K, Congestion, and Pollution databases at different  $minPS$  values. It can be observed that, even though the runtime requirements of both 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 3.5 shows the memory requirements of 3P-growth and 3P-ECLAT algorithms in the T20I6d100K, Congestion, and Pollution databases at different  $minPS$  values. It can be observed that, though an increase in  $minPS$  resulted in a decrease in memory requirements for both algorithms, the 3P-ECLAT algorithm has consumed relatively little memory in all databases at different  $minPS$  values. More importantly, 3P-growth has taken up a huge amount of memory, especially at low  $minPS$  values in all of the databases, and ran out of memory in the Pollution database.

### 3.3.2 Evaluation of algorithms by varying $per$

In this experiment, we evaluate 3P-growth and 3P-ECLAT algorithms performance by varying only the  $per$  constraint in Congestion database. The  $minPS$  is set at 23% during the evaluation.

Figure 3.6 first graph shows the number of  $3Ps$  generated in Congestion database at different  $per$  values. It can be observed that an increase in  $per$  has increased the number of  $3Ps$  in both of the algorithms.

Figure 3.6 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

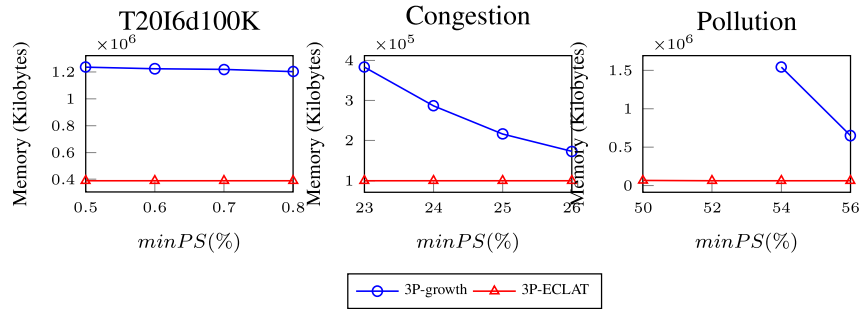


Figure 3.5: Memory evaluation of 3P-growth and 3P-ECLAT algorithms at constant  $per$

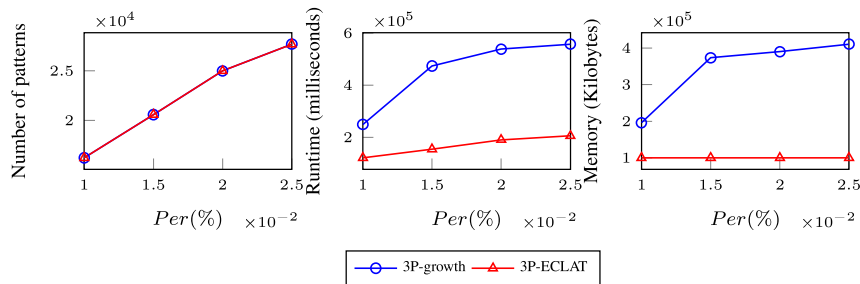


Figure 3.6: Evaluation of 3P-growth and 3P-ECLAT algorithms using Congestion database

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 3.6 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.

### 3.3.3 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 portions. Figure 3.7 shows the runtime 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.



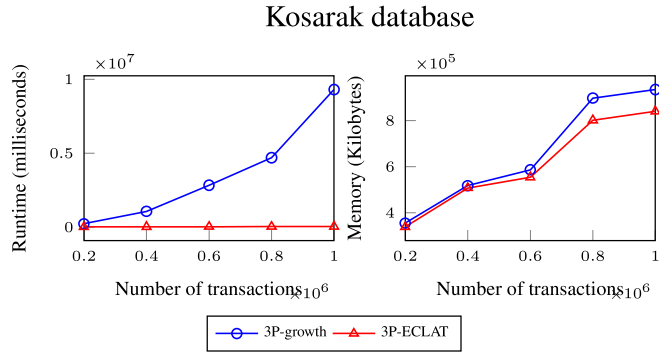


Figure 3.7: Scalability of 3P-growth and 3P-ECLAT

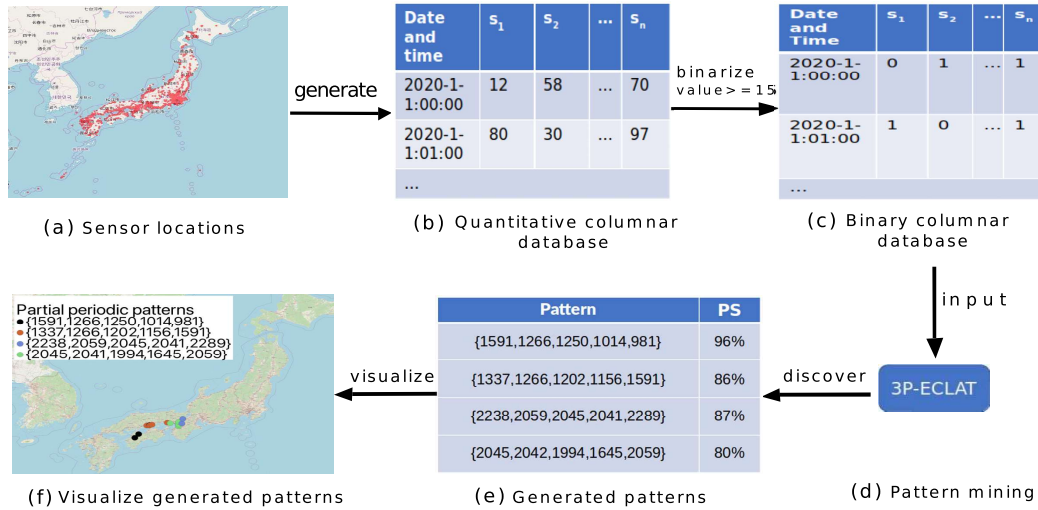


Figure 3.8: Finding 3Ps in Pollution data. The terms ' $s_1$ ', ' $s_2$ ',  $\dots$ ' ' $s_n$ ' represents 'sensor identifiers' and ' $PS$ ' represents 'period-support'

### 3.3.4 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 SO-RAMAME, to monitor air pollution throughout Japan, as shown in Figure 3.8(a). The raw data produced by these sensors, i.e., quantitative CTDB (see Figure 3.8(b)), can be transformed into a binary CTDB, if the raw data value is  $\geq 15$  (see Figure 3.8 (c)). The transformed data is provided to 3P-ECLAT algorithm (see Figure 3.8(d)) to identify all sets of sensor identifiers in which pollution levels are high (see Figure 3.8(e)). The spatial locations of interesting patterns generated from the **Pollution** database are visualized in Figure 3.8(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 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 during particular time intervals of a day, can also be efficiently carried out with our algorithm.

## 3.4 Discussion about PAMI package

We have also extended our efforts to ensure the practical applicability and accessibility of the 3P-ECLAT [66] algorithm by incorporating it into the PAMI package. The PAMI package, named PAttern MIning (PAMI), is a comprehensive Python library designed to facilitate efficient and scalable pattern mining. Within the package, we have implemented the 3P-ECLAT algorithm, which introduces a novel approach to address the limitations of the state-of-the-art 3P-growth algorithm. By integrating 3P-ECLAT into the PAMI package, we provide researchers and practitioners with a powerful tool for efficient 3P mining in diverse columnar databases.

The PAMI package encapsulates key functionalities necessary for the effective application of 3P-ECLAT. It leverages optimized data structures and algorithms that expedite the mining process, ensuring computational efficiency even for large-scale databases. Researchers and practitioners can benefit from the seamless integration of 3P-ECLAT into the PAMI package, which provides intuitive well documented analysis of the results. Through the well-documented comprehensive examples and user-friendly Github guide offered by PAMI, users can easily navigate the installation, configuration, and utilization of 3P-ECLAT, fostering a smooth and intuitive experience.

In addition to its technical aspects, PAMI fosters community engagement and collaboration. The package is open-source and hosted on a publicly accessible repository, encouraging contributions, bug reports, and feature requests from the wider data mining community. This collaborative approach ensures continuous improvements as users can tap into the expertise and diverse perspectives of fellow researchers and practitioners, enhancing the overall effectiveness and functionality of 3P-ECLAT within the PAMI package.

In addition to the technical aspects, PAMI encourages community engagement and collaboration. The package is open-source and hosted on a publicly accessible repository, facilitating contributions, bug reports, and feature requests from the wider data mining community. This collaborative approach fosters continual improvements, as users can benefit from the expertise and diverse perspectives of fellow researchers and practitioners.

## 3.5 Conclusions

This chapter has proposed an efficient algorithm named 3P-ECLAT to find  $3Ps$  in CTDBs. 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  $3P$  mining and can obtain all  $3Ps$  faster and with less memory usage than 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.

Finally, in the next chapter, we will have considered the spatial characteristics of the patterns along with their frequency and temporal characteristics, discussed the model of  $GFPF$  mining, and proposed a novel  $GFPF$ -Miner algorithm to discover  $GFPFs$  in  $GTSDs$ .

## Chapter 4

# Discovering Geo-referenced Periodic-Frequent Patterns in Geo-referenced Time Series Databases

A *GTSD* is a type of spatiotemporal database [56] that stores time-stamped data that is associated with specific geographical locations. The geo-referencing component of this type of database ensures that each data point is accurately located in geographic space, while the time series component allows for the analysis of temporal patterns and trends in the data. By combining these two components, a *GTSD* provides a powerful tool for analyzing and visualizing complex spatiotemporal data.

A classical application is air pollution analysis, which involves discovering the set of sensor identifiers whose pollution levels are periodically at danger levels and who are close in their proximity.

**Example 14.** *Air pollution is the primary cause of several respiratory diseases. In Japan, the Atmospheric Environmental Regional Observation System [67] has deployed a network of sensors across the country to monitor pollution levels in various locations or spatial identities. This research focuses on identifying the spatial identities of sensors that exceed a user-specified pollution threshold. Figure 4.1 (a) illustrates the spatial locations of the sensors associated with high pollution levels. The data generated by these sensors, depicted in Figure 4.1 (b), represents a significant volume of information commonly referred to as a spatiotemporal big data (GTSD). By performing pattern mining on this database, as shown in Figure 4.1 (c), users can identify patterns of neighboring sensor locations that exhibit high levels of air pollutants, specifically focusing on pollutants like  $PM_{2.5}$ . The acquired patterns from the Pollution database, depicted in Figure 4.1 (d), provide valuable insights into the spatial regions associated with high air pollutant concentrations. These patterns play a crucial role in assisting ecologists and policymakers in formulating effective measures to control air pollutants, particularly targeting pollutants like  $PM_{2.5}$ . By leveraging this information, it becomes possible to enhance living standards and minimize environmental damage.*

Periodic pattern mining algorithms, which were discussed in the earlier chapters, have been widely used for identifying periodic patterns in spatiotemporal databases. Unfortunately, those approaches have only considered the support and temporal information of patterns while disregarding their spatial information. This is a significant limitation of the aforementioned algorithms in Chapter 2 and 3.

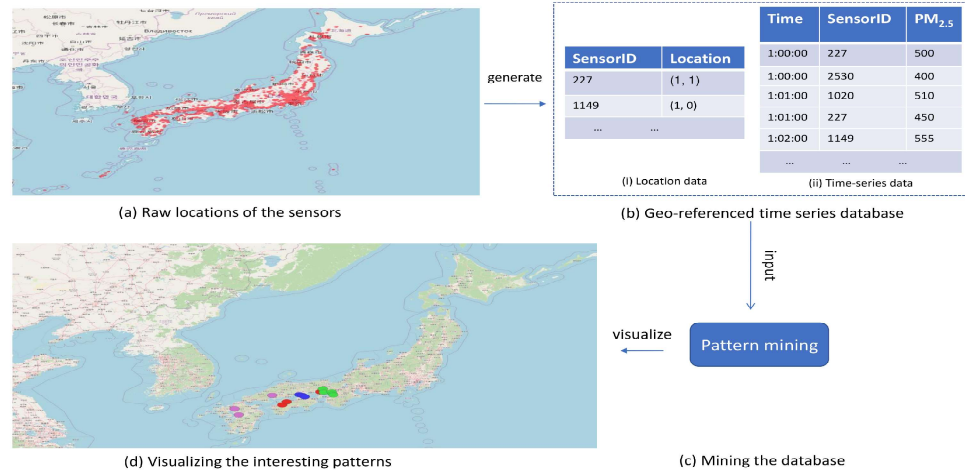


Figure 4.1: Air pollution analytics. The terms ‘SensorID’ and ‘ $PM_{2.5}$ ’ represent ‘Sensor Identifier’ and ‘value of the air pollutant,’ respectively

Table 4.1: Location (or Geo-referential) database

Item	Point	Item	Point
$p$	(2,3)	$s$	(2,3)
$q$	(6,8)	$t$	(1,5)
$r$	(1,4)	$u$	(3,4)

In this chapter, we discuss the model of  $GPFPS$  in  $GTSDs$ . Next, we discuss the  $GFP$ -Miner [32] algorithm, which is designed to identify the complete set of  $GPFPS$  in  $GTSD$ . After that, we check the performance of the  $GFP$ -Miner [31] algorithm against the state-of-the-art algorithm by considering both synthetic and real-world databases. Finally, we conclude the chapter by summarizing our findings.

## 4.1 Proposed Model

This section begins by introducing the concept of  $GTSD$ , which can be viewed as a combination of two fundamental data bases: Location database (LD) and time series data (TSD). LD refers to data associated with specific spatial locations or regions, such as the geographic coordinates of the items in a database. On the other hand, TSD captures data collected over time, such as the temporal variations in a database. To effectively analyze and extract meaningful insights from  $GTSD$ , we introduce the model of  $GPFPS$ .

### 4.1.1 Location database and time series database

Let  $I = \{i_1, i_2, \dots, i_n\}$ ,  $n \geq 1$ , be a set of items. A LD, denoted as  $LD$ , is a collection of items and their coordinates. That is,  $LD = \cup_{i_j \in I} (i_j, Coor_{i_j})$ , where  $Coor_{i_j}$  represent the set of coordinates of an item  $i_j \in I$ . Please note that the coordinates of an item can represent a point, a line, or a polygon. A TSD, denoted as TSD, is a set of events. Each event represents,  $ts$  and  $items$ . That is,  $TSD = \cup_{ts \in \mathbb{R}^+} \cup_{j=1}^n (ts, i_j)$ , where  $ts \in$

Table 4.2: Time series database. The items whose values were equal to 0 at a particular timestamp were removed for brevity

ts	items	ts	item
1	$p, q, r, s$	6	$p, q, r, s$
2	$r, s, t$	7	$p, q$
3	$p, q, r, u$	8	$t, u$
4	$p, s, t$	9	$r, s$
5	$q, r, s, t, u$	10	$p, q, r, s, t, u$

$\mathbb{R}+$  represents the timestamp. For the sake of brevity, the TSD can also be represented by grouping the events with respect to a timestamp as follows: An (irregular) **TSD** is a collection of transactions. That is,  $\text{TSD} = \{t_i, t_j, \dots, t_k\}, i \leq j \leq k \leq |\text{TSD}|$ , where  $t_k = (ts, Y)$ , where  $Y \subseteq I$  is a pattern. and  $|\text{TSD}|$  represents the size of the database. If a pattern  $X \subseteq Y$ , it is said that  $X$  occurs in transaction  $t_k$ . The  $ts$  of this transaction is denoted as  $ts_k^X$ . Let  $TS^X = \{ts_i^X, ts_j^X, \dots, ts_k^X\}, i, j, k \in (1, |\text{TSD}|)$ , denote the set of all  $ts$  in which the pattern  $X$  has appeared in the database.

**Example 15.** We have a set of sensor identifiers (or items),  $I$ , which includes  $p, q, r, s, t$ , and  $u$ . The spatial locations of these items are represented in Table 4.1. In Table 4.2, we have a hypothetical TSD that contains transactions of these items. Each transaction represents a specific  $ts$  and a set of items. For example, in the first transaction,  $ts$  is '1', and the set of items is  $\{p, q, r, s\}$ . This transaction indicates a high level of pollution for the sensors  $p, q, r$ , and  $s$  at  $ts$  '1'. Similar information can be inferred from the remaining transactions in Table 4.2. The size of this temporal database is given by  $|\text{TSD}| = 10$ . Moreover, the complete set of timestamps ( $ts$ ) at which the pattern ' $r, s$ ' has occurred in Table 4.2 is represented by  $TS^{rs} = \{1, 2, 5, 6, 9, 10\}$ .

#### 4.1.2 Model of Geo-referenced Periodic-Frequent Patterns

Both TDBs and TSDs deal with  $ts$  data, but TDBs are more general-purpose and can handle any type of temporal data (such as historical data,  $ts$  data, and data with effective dates), while TSDs are specialized for handling  $ts$  data exclusively (such as sensor data, financial data, and log data). In this research, we have used TSDs which is quite equivalent to TDBs. Therefore, in this section, we use the few definitions discussed in the Chapter 1. We use the Definition 2, 3, 4, and 5 to discover the *PFPs*.

**Example 16.** The support of ' $r, s$ ', denoted as  $sup(rs)$ , is calculated as the cardinality of  $TS^{rs}$ , which is  $sup(rs) = |\{1, 2, 5, 6, 9, 10\}| = 6$ . If the user-specified  $minSup$  is 3, then ' $r, s$ ' is considered a frequent pattern because  $sup(rs) \geq minSup$ . The periods for this pattern are:  $p_1^{rs} = 1$  ( $= 1 - ts_{initial}$ ),  $p_2^{rs} = 1$  ( $= 2 - 1$ ),  $p_3^{rs} = 3$  ( $= 5 - 2$ ),  $p_4^{rs} = 1$  ( $= 6 - 5$ ),  $p_5^{rs} = 3$  ( $= 9 - 6$ ),  $p_6^{rs} = 1$  ( $= 10 - 9$ ), and  $p_7^{rs} = 0$  ( $= ts_{final} - 10$ ). Here,  $ts_{initial} = 0$  represents the  $ts$  of the initial transaction, and  $ts_{final} = |\text{TSD}| = 10$  represents the  $ts$  of the final transaction in the database. The periodicity of ' $r, s$ ', denoted as  $per(r, s)$ , is calculated as the maximum value among the periods:  $= \text{maximum}(1, 2, 3, 1, 3, 1, 0) = 3$ . If the user-defined  $maxPer$  is 4, then the frequent pattern ' $r, s$ ' is considered a *PFP* because  $per(r, s) \leq maxPer$ .

**Definition 11. (Geo-referenced periodic-frequent pattern  $X$ .)** A *PFP*  $X$  is considered a *GPFP* if the maximum distance between any two items in  $X$  is less than or equal to

Table 4.3: Neighbors

Item	Neighbor list
$r$	$s, p, u$
$s$	$r, p, u$
$p$	$r, s, u$
$q$	$t, u$
$t$	$q, u$
$u$	$r, s, p, q, t$

the user-specified  $maxDist$ . In other words,  $X$  is a GPFPP if  $\max(Dist(i_p, i_q) | \forall i_p, i_q \in X) \leq maxDist$ , where  $Dist()$  is a distance function (e.g., Euclidean distance). The condition ensures that the maximum distance between any two items in  $X$  is within the specified threshold  $maxDist$ .

**Example 17.** The pattern ‘ $r, s$ ’ is a GPFPP because the  $maxDist$  between the items  $r$  and  $s$ , i.e.,  $\max(Dist(r, s))$ , is less than or equal to the specified  $maxDist$ . On the other hand, the pattern  $rst \supset rs$  is not a GPFPP because  $Dist(r, t)$  is not less than or equal to  $maxDist$ , or in other words,  $\max(Dist(r, s), Dist(r, t), Dist(s, t))$  is not less than or equal to  $maxDist$ . The Table 4.3 shows the complete list of neighbors for all the one-length patterns.

In various research studies, different distance functions, such as the Euclidean and Geodesic distances, have been utilized to calculate the separation between two items. However, it is important to consider the user’s or application’s requirements before choosing an appropriate distance function. For the purpose of this study, we represented geo-referenced items as points and opted for the Euclidean distance function for the sake of simplicity.

**Definition 12. (Problem definition.)** The problem of GPFPP mining involves finding all patterns in a TSD that satisfy certain criteria. Specifically, we aim to discover patterns with a support value greater than or equal to  $minSup$ , a periodicity value less than or equal to  $maxPer$ , and a maximum distance between any two items in the pattern that does not exceed  $maxDist$ .

## 4.2 Proposed algorithm

In this section, we commence by outlining the initial approach, a naïve algorithm, employed for the identification of GPFPPs, highlighting its inherent limitations. Subsequently, we propose a novel strategy aimed at mitigating these limitations by effectively reducing both the search space and computational overhead associated with GPFPPs identification. Finally, we introduce our meticulously designed algorithm, specifically tailored to facilitate the comprehensive detection of all GPFPPs within a given database.

### 4.2.1 Naïve algorithm

The process of discovering GPFPPs in a GTSD typically involves a naïve algorithm consisting of two steps. In the first step, all PFPs are generated from the database

---

using the PF-ECLAT algorithm [30]. This step aims to extract patterns that meet the  $minSup$  threshold. In the second step, the generated  $PFPs$  are further processed to identify  $GFPFs$  by pruning those patterns that are deemed uninteresting based on a distance measure. However, it is evident that this naïve two-step algorithm suffers from significant inefficiencies. The main issues lie in the vast search space it explores and the generation of a large number of  $PFPs$ , many of which may hold little interest to the user. Consequently, the algorithm’s computational cost becomes impractical, hindering its usability and effectiveness.

In the subsequent subsection, we present our fundamental concept aimed at significantly reducing the search space and computational overhead associated with discovering  $GFPFs$ .

## 4.2.2 Basic idea

When searching for  $GFPFs$  inside a database, the search space is represented by an itemset lattice. As a result, the search space has a size of  $2^{|n|} - 1$  itemsets, where  $n$  represents the total number of items in a database. To identify  $GFPFs$  efficiently, one tough challenge that must be tackled is reducing this enormous search space. We tackle this challenging problem by exploiting the *downward closure property* of  $GFPFs$  (see Property 6). In particular, we exploit two pruning techniques to reduce the search space and the computational cost of finding  $GFPFs$ . We now describe these two pruning techniques.

**Definition 13. (Neighborhood-based pruning)** Let  $X$  and  $Y$  be two patterns such that  $Y = X \cup i_j$ , where  $i_j \in I$  and  $i_j \notin X$ . If  $X$  is a  $GFPF$ , then  $Y$  can be a candidate  $GFPF$  if and only if  $i_j$  is a neighbor of every item in  $X$ . That is,  $Y$  can be a candidate  $GFPF$  if and only if  $Dist(i_j, i_k) \leq maxDist$ , where  $i_k \in X$ .

**Example 18.** Consider the items and their neighborhood information in Table 4.3. We start with item  $r$ . If  $r$  is an interesting pattern, then we perform DFS by considering only the neighbors of  $r$ , i.e.,  $s$ ,  $p$ , and  $u$ . If  $ru$  an interesting pattern, then we perform DFS (or expand the  $ru$  pattern) by considering the common neighbors of  $r$  and  $u$ , i.e.,  $s$  and  $p$ . We do not perform DFS by considering the items  $q$  and  $t$ , which are the neighbors of  $u$ . It is because they are not neighbors of  $r$ . Thus, this neighborhood-based pruning (or expansion) technique can effectively reduce the search space and  $GFPF$  mining practicable in real-world applications.

**Definition 14. (Support and Periodicity-based pruning:)** If  $X$  is not a  $PFP$ , then  $Y \supset X$  cannot be a  $PFP$  [20].

We use both of these pruning techniques to effectively reduce the search space.

**Property 6. (The downward closure property of  $GFPFs$ .)** If  $Y$  is a  $GFPF$ , then  $\forall X \subset Y$  and  $X \neq \emptyset$ ,  $X$  is a  $GFPF$ . The correctness of this property is based on Properties 7 and 8.

**Property 7.** If  $X \subset Y$ , then  $sup(X) \geq sup(Y)$  and  $per(X) \leq per(Y)$ .

**Property 8.** If  $X$  is not a  $GFPF$ , then  $Y \supset X$  cannot be a  $GFPF$ . The correctness is based on Property 9 and shown in Lemma 2.

**Example 19.** In Table 4.2,  $prsu$  is a GFPF. Thus, all its non-empty subsets, i.e.,  $rs$ ,  $rp$ ,  $ru$ ,  $sp$ ,  $su$ ,  $pu$ ,  $rsp$ ,  $rsu$ ,  $rpu$  and  $spu$  must also be geo-referenced periodic-frequent patterns. Now consider the pattern  $pt$ . Since  $Dist(p, t) \not\leq maxDist$ ,  $pt$  is not a geo-referenced pattern. Furthermore, all supersets of  $pt$  cannot be GFPFs. Thus, we can effectively reduce the search space by pruning uninteresting GFPFs.

**Lemma 2.** If  $X$  is a GFPF, then  $\forall Y \subset X$  and  $Y \neq \emptyset$ ,  $Y$  is also a GFPF. The correctness is straight forward to prove from Property 9.

**Property 9.** If  $X$  is a GFPF, then every item  $p \in X$  is a neighbor to another item  $q \in X$ , where  $p \neq q$ . That is, if  $X$  is a GFPF, then  $\forall p, q \in X$ ,  $Dist(p, q) \leq maxDist$ , where  $p \neq q$ .

### 4.2.3 GFPF-Miner

Our GFPF-Miner basically involves the following two steps:

1. Since GFPFs have the property of downward closure, one-length GFPFs, also called 1-patterns, are crucial to finding all interesting GFPFs efficiently. So, we start by getting one-length GFPFs from the database.
2. In the next step, we find the complete set of GFPFs by recursively mining the GFPFs in a DFS fashion.

In the following subsection, we will provide a detailed explanation of both of these steps.

#### Finding one-length geo-referenced periodic-frequent patterns

The technique for discovering 1-patterns using GFPF-list, which is a dictionary, is outlined in Algorithm 5. Now, with the help of the temporal database shown in table 4.2, we will explain how this method works. First, let's assume that  $minSup$  is 3,  $maxPer$  is 3, and  $maxDist$  is 3.

We will scan the complete database once to generate 1-patterns from the TSD. The scan on the first transaction, "1 :  $pqr$ s", with  $ts_{cur} = 1$  inserts the items  $p$ ,  $q$ ,  $r$ , and  $s$  in the GFPF-list. The timestamps of these items are set to 1 ( $= ts_{cur}$ ). Similarly,  $per$  and  $TS_i$  values of these items were also set to 1 and 1, respectively (lines 5 and 6 in Algorithm 5). The GFPF-list generated after scanning the first transaction is shown in Figure 4.2(a). The scan on the second transaction, "2 :  $rst$ ", with  $ts_{cur} = 2$  inserts the new item  $t$  into the GFPF-list by adding 2 ( $= ts_{cur}$ ) in their TS-list. Simultaneously, the  $per$  and  $TS_i$  values were set to 2 and 2, respectively. On the other hand, 2 ( $= ts_{cur}$ ) was added to the TS-list of already existing items  $r$  and  $s$  with  $per$  and  $TS_i$  set to 1 and 2, respectively (lines 7 and 8 in Algorithm 5). The GFPF-list generated after scanning the second transaction is shown in Figure 4.2(b). The other transactions in the database go through a procedure that is quite similar to this one. The complete GFPF-list obtained after searching through the full database can be shown in Figure 4.2(c). All the patterns have satisfied the  $minSup$  and  $maxPer$  constraints. Hence, no pattern is pruned from the GFPF-list. All the items on the GFPF-list are the items that are deemed to be one-length GFPF and arranged in increasing order of their  $support$  values. The complete GFPF-list can be seen in Figure 4.2(d). Now for each one-length pattern  $i_j$  available in



i	TS-list	per	ts <sub>1</sub>
p	1	1	1
q	1	1	1
r	1	1	1
s	1	1	1

i	TS-list	per	ts <sub>1</sub>
p	1	1	1
q	1	1	1
r	1,2	1	2
s	1,2	1	2
t	2	2	2

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

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

Figure 4.2: Finding one-length *GFPs*. (a) Content of the list after reading the 1<sup>st</sup> transaction, (b) after reading the 2<sup>nd</sup> transaction, (c) Final content after reading the whole database, and (d) The complete list of one-length *GFPs* sorted in increasing order of *support* (or the size of *ts-list*).

the *GFP-list*, we calculate the corresponding neighbors list, say  $N(i_j)$ . We will only focus on immediate supersets whose items have common neighbors with all of the items in a pattern.

### Finding geo-referenced periodic-frequent patterns using the *GFP-list* and neighboring lists.

The process of extracting all *GFPs* from a *GTSD* is outlined in Algorithm 6. We will explain how this algorithm works using the *GFP-list* and neighboring lists.

We start with item ‘*r*’, which is the first pattern in the *GFP-list* (line 2 in Algorithm 5). We record its *support* and *periodicity*, as shown in Figure 4.3(a). Since ‘*r*’ is a *GFP*. We check its neighbor list from the Table 4.3, i.e.,  $N(r) = s, p, u$ . We create pattern *r*’s childs from the only items present in the  $N(r)$  (Algorithm 6 Line 3). Therefore, we move to its child node ‘*r,s*’ and generate its *TS-list* by performing the intersection of *TS-lists* of ‘*s*’ and ‘*r*’, i.e.,  $TS^{r,s} = TS^r \cap TS^s$  (lines 4 to 7 in Algorithm 6). We record *support* and *periodicity* of ‘*r,s*’, as shown in Figure 4.3(b). We verify whether ‘*r,s*’ is *GFP* or an uninteresting pattern (line 8 in Algorithm 6). Since ‘*rs*’ is *GFP*, we create *rs*’s neighbor list by performing the intersection of neighbor lists of ‘*s*’ and ‘*r*’, i.e.,  $N(r, s) = N(r) \cap N(s) = p, u$  and it is shown in Figure 4.3(b). We create pattern *rs*’s childs from the only the items present in the  $N(r)$ . We move to its child node ‘*r,s,p*’ and generate its *TS-list* by performing the intersection of *TS-lists* of ‘*rs*’ and ‘*p*’, i.e.,  $TS^{r,s,p} = TS^{r,s} \cap TS^p$ . We record *support* and *periodicity* of ‘*r,s,p*’, as shown in FPerig. 4.3(c). As the *periodicity* of ‘*r,s,p*’ is greater than the user-specified *maxPer*, we will prune the pattern ‘*r,s,p*’ from the *GFP-list* as shown in Figure 4.3(d). In or-

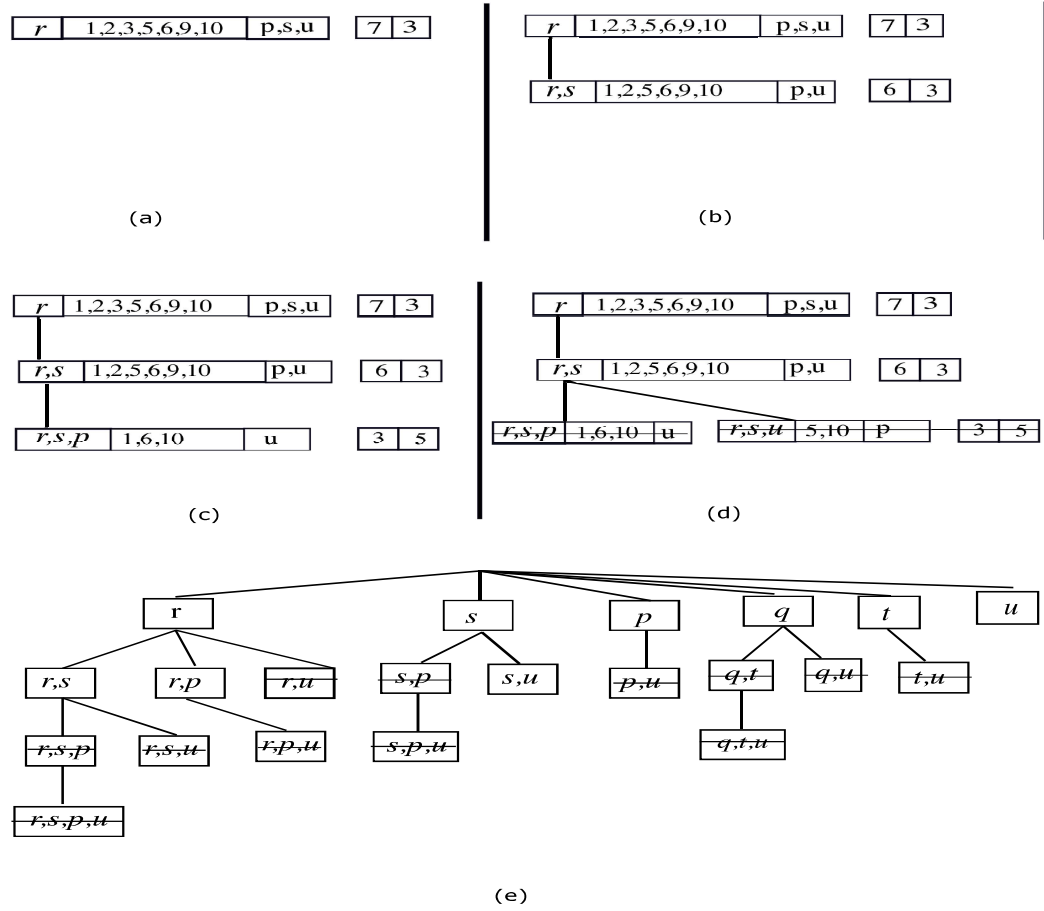


Figure 4.3: Mining of *GPFPS*.

der to discover *GPFPS*, the same procedure is followed for the rest of the nodes in the set-enumeration tree. The complete set of *GPFPS* obtained from Table 4.2 can be shown in Figure 4.3(e). The procedure described above for discovering *GPFPS* using the downward closure property is beneficial because it minimizes the search space and the computational cost. This makes the proposed approach more efficient.

## 4.3 Experimental results

This section will begin by demonstrating that *GPFPS* are an order of magnitude smaller than *PFPS*, particularly when the *minSup* value is low. Next, we demonstrate that *GFPF-Miner* is more effective than the current state-of-the-art *PF-ECLAT* method in terms of memory, shown in *Bytes* (*bytes*), and runtime, shown in *seconds*. Finally, we present two separate case studies to illustrate how beneficial the proposed model is in practice.

### 4.3.1 Experimental setup

Both the *GFPF-Miner* and the *PF-ECLAT* algorithms were programmed in Python 3.7 and ran on a Dell 2U Tower server system that was equipped with 2.30GHz Intel

---

**Algorithm 5** PeriodicFrequentItems(time series database ( $TSD$ ), location database ( $LD$ ),  $minimumsupport(minSup)$ ,  $maximumperiodicity(maxPer)$ , maximum distance ( $maxDist$ ):

---

- 1: Let's say that the GPF-list= $(Y, TS-list(Y))$  is a dictionary that keeps track of temporal information about a pattern that occurs in a TSD. First, let's create a temporary list called  $TS_l$  and use it to keep track of the *timestamp* of the last time an item appeared in the database. Next, let's say that the *per* list is a temporary one that we use to keep track of the database item's periodicity. Finally, let's use another temporary list called *support* to capture the database's *support* for an item.
  - 2: **for** each transaction  $t_{cur} \in TSD$  **do**
  - 3:   Set  $ts_{cur} = t_{cur}.ts$ ;
  - 4:   **for** each item  $j \in t_{cur}.Y$  **do**
  - 5:     **if**  $j$  does not exist in GPF-list **then**
  - 6:       Insert  $j$  and its timestamp into the GPF-list. Set  $TS_l[j] = ts_{cur}$  and  $per[j] = (ts_{cur} - ts_{initial})$ ;
  - 7:     **else**
  - 8:       Add  $j$ 's timestamp to the GPF-list. Update  $TS_l[j] = ts_{cur}$  and  $per[j] = max(per[j], (ts_{cur} - TS_l[j]))$ ;
  - 9:   **for** each item  $j$  in the GPF-list **do**
  - 10:      $support[j] = length(TS-list(j))$
  - 11:     **if**  $support[j] < minSup$  **then**
  - 12:       Prune  $j$  from the GPF-list;
  - 13:     **else**
  - 14:       Calculate  $per[j] = max(per[j], (ts_{final} - TS_l[j]))$ ;
  - 15:       **if**  $per[j] > maxPer$  **then**
  - 16:       Prune  $j$  from the GPF-list.
  - 17: Sort the remaining items in the GPF-list in ascending or descending order of their *support*.
  - 18: Scan the LD and find neighbors for each item available in the GPF-list using the distance function. Let  $N(i_j)$  denote the list of neighboring items for an item  $i_j$ . We call these items as one-length *GFPFs*.
  - 19: We identify the remaining *GFPFs* for each item available in the neighbours list ( $N(i_j)$ ) by Calling GPF-Miner(GPF-list, neighbor list of all the items).
-

**Algorithm 6** GPFMiner(GFPList, neighborList)

---

```

1: for each item  $j$  in GFPList do
2:   Set  $pi = \emptyset$  and  $Y = j$ ;
3:   Let  $neighY = neighborList[j]$ 
4:   for each item  $i$  that comes after  $j$  in the GFPList do
5:     if  $i$  not in  $neighY$  then
6:       continue
7:     Set  $X = Y \cup i$  and  $TS^X = TS^Y \cap TS^i$ ;
8:     if  $sup(TS^X) \geq minSup$  and  $per(TS^X) \leq maxPer$  then
9:       let  $neighX = neighborList[i]$ 
10:       $ne = neighY \cap neighX$ 
11:      Add  $X$  to  $pi$  and  $X$  is considered as GFP. update  $ne$  with their respective
         $X$  to the neighborList;
12:   GFPMiner(pi)

```

---

Xeon Gold 6140 processors. This server computer has 32 gigabytes of RAM and works with CentOS 7. The experiments have been carried out on both synthetic (**T10I4D100K**) and real-world (**Pollution**, and **Congestion**) databases. **The complete evaluation results, databases, and algorithms have been provided through GitHub<sup>1</sup> to verify the repeatability of our experiments.**

### 4.3.2 Evaluation of PF-ECLAT and GPFMiner algorithms by varying $minSup$ constraint

This experiment aims to analyse the performance of the PF-ECLAT and GPFMiner algorithms by varying just the  $minSup$  constraint present in each of the databases. The  $maxPer$  in Pollution, Congestion, and T10I4D100K databases has been set at 0.07(%), 6(%), and 0.1(%), respectively. The  $maxDist$  in Pollution, Congestion, and T10I4D100K databases has been set at 25, 35, and 15, respectively.

Figure 4.4 (a) – Figure 4.4 (c) shows the number of *GFPs* and *PFPs* generated by GPFMiner and PF-ECLAT respectively, over different databases at a distinct  $minSup$  values. The following are some noteworthy findings that may be derived from these figures: (i) The increase in  $minSup$  has decreased the number of *GFPs* and *PFPs*. More importantly, *GFPs* were an order of magnitude smaller than the *PFPs* in any database. (ii) In any database, the total number of *GFPs* generated is relatively smaller than the total number of *PFPs* at a low  $minSup$  value. The GPFMiner pruned all those uninteresting patterns whose items were not neighbors to each other. More importantly, GPFMiner has generated fewer patterns at the low  $minSup$  values.

Figure 4.5 (a) – Figure 4.5 (c) shows the runtime requirements of PF-ECLAT and GPFMiner algorithms over different databases at a distinct  $minSup$  values. It can be observed that GPFMiner is significantly faster than the PF-ECLAT. It is because our algorithm prunes the uninteresting patterns locally to reduce the search space.

Figure 4.6 (a) – Figure 4.6 (c) shows the memory consumption of both PF-ECLAT and GPFMiner algorithms over different databases at a distinct  $minSup$  value. The following are some noteworthy finding that may be derived from these figures: The

---

<sup>1</sup><https://github.com/udayRage/PAMI/tree/main/PAMI/geoReferencedPeriodicFrequentPattern/basic>

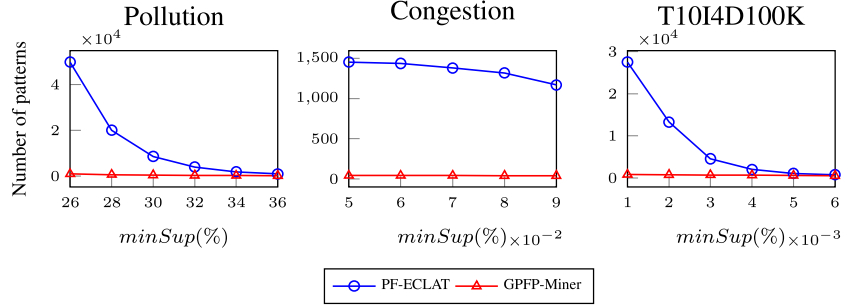


Figure 4.4: Patterns evaluation of PF-ECLAT and GPFM-Miner algorithms at constant  $maxPer$

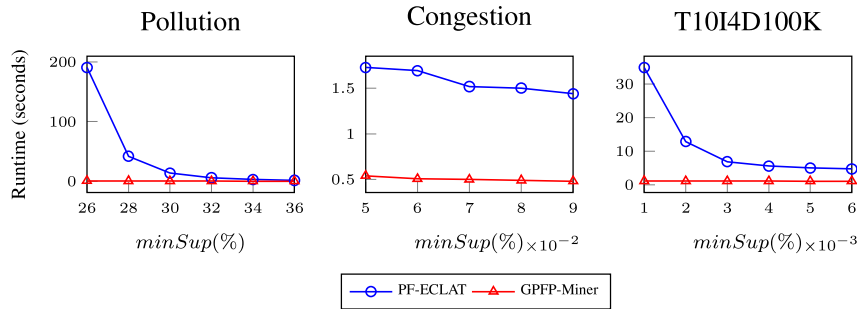


Figure 4.5: Runtime evaluation of PF-ECLAT and GPFM-Miner algorithms at constant  $maxPer$

increase in  $minSup$  has decreased the memory requirements of both PF-ECLAT and GPFM-Miner algorithms. As a result, both algorithms must spend fewer resources to generate fewer  $PFPs$  and  $GFPFs$ .

### 4.3.3 Evaluation of PF-ECLAT and GPFM-Miner algorithms by varying $maxPer$ constraint

This experiment aims to analyse the performance of the PF-ECLAT and GPFM-Miner algorithms by altering just the  $maxPer$  constraint present in each of the databases. The  $minSup$  and  $maxDist$  value in each of the databases will be set to a particular value. The  $minSup$  in Pollution, Congestion, and T10I4D100K databases has been set at 15(%), 5(%), and 10(%), respectively. The  $maxDist$  in Pollution, Congestion, and T10I4D100K databases has been set at 15, 25, and 30, respectively.

Figure 4.7 (a) – Figure 4.7 (c) shows the number of  $GFPFs$  and  $PFPs$  generated by GPFM-Miner and PF-ECLAT respectively, over different databases at distinct  $maxPer$  values. The following are some noteworthy findings that may be derived from these figures: (i) An increase in the  $maxPer$  value has increased the number of  $GFPFs$  and  $PFPs$ . It is because most patterns have become periodic due to an increase in the  $maxPer$  value. (ii) In any database, the total number of  $GFPFs$  generated is relatively smaller than the total number of  $PFPs$  at all  $maxPer$  values. The GPFM-Miner pruned all those uninteresting patterns whose items were not neighbors to each other. More importantly, GPFM-Miner has generated fewer patterns at the higher  $maxPer$  values.

Figure 4.8 (a) – Figure 4.8 (c) shows the runtime requirements of PF-ECLAT and

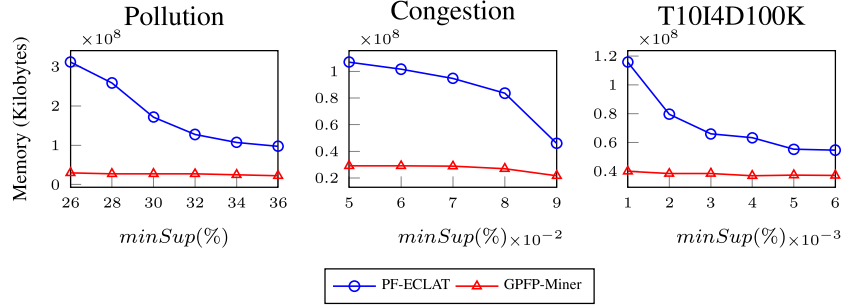


Figure 4.6: Memory evaluation of PF-ECLAT and GPFM-Miner algorithms at constant  $maxPer$

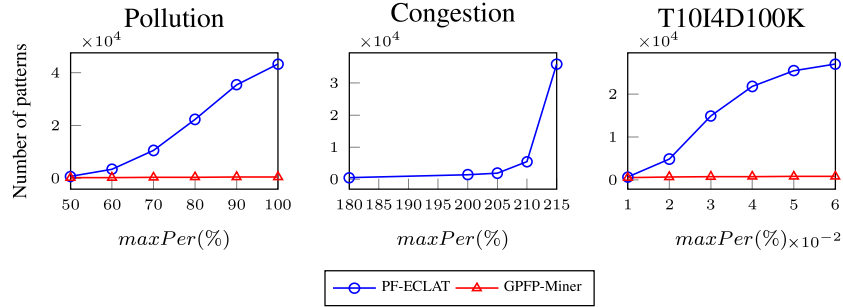


Figure 4.7: Patterns evaluation of PF-ECLAT and GPFM-Miner algorithms at constant  $minSup$

GPFM-Miner algorithms over different databases at distinct  $maxPer$  values. It can be observed that GPFM-Miner is significantly faster than the PFP-Miner. Our algorithm is pruning the uninteresting patterns locally to reduce the search space.

Figure 4.9 (a) – Figure 4.9 (c) shows the memory consumption of both PF-ECLAT and GPFM-Miner algorithms over different databases at a distinct  $maxPer$  value. The following are some noteworthy finding that may be derived from these figures: Increase in  $maxPer$  has increased the memory requirements of both PF-ECLAT and GPFM-Miner algorithms. As a result, both algorithms have to spend more resources to generate more  $PFPs$  and  $GFPs$ .

#### 4.3.4 A case study 1: air pollution analytics

Figure 4.10 shows two patterns colored in green and blue. The parameters used to generate these patterns were  $minSup = 500$ ,  $maxPer = 4111$ , and  $maxDist = 40$  ( $KM$ ). Since PF-ECLAT completely disregards the spatial information of the items, it considered both patterns equally interesting and found them. More importantly, many of the patterns discovered by PF-ECLAT were similar to the blue-colored pattern, where sensors were far apart from each other. In contrast, the proposed GPFM-Miner has taken into account the spatial information of the items and found only the green-colored pattern as an interesting pattern. It can be observed that the green-colored pattern may be found to be more interesting than the blue-colored pattern, as the former represents a concentrated area. Therefore, one might draw the conclusion that those who work or live in the vicinity of green-colored regions have been routinely exposed

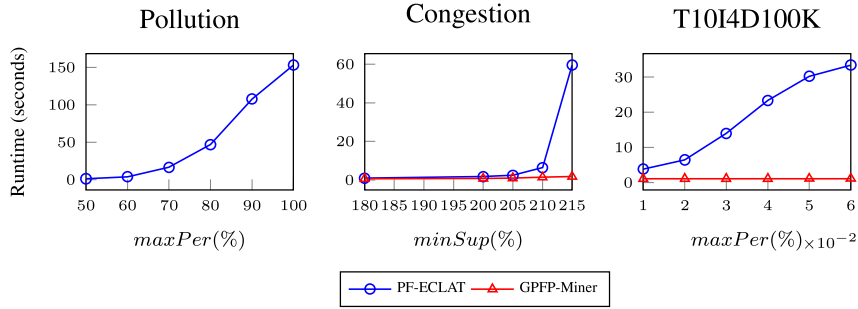


Figure 4.8: Runtime evaluation of PF-ECLAT and GPFM-Miner algorithms at constant minSup

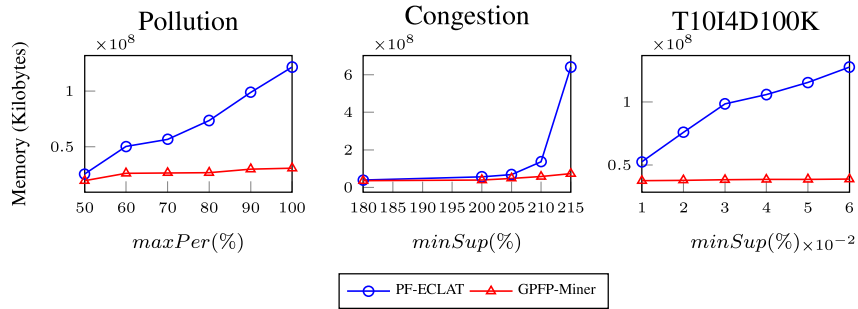


Figure 4.9: Memory evaluation of PF-ECLAT and GPFM-Miner algorithms at constant minSup

to high amounts of  $PM_{2.5}$ . For the sake of formulating policies to reduce pollution and promote public health, such knowledge could prove to be of great use to ecologists.

### 4.3.5 A case study 2: traffic congestion analytics

Figure 4.11 shows two patterns colored in black and red. The parameters used to generate these patterns were  $minSup = 200$ ,  $maxPer = 3211$ , and  $maxDist = 80$  ( $KM$ ). Since PF-ECLAT completely disregards the spatial information of the items, it considered both patterns as equally interesting and found them. More important, many patterns discovered by PF-ECLAT were similar to black pattern, where roads were far apart from each other. In contrast, the proposed GPFM-Miner has taken into account the spatial information of the items and found only the red-colored pattern as an interesting pattern. It can be observed that the red pattern may be found to be more interesting than the black pattern as the former represents a concentrated area. The information that our model gives us can be very helpful when making plans to monitor and diverting the traffic during peak situations.

## 4.4 Discussoin about PAMI package

In this section, to ensure the practical applicability and accessibility of GPFM-Miner, we have developed a dedicated Python package, referred to as PAttern MIning (PAMI). The PAMI serves as a comprehensive Python library designed to facilitate efficient and scalable pattern mining. The package encompasses the implementation of the



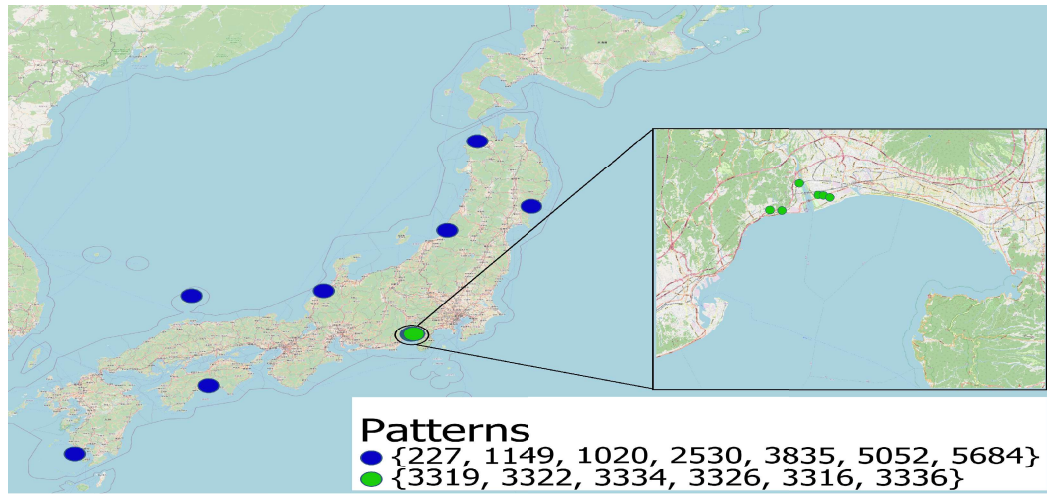


Figure 4.10: Interestingness of our patterns

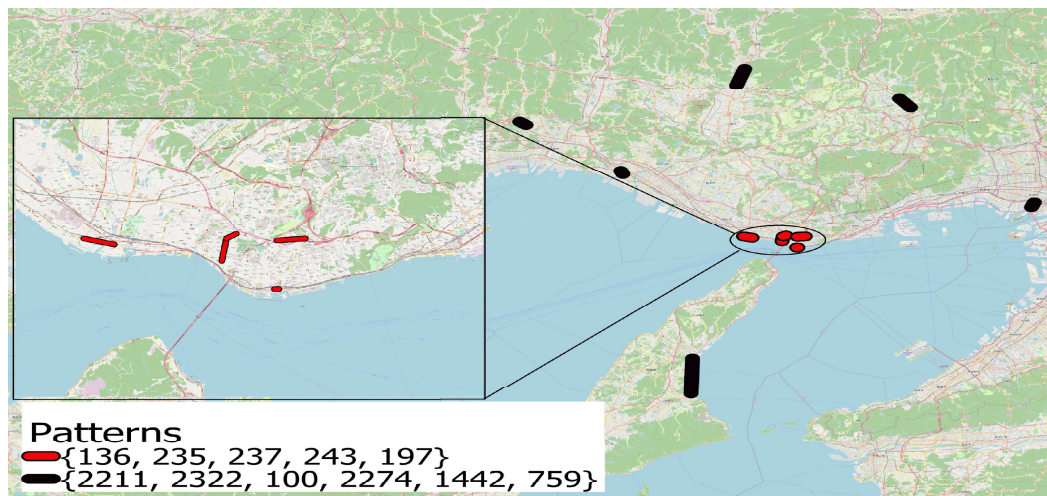


Figure 4.11: Interestingness of our patterns



---

GFPF-Miner algorithm [68], providing researchers and practitioners with a powerful tool for discovering geo-referenced periodic-frequent patterns in large databases. The PAMI package encapsulates key functionalities necessary for effective application of the GFPF-Miner algorithm. These include optimized data structures and algorithms that expedite the mining process, ensuring computational efficiency even for databases of considerable size. Moreover, PAMI offers a user-friendly Github [64] guide that streamlines the integration of GFPF-Miner into existing data mining workflows. Its well-documented comprehensive examples guide users through the installation, configuration, and utilization of the package, ensuring a smooth and intuitive experience.

In addition to the technical aspects, PAMI encourages community engagement and collaboration. The package is open-source and hosted on a publicly accessible repository, facilitating contributions, bug reports, and feature requests from the wider data mining community. This collaborative approach fosters continual improvements, as users can benefit from the expertise and diverse perspectives of fellow researchers and practitioners.

## 4.5 Conclusions

In this chapter, we provide a flexible model of *GFPFs* that could be present in the *GTSD*. In addition, a list-based set enumeration tree methodology that is both effective and efficient has been developed to discover all the needed *GFPFs* inside a *GTSD*. The results of the experiments show that the suggested approach is not only cost-effective in terms of memory and runtime, but that it also excludes a large number of patterns that are not interesting by taking into consideration the spatial information of the items.

# Chapter 5

## Conclusion

### 5.1 Conclusion

This thesis highlights the challenges present in traditional PFP algorithms in RTDB architecture. These challenges include the need for multiple scans of the database and the inability to compute *PFPs* asynchronously. To address these challenges, the thesis proposes a novel and efficient algorithm in Chapter 2 that considers CTDB architecture for discovering interesting patterns. The thesis also notes that periodic pattern mining algorithms can discover both *PFPs* and *3Ps*, each with its own ability to reveal important insights and patterns in the data. Chapter 3 proposes a novel and efficient algorithm for discovering *3Ps* in CTDB architecture. Furthermore, the thesis argues that earlier models overlooked the spatial or geo-referenced information of patterns, which is highly important. To address this issue, Chapter 4 proposes a novel and efficient algorithm for discovering *GPFPs* in CTDB architecture.

Chapter 2 of this thesis introduces an algorithm called PF-ECLAT, which is designed to effectively identify *PFPs* in CTDBs. *PFPs* are patterns that recur regularly over a period of time. The PF-ECLAT algorithm employs two criteria, namely *minSup* and *maxPer*, to eliminate uninteresting patterns. By doing so, it is able to focus on the most relevant patterns and reduce the search space. The algorithm also uses a PFP-List structure to eliminate non-candidate patterns, resulting in decreased memory usage and runtime. To evaluate the effectiveness of the PF-ECLAT algorithm, it was compared to several state-of-the-art algorithms using different real-world and synthetic databases. The experimental results demonstrate that PF-ECLAT outperforms other algorithms in terms of achieving faster and more memory-efficient *PFP* identification. Finally, two case studies are presented to demonstrate the usefulness of PF-ECLAT in air pollution and traffic congestion analytics. In the air pollution case study, the algorithm is used to identify patterns of pollution that recur regularly over time, enabling more effective pollution reduction strategies. In the traffic congestion case study, PF-ECLAT is used to identify regular traffic patterns, which can be used to optimize traffic flow and reduce congestion. Overall, the PF-ECLAT algorithm offers an effective and efficient way to identify *PFPs* in CTDBs, with potential applications in a range of fields, including environmental monitoring and transportation optimization.

Chapter 3 of this thesis introduces the 3P-ECLAT algorithm, which is designed to identify *3Ps* in CTDBs. One of the key challenges in this area is identifying patterns that recur only within specific intervals of time, which is precisely what 3P-ECLAT is designed to address. To evaluate the effectiveness of 3P-ECLAT, we conducted a se-

---

ries of experiments on different real-world and synthetic databases. The results of these experiments demonstrate that 3P-ECLAT outperforms the state-of-the-art 3P-growth algorithm, as it can identify all  $3Ps$  more efficiently and with less memory usage. We also presented a case study to demonstrate the practical usefulness of the discovered patterns in a real-world application. In this case study, we applied 3P-ECLAT to a pollution database of sensor recordings to identify the frequently polluted sensor identifiers during specific time intervals. This information can be used to target pollution reduction efforts more effectively and efficiently. Overall, our study provides a promising new approach to identifying  $3Ps$  in CTDBs, with potential applications in fields such as healthcare, marketing, and environmental monitoring. The 3P-ECLAT algorithm offers a more efficient and effective way to mine these patterns, enabling researchers and practitioners to extract valuable insights from large databases.

Chapter 4 of this thesis presents a new and adaptable model for identifying  $GFPFs$  in a  $STDs$  or  $GTSDs$ . The model is designed to identify patterns based on geographical points (or spatial information) and can be applied in various applications, such as traffic and pollution analytics. This novel model offers a flexible approach to identifying patterns in the data and can be used to gain valuable insights for real-world applications. To efficiently identify all relevant  $GFPFs$  in the database, the authors have devised a list-based set enumeration tree methodology. This technique has been shown to save memory and runtime while also eliminating a significant number of irrelevant patterns by taking into account the spatial information of the items. The approach was tested on various synthetic and real-world databases, and the experimental results show that the proposed technique is effective in terms of memory and runtime while minimizing the number of irrelevant patterns. The authors have demonstrated the usefulness of the proposed model by applying it to traffic congestion and air pollution analytics. The results show that the model provides valuable insights and supports data-driven decision-making.

In summary, this thesis focuses on proposing algorithms for discovering interesting patterns in  $STDs$ , specifically through identifying  $PFPs$ ,  $3Ps$ , and  $GFPFs$ . Chapter 2 introduces the PF-ECLAT algorithm, which identifies  $PFPs$  that occur regularly over a period of time, while Chapter 3 presents the 3P-ECLAT algorithm, which identifies  $3Ps$  that recur within specific intervals of time. Both algorithms do not consider the spatial information of the patterns. However, in Chapter 4, we propose a model for identifying  $GFPFs$  that considers the spatial information of patterns along with frequency and temporal information. Overall, the proposed algorithms and model are evaluated on both real-world and synthetic databases and demonstrate improved memory usage and runtime compared to existing methods. The usefulness of the proposed approaches is demonstrated through case studies in fields such as environmental monitoring and transportation optimization.

## 5.2 Future Research

In recent years,  $STDs$  have grown in size, which means that processing the data in a distributed environment may become necessary to handle the computational load. However, the algorithms proposed in Chapters 2, 3, and 4 are not directly applicable to discovering interesting patterns in a distributed environment. This is because distributed computing requires new algorithms that can process the data in parallel across multiple

machines and strategies for aggregating the results obtained from each machine. As a result, future research could explore the development of extended versions of the algorithms that can process the data in a distributed environment.

Furthermore, although the algorithms presented in Chapters 2, 3, and 4 offer an efficient way of discovering *PFPs*, *3Ps*, and *GPFPs*, there may be further opportunities to reduce the computational cost of mining these patterns. For example, future research could investigate novel measures or techniques that reduce the search space, optimize the algorithms themselves to further reduce computational costs, or develop new data structures that are more efficient at storing and retrieving patterns.

Lastly, Chapter 4 focuses on the challenge of discovering *GPFPs* using static and definite data. However, in real-world scenarios, data can be dynamic and uncertain. Future research could explore the adaptation of the proposed algorithm to data streams and uncertain databases by developing algorithms that update patterns dynamically and account for uncertainty. This would involve developing algorithms that can quantitatively measure and account for uncertainty in the patterns and dynamically update the patterns as new data becomes available.

# References

- [1] D. Abadi, S. Madden, and M. Ferreira, “Integrating compression and execution in column-oriented database systems,” in *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD ’06. New York, NY, USA: Association for Computing Machinery, 2006, p. 671–682. [Online]. Available: <https://doi.org/10.1145/1142473.1142548>
- [2] D. Abadi, P. Boncz, and S. Harizopoulos, *The Design and Implementation of Modern Column-Oriented Database Systems*. Hanover, MA, USA: Now Publishers Inc., 2013.
- [3] MySQL, “Mysql,” <https://www.mysql.com/>.
- [4] Postgres, “Postgres,” <https://www.postgresql.org/>.
- [5] Snowflake, “Snowflake,” <https://www.snowflake.com/>.
- [6] BigQuery, “Bigquery,” <https://cloud.google.com/bigquery>.
- [7] A. K. Jain, M. N. Murty, and P. J. Flynn, “Data clustering: A review,” *ACM Comput. Surv.*, vol. 31, no. 3, p. 264–323, sep 1999. [Online]. Available: <https://doi.org/10.1145/331499.331504>
- [8] V. Chandola, A. Banerjee, and V. Kumar, “Anomaly detection: A survey,” *ACM Comput. Surv.*, vol. 41, no. 3, jul 2009. [Online]. Available: <https://doi.org/10.1145/1541880.1541882>
- [9] P. Domingos, “A few useful things to know about machine learning,” *Commun. ACM*, vol. 55, no. 10, p. 78–87, oct 2012. [Online]. Available: <https://doi.org/10.1145/2347736.2347755>
- [10] R. Agrawal, T. Imieliński, and A. Swami, “Mining association rules between sets of items in large databases,” in *International Conference on Management of Data*, 1993, pp. 207–216.
- [11] C. C. Aggarwal, *Applications of Frequent Pattern Mining*. Springer International Publishing, 2014, pp. 443–467.
- [12] R. Agrawal, T. Imieliński, and A. Swami, “Mining association rules between sets of items in large databases,” in *SIGMOD*, 1993, pp. 207–216.
- [13] J. M. Luna, P. Fournier-Viger, and S. Ventura, “Frequent itemset mining: A 25 years review,” *Wiley Interdiscip. Rev. Data Min. Knowl. Discov.*, vol. 9, no. 6, 2019.

- [14] J. Han, J. Pei, and Y. Yin, “Mining frequent patterns without candidate generation,” *SIGMOD Rec.*, vol. 29, no. 2, p. 1–12, may 2000. [Online]. Available: <https://doi.org/10.1145/335191.335372>
- [15] M. J. Zaki, “Scalable algorithms for association mining,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 12, no. 3, pp. 372–390, 2000.
- [16] P. Fournier-Viger, J. C.-W. Lin, R. U. Kiran, and Y. S. Koh, “A survey of sequential pattern mining,” *Data Science and Pattern Recognition*, vol. 1, no. 1, pp. 54–77, 2017.
- [17] C.-K. Chui, B. Kao, and E. Hung, “Mining frequent itemsets from uncertain data,” in *Advances in Knowledge Discovery and Data Mining*, Z.-H. Zhou, H. Li, and Q. Yang, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 47–58.
- [18] X. Yan and J. Han, “gspan: graph-based substructure pattern mining,” in *2002 IEEE International Conference on Data Mining, 2002. Proceedings.*, 2002, pp. 721–724.
- [19] J. H. Chang and W. S. Lee, “Finding recent frequent itemsets adaptively over online data streams,” in *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '03. New York, NY, USA: Association for Computing Machinery, 2003, p. 487–492. [Online]. Available: <https://doi.org/10.1145/956750.956807>
- [20] S. K. Tanbeer, C. F. Ahmed, B.-S. Jeong, and Y.-K. Lee, “Discovering periodic-frequent patterns in transactional databases,” in *PAKDD*, 2009, pp. 242–253.
- [21] K. Amphawan, P. Lenca, and A. Surarerks, “Mining top-k periodic-frequent pattern from transactional databases without support threshold,” in *Advances in Information Technology*, 2009, pp. 18–29.
- [22] R. U. Kiran, M. Kitsuregawa, and P. K. Reddy, “Efficient discovery of periodic-frequent patterns in very large databases,” *Journal of Systems and Software*, vol. 112, pp. 110–121, 2016. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0164121215002307>
- [23] R. U. Kiran and M. Kitsuregawa, “Novel techniques to reduce search space in periodic-frequent pattern mining,” in *Database Systems for Advanced Applications*, S. S. Bhowmick, C. E. Dyreson, C. S. Jensen, M. L. Lee, A. Muliantara, and B. Thalheim, Eds. Cham: Springer International Publishing, 2014, pp. 377–391.
- [24] R. U. Kiran, M. Kitsuregawa, and P. K. Reddy, “Efficient discovery of periodic-frequent patterns in very large databases,” *J. Syst. Softw.*, vol. 112, pp. 110–121, 2016.
- [25] O. Database, “Oracle spatial and graph,” <http://www.oracle.com/technetwork/database-options/spatialandgraph/overview/index.html>.
- [26] PostGres, “Postgis,” <https://postgis.net/>.
- [27] O. S. G. Foundation, “Geoserver,” <http://geoserver.org/>.

- 
- [28] A. Anirudh, R. U. Kiran, P. K. Reddy, and M. Kitsuregawa, "Memory efficient mining of periodic-frequent patterns in transactional databases," in *2016 IEEE Symposium Series on Computational Intelligence, SSCI 2016, Athens, Greece, December 6-9, 2016*. IEEE, 2016, pp. 1–8.
- [29] R. Penugonda, L. Palla, U. K. Rage, Y. Watanobe, and K. Zettsu, "Towards efficient discovery of periodic-frequent patterns in columnar temporal databases," in *Advances and Trends in Artificial Intelligence. Artificial Intelligence Practices*, H. Fujita, A. Selamat, J. C.-W. Lin, and M. Ali, Eds. Cham: Springer International Publishing, 2021, pp. 28–40.
- [30] P. Ravikumar, P. Likhitha, B. Venus Vikranth Raj, R. Uday Kiran, Y. Watanobe, and K. Zettsu, "Efficient discovery of periodic-frequent patterns in columnar temporal databases," *Electronics*, vol. 10, no. 12, 2021.
- [31] P. Ravikumar, V. V. Raj, P. Likhitha, R. U. Kiran, Y. Watanobe, S. Ito, K. Zettsu, and M. Toyoda, "Towards efficient discovery of partial periodic patterns in columnar temporal databases," in *Intelligent Information and Database Systems*, N. T. Nguyen, T. K. Tran, U. Tukayev, T.-P. Hong, B. Trawiński, and E. Szczerbicki, Eds. Cham: Springer Nature Switzerland, 2022, pp. 141–154.
- [32] P. Ravikumar, R. U. Kiran, P. Likhitha, T. Chandrasekhar, Y. Watanobe, and K. Zettsu, "Discovering geo-referenced periodic-frequent patterns in geo-referenced time series databases," in *2022 IEEE 9th International Conference on Data Science and Advanced Analytics (DSAA)*, 2022, pp. 1–10.
- [33] R. U. Kiran and P. K. Reddy, "Towards efficient mining of periodic-frequent patterns in transactional databases," in *International Conference on Database and Expert Systems Applications*. Springer, 2010, pp. 194–208.
- [34] K. Amphawan, A. Surarerks, and P. Lenca, "Mining periodic-frequent itemsets with approximate periodicity using interval transaction-ids list tree," in *2010 Third International Conference on Knowledge Discovery and Data Mining*, 2010, pp. 245–248.
- [35] R. U. Kiran and P. K. Reddy, "An alternative interestingness measure for mining periodic-frequent patterns," in *DASFAA (1)*, 2011, pp. 183–192.
- [36] M. M. Rashid, M. R. Karim, B. S. Jeong, and H. J. Choi, "Efficient mining regularly frequent patterns in transactional databases," in *International Conference on Database Systems for Advanced Applications (1)*, 2012, pp. 258–271.
- [37] A. Anirudh, R. U. Kiran, P. K. Reddy, and M. Kitsuregawa, "Memory efficient mining of periodic-frequent patterns in transactional databases," in *2016 IEEE Symposium Series on Computational Intelligence*, 2016, pp. 1–8.
- [38] J. N. Venkatesh, R. U. Kiran, P. K. Reddy, and M. Kitsuregawa, "Discovering periodic-frequent patterns in transactional databases using all-confidence and periodic-all-confidence," in *Database and Expert Systems Applications - 27th International Conference, DEXA 2016, Porto, Portugal, September 5-8, 2016, Proceedings, Part I*, 2016, pp. 55–70.
-

- [39] J. Han, W. Gong, and Y. Yin, "Mining segment-wise periodic patterns in time-related databases." in *KDD*, vol. 98, 1998, pp. 214–218.
- [40] J. Han, G. Dong, and Y. Yin, "Efficient mining of partial periodic patterns in time series database," in *Proceedings 15th International Conference on Data Engineering (Cat. No. 99CB36337)*. IEEE, 1999, pp. 106–115.
- [41] F. Rasheed and R. Alhaji, "Stnr: A suffix tree based noise resilient algorithm for periodicity detection in time series databases," *Applied Intelligence*, vol. 32, no. 3, pp. 267–278, Jun 2010. [Online]. Available: <https://doi.org/10.1007/s10489-008-0144-9>
- [42] R. Yang, W. Wang, and P. S. Yu, "Infominer+: mining partial periodic patterns with gap penalties," in *2002 IEEE International Conference on Data Mining, 2002. Proceedings.* IEEE, 2002, pp. 725–728.
- [43] J. Yang, W. Wang, and P. S. Yu, "Infominer: Mining surprising periodic patterns," in *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '01. New York, NY, USA: Association for Computing Machinery, 2001, p. 395–400. [Online]. Available: <https://doi.org/10.1145/502512.502571>
- [44] C. Berberidis, W. G. Aref, M. Atallah, I. Vlahavas, and A. K. Elmagarmid, "Multiple and partial periodicity mining in time series databases," in *Proceedings of the 15th European Conference on Artificial Intelligence*, ser. ECAI'02. NLD: IOS Press, 2002, p. 370–374.
- [45] B. Özden, S. Ramaswamy, and A. Silberschatz, "Cyclic association rules," in *Proceedings of the Fourteenth International Conference on Data Engineering*, ser. ICDE '98. USA: IEEE Computer Society, 1998, p. 412–421.
- [46] R. U. Kiran, J. N. Venkatesh, M. Toyoda, M. Kitsuregawa, and P. K. Reddy, "Discovering partial periodic-frequent patterns in a transactional database," *J. Syst. Softw.*, vol. 125, pp. 170–182, 2017.
- [47] S. Nakamura, R. U. Kiran, P. Likhitha, P. Ravikumar, Y. Watanobe, M. S. Dao, K. Zettsu, and M. Toyoda, "Efficient discovery of partial periodic-frequent patterns in temporal databases," in *Database and Expert Systems Applications*, C. Strauss, G. Kotsis, A. M. Tjoa, and I. Khalil, Eds. Cham: Springer International Publishing, 2021, pp. 221–227.
- [48] R. U. Kiran, H. Shang, M. Toyoda, and M. Kitsuregawa, "Discovering partial periodic itemsets in temporal databases," in *Proceedings of the 29th International Conference on Scientific and Statistical Database Management*, ser. SSDBM '17, 2017.
- [49] R. Agrawal, R. Srikant *et al.*, "Fast algorithms for mining association rules," in *Proc. 20th int. conf. very large data bases*, vol. 1215, 1994, pp. 487–499.
- [50] W. Ding, C. F. Eick, J. Wang, and X. Yuan, "A framework for regional association rule mining in spatial datasets," in *Proceedings of the Sixth International Conference on Data Mining*, ser. ICDM '06, 2006, pp. 851–856.



- 
- [51] C. F. Eick, R. Parmar, W. Ding, T. F. Stepinski, and J.-P. Nicot, "Finding regional co-location patterns for sets of continuous variables in spatial datasets," in *Proceedings of the 16th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, ser. GIS '08, 2008, pp. 30:1–30:10.
- [52] P. Mohan, S. Shekhar, J. A. Shine, J. P. Rogers, Z. Jiang, and N. Wayant, "A neighborhood graph based approach to regional co-location pattern discovery: A summary of results," in *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, ser. GIS '11, 2011, pp. 122–132.
- [53] H. Tran-The and K. Zetsu, "Discovering co-occurrence patterns of heterogeneous events from unevenly-distributed spatiotemporal data," in *2017 IEEE International Conference on Big Data, BigData 2017, Boston, MA, USA, December 11-14, 2017*, 2017, pp. 1006–1011.
- [54] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, ser. KDD'96. AAAI Press, 1996, p. 226–231.
- [55] R. U. Kiran, S. Shrivastava, P. Fournier-Viger, K. Zetsu, M. Toyoda, and M. Kitsuregawa, "Discovering frequent spatial patterns in very large spatiotemporal databases," in *SIGSPATIAL '20: 28th International Conference on Advances in Geographic Information Systems, Seattle, WA, USA, November 3-6, 2020*, C. Lu, F. Wang, G. Trajcevski, Y. Huang, S. D. Newsam, and L. Xiong, Eds. ACM, 2020, pp. 445–448.
- [56] M. Y. Ansari, A. Ahmad, S. S. Khan, G. Bhushan, and Mainuddin, "Spatiotemporal clustering: a review," *Artificial Intelligence Review*, vol. 53, no. 4, pp. 2381–2423, 2020.
- [57] R. U. Kiran, "PAttern MIning-Python Kit (PAMI-PyKit)," [https://github.com/udayRage/pami\\_pykit/tree/master/traditional/Eclat-pfp](https://github.com/udayRage/pami_pykit/tree/master/traditional/Eclat-pfp), 2020, [Online; accessed 4-June-2020].
- [58] P. Fournier-Viger, "Spmf: A java open-source data mining library," <http://www.philippe-fournier-viger.com/spmf/index.php?link=datasets.php>, 2020, [Online; accessed 4-June-2020].
- [59] National Center for Atmospheric Research, University Corporation for Atmospheric Research, "Standardized precipitation index (spi) for global land surface (1949-2012)," Boulder CO, 2013.
- [60] JARTIC, "Japan Road Traffic Information Center," <https://www.jartic.or.jp>, 2020, [Online; accessed 11-November-2020].
- [61] T. J. Times, "Air pollution deaths in japan," <https://www.japantimes.co.jp/life/2019/05/11/environment/reading-air-tokyo-still-work-air-pollution>, 2019, [Online; accessed 12-December-2020].
-

- [62] J. The Ministry of Environment, “Soramame,” <http://soramame.taiki.go.jp/>, [Online; accessed 12-December-2020].
- [63] R. U. Kiran, “PAttern MIning (PAMI),” <https://github.com/udayRage/PAMI/blob/main/PAMI/periodicFrequentPattern/basic/PFECLAT.py>, 2023, [Online; accessed 17-April-2023].
- [64] —, “PAttern MIning (PAMI),” <https://github.com/udayRage/PAMI/tree/main/PAMI>, 2023, [Online; accessed 17-April-2023].
- [65] R. U. Kiran, P. Veena, P. Ravikumar, C. Saideep, K. Zettsu, H. Shang, M. Toyoda, M. Kitsuregawa, and P. K. Reddy, “Efficient discovery of partial periodic patterns in large temporal databases,” *Electronics*, vol. 11, no. 10, 2022. [Online]. Available: <https://www.mdpi.com/2079-9292/11/10/1523>
- [66] R. U. Kiran, “PAttern MIning (PAMI),” [https://github.com/udayRage/PAMI/blob/main/PAMI/partialPeriodicPattern/basic/PPP\\_ECLAT.py](https://github.com/udayRage/PAMI/blob/main/PAMI/partialPeriodicPattern/basic/PPP_ECLAT.py), 2023, [Online; accessed 17-April-2023].
- [67] “World air quality index team,” <http://aqicn.org/here/>, [Online; accessed 16-August-2019].
- [68] R. U. Kiran, “PAttern MIning (PAMI),” <https://github.com/udayRage/PAMI/blob/main/PAMI/geoReferencedPeriodicFrequentPattern/basic/GPFPMiner.py>, 2023, [Online; accessed 17-April-2023].