Improved Approaches to Mine Periodic-Frequent Patterns in Non-Uniform Temporal Databases

Thesis submitted in partial fulfillment
of the requirements for the degree of

Master of Science
in
Computer Science and Engineering
by Research

by

J N Venkatesh
201102007
jn.venkatesh@research.iiit.ac.in

International Institute of Information Technology, Hyderabad
(Deemed to be University)
Hyderabad - 500 032, INDIA
March 2018

International Institute of Information Technology
Hyderabad, India

CERTIFICATE

It is certified that the work contained in this thesis, titled "Improved Approaches to Mine Periodic-Frequent Patterns in Non-Uniform Temporal Databases" by J N Venkatesh, has been carried out under my supervision and is not submitted elsewhere for a degree.

Date

Adviser: Prof. P. Krishna Reddy

To my Family and Friends

Acknowledgments

This thesis is a result of inputs and contribution from various people. I owe my deepest gratitude to my guide, Prof. P. Krishna Reddy, for his guidance, suggestions and constructive criticism throughout the duration of this work. He helped in shaping the direction of this work, filled in many of the gaps in my knowledge, and helped me towards solutions. Without his affectionate guidance and help, I may not have completed this thesis work. I thank him for the enormous patience that he has shown towards me.

I express my gratitude to Dr. R Uday Kiran from University of Tokyo who has put a great effort in providing me guidance and shaping my research. The diverse knowledge he had in various areas helped me in learning many concepts. He helped me a lot in improving my technical writing skills. He has supported me throughout my thesis with his patience and knowledge. I am grateful to him for a ton of productive discussions we had throughout this thesis.

I am also grateful to Kumara Swamy, Anirudh, Mamatha, Narendra, Kavya, Raghav, and Amar with whom I have spent many hours discussing and working with. I thank them for helping me write research papers. Their company has contributed greatly in making this thesis fun working on. I would also like to thank my friends who were always with me in my ups and downs especially during this period of two years.

I would also like to thank the most important people in my life, my family. I cannot thank my parents enough for all the love, support and sacrifices they have made for me throughout my life. I am also thankful to my brother and the rest of my family for always believing in me. Without the endless love, guidance and support of my family, this work would not have been possible.

I am deeply obliged to ACM India-IARCS for providing financial assistance to attend and present my research work at PAKDD-2017 in Jeju, South Korea.

Finally, I would like to thank the almighty God who gave me all the help, knowledge and courage to complete my work.

Abstract

The field of data mining has emerged to extract knowledge hidden in large databases for better decision making. The process of frequent pattern (a set of items represents a pattern (or an itemset)) mining finds interesting information about the association among the items in a transactional database. The notion of $support$ is employed to extract the frequent patterns. A pattern is called a frequent pattern if it satisfies the user-defined threshold on minimum $support$. An important criterion to assess the interestingness of a frequent pattern is its temporal occurrences in a database. That is, whether a frequent pattern is occurring periodically, irregularly, or mostly at specific time intervals in a database. The class of frequent patterns that are occurring periodically within a database are known as periodic-frequent patterns. Finding these patterns is a significant task with many real-world applications like improving the performance of recommender systems, intrusion detection in computer networks, discovering events in Twitter.

Current periodic-frequent pattern models cannot handle datasets in which multiple transactions share a common timestamp or when transactions occur at irregular time intervals. This issue limits the applicability of the model as in many real-world databases like e-Commerce, Twitter, etc., transactions share a common timestamp and uneven time gaps exist in between the consecutive transactions. Most previous models on periodic-frequent pattern mining have focused on finding all patterns in a transactional database that satisfy the user-specified minimum support ($minSup$) and maximum periodicity ($maxPer$) constraints. The $minSup$ constraint controls the minimum number of transactions that a pattern must cover in a database. The $maxPer$ constraint controls the maximum duration between the two transactions below which a pattern should reoccur in a database. The usage of a single $minSup$ and $maxPer$ for an entire database leads to the *rare item problem*, because real-world databases have a non-uniform item distribution, which considers that items have different $support$ and $periodicity$ values. Also, current periodic-frequent pattern models have focused on discovering full periodic-frequent patterns, i.e., finding all patterns that have exhibited complete cyclic repetitions throughout the entire database. These models evaluate the periodic interestingness of a frequent pattern by determining whether all of its inter-arrival times are within the user-specified $maxPer$ threshold. Therefore, the model cannot assess the partial periodic behavior of a frequent pattern in a database. However, partial periodic-frequent patterns are more common due to the imperfect nature of real-world databases.

So, to address the above issues, in this thesis, we are proposing two improved approaches that discover periodic-correlated patterns and partial periodic-frequent patterns in non-uniform temporal

databases, respectively. In the first approach, we tackle *rare item problem* by proposing a improved model that discovers periodic-correlated patterns in a non-uniform temporal database. In this thesis, we consider temporal database as a collection of transactions, ordered by their timestamps. Further, a temporal database facilitates multiple transactions to share a common timestamp and allows time-gaps in between consecutive transactions. A temporal database is said to be non-uniform if it contains items with dissimilar *support* and *periodicity*. To tackle *rare item problem* in non-uniform temporal databases, the proposed model considers a pattern as interesting if its $support$ and $periodicity$ are close to that of its individual items. The existing *all-confidence* measure is used to determine how close is the $support$ of a pattern with respect to the $support$ of its individual items. A new interestingness measure, called *periodic-all-confidence*, is being proposed to determine how close is the $periodicty$ of a pattern with respect to the $periodicity$ of its individual items. A pattern-growth algorithm has also been discussed to find periodic-correlated patterns. Experimental results show that the proposed model is efficient and tackles *rare item problem*. We discuss the usefulness of periodic-correlated patterns with a real-world case study on FAA-Accidents database and show that the proposed model may be utilized to discover interesting periodic-correlated patterns involving both frequent and rare items effectively.

In the second approach, we have introduced a improved model to discover partial periodic-frequent patterns in non-uniform temporal databases. The proposed model lets the user specify a different *maximum inter-arrival time* ($MIAT$) for each item. An inter-arrival time of a pattern is considered periodic (or cyclic) if it is no more than $period$. Thus, different patterns may satisfy different $period$ depending on their items' $MIAT$ values. This solves the *rare item problem* in partial periodic-frequent pattern mining. A new measure, *Relative Periodic-Support* (RPS), is proposed to determine the (partial) periodic interestingness of a pattern by considering the number of cyclic repetitions in the database. This measure assess the interestingness of a pattern by taking into account both the $support$ and $periodicity$ information of patterns. A pattern-growth algorithm has been discussed to discover all partial periodic-frequent patterns. Experimental results demonstrate that the proposed model is efficient and tackles the problem of *rare item problem* as well as the problem of discovering partial periodic-frequent patterns. We discuss the usefulness of partial periodic-frequent patterns with a real-world case study and show that the proposed model may be utilized to find prior knowledge about event keywords and their associations in Twitter data.

Overall, in this thesis, we have proposed improved approaches to extract periodic-frequent patterns in non-uniform temporal databases and have shown the advantages through experimental results.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

We are in the era of data explosion. With the computers and technology becoming cheaper and more powerful, enormous amount of data is being stored in files, databases and other repositories. As a result, it is becoming difficult for humans to take effective decisions from the huge amount of raw data. The field of data mining has emerged to extract information/knowledge hidden in the voluminous data. Data mining is defined as the nontrivial extraction of implicit, previously unknown, and potentially useful information from data. Data mining transforms data into business intelligence and has become an important part of the modern organizations. The data mining is being used in wide range of industry applications, such as marketing, surveillance, fraud detection, customer relationship management, bioinformatics, etc.

The process of data mining mainly include frequent pattern mining, association rules mining, classification, regression, and clustering. Over last two decades, researchers have made significant efforts to propose data mining algorithms for above techniques. Currently, data mining is a very active research area of computer science and research efforts are being made to discover knowledge patterns from data streams [20, 29, 59], time-series data [16], probabilistic databases [22, 46], high-dimensional data [36, 40], clustering [9, 40], and classification [12, 48].

1.1 Frequent Patterns

Frequent patterns (or itemsets) are an important class of regularities that exist in a transactional database. These patterns play a key role in many knowledge discovery tasks such as association rule mining [3, 23], clustering [66], classification [17] and recommender systems [1, 2]. The process of frequent pattern extraction finds interesting information about the association among the items in a transactional database which co-occur frequently. An example of a frequent pattern is as follows:

$$\{Bat, Ball\} \quad [support = 10\%].$$

The above pattern says that 10% of the purchases have the items 'Bat' and '$Ball$,' together. This kind of information will help e-Commerce/retailer in product recommendation and inventory management.

Figure 1.1 shows the application of frequent patterns on Flipkart website by recommending the users that the items Concepts of Physics by H.C. Verma - Vol.1 & Vol.2 and Objective Mathematics for JEE (Main & Advanced) are bought together frequently by the customers. Due to its usefulness in decision making process, research is also going on to investigate efficient approaches to extract the patterns pertaining to rarely occurring objects besides discovering patterns pertaining to frequent objects.

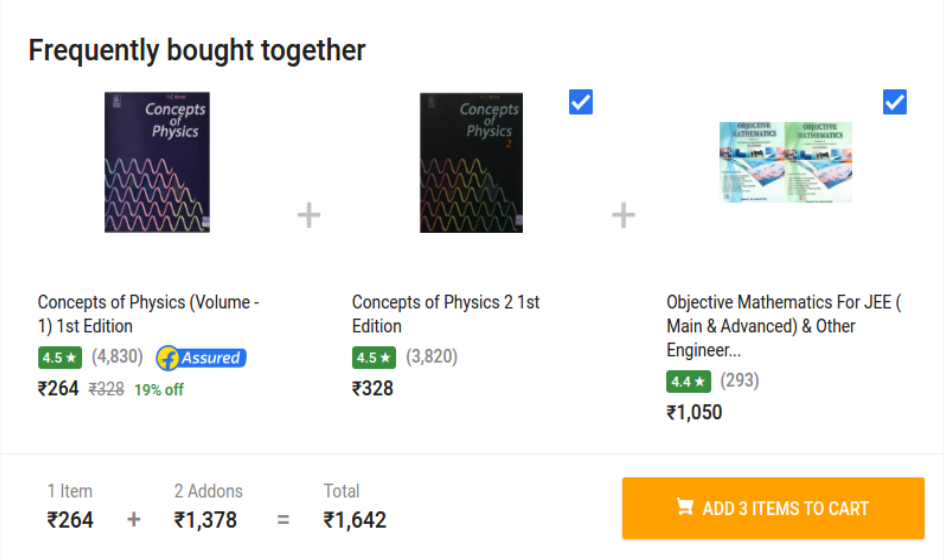


Figure 1.1 Application of Frequent patterns in e-Commerce @ Flipkart

However, frequent pattern mining often generates a huge number of patterns and majority of them may be found insignificant depending on application or user requirements. To solve this problem, researchers have tried to reduce the desired set by finding user interest-based patterns such as maximal frequent patterns [21], demand driven patterns, utility patterns [63], constraint-based patterns [43], diverse-frequent patterns [52], top-k patterns [27] and periodic-frequent patterns [54].

In this thesis, we focus on periodic-frequent patterns and propose two innovative approaches to solve important problems existing in periodic-frequent pattern mining.

1.2 Periodic-Frequent Patterns

An important criterion to assess the interestingness of a frequent pattern is its temporal occurrences in a database. That is, whether a frequent pattern is occurring periodically, irregularly, or mostly at specific time intervals in a database. The class of frequent patterns that are occurring periodically within a database are known as periodic-frequent patterns. A classic application to illustrate the usefulness of these patterns is market-basket analysis. It analyzes how regularly the set of items are being purchased. An example of a periodic-frequent pattern is as follows:

$$\{Bat, Ball\} \quad [support = 10\%, \ \ periodicity = 1 \ hour].$$

The above pattern says that 10% of the purchases have the items 'Bat' and '$Ball$', and the maximum duration between any two consecutive purchases containing both of these items is no more than an hour. Given the extra information of *periodicity* in addition to *support*, this predictive behavior of the customers' purchases may better enable the e-Commerce/retailer to do inventory management or facilitate the users in product recommendation.

Periodic-frequent pattern[1] mining is an important model in data mining. Tanbeer et al. [54] introduced this model which involves discovering all patterns in a transactional database that satisfy the user-specified minimum support ($minSup$) and maximum periodicity ($maxPer$) constraints. The $minSup$ controls the minimum number of transactions that a pattern must cover, and the $maxPer$ controls the maximum interval within which a pattern must reoccur in the entire database. With this model, one can ask queries like, for example in the domain of e-Commerce, extract all patterns (or itemsets) which appear in at-least 10% of the purchases and are purchased at least once in every hour.

Finding periodic-frequent patterns is a significant task with many real-world applications. Examples include improving the performance of recommender systems [49], intrusion detection in computer networks [39] and finding co-occurring genes in biological data sets [65]. In stock market trading, the set of high stocks indices that rise periodically may be of special interest to companies. In a retail market, among all frequently sold products, the user may be interested only in the regularly sold products compared to the rest. For improved web site design or web administration, an administrator may be interested on the click sequences of heavily hit web pages. In genetic data analysis [65], the set of all genes that not only appear frequently but also co-occur at regular interval in DNA sequence may carry more significant information to the scientists.

The problem of finding periodic-frequent patterns has been widely studied in the past [5, 6, 7, 19, 32, 45]. Amphawan et al. [5] extended the model to find top-k periodic-frequent patterns in a transactional database. Anirudh et al. [6, 7] have proposed various techniques like *period-summary* and parallelization to improve the memory and runtime performance of periodic-frequent pattern mining. Fournier-Viger et al. [19] extended the periodic-frequent pattern model to find periodic-utility patterns in a transactional database. Uday et al. [32] proposed an optimization technique based on greedy search technique to performance of periodic-frequent pattern mining. Rashid et al. [45] employed standard deviation of periods as a criterion to assess the periodic behavior of frequent patterns.

The popular adoption and successful industrial application of this basic model used in all these studies, suffers from various issues. In the next subsection, we discuss about these issues.

1.3 Issues with Existing Approaches

The popular adoption and successful industrial application of this basic model used in all the above studies suffers from the following issues:

[1] A set of items represents a pattern (or an itemset)

- Since the model accepts *transactional database* as an input, the model implicitly assumes that all transactions in a database occur at a fixed time interval. This assumption limits the applicability of the model as transactions in many real-world databases occur at irregular time intervals. In particular, multiple transactions can share a common timestamp and uneven time gaps can exist in between the consecutive transactions.

- Since only a single $minSup$ and $maxPer$ are used for the whole data, the model implicitly assumes that all items in the data have uniform $support$ and $periodicity$. However, this is seldom the case in many real-world applications. In many applications, some items appear very frequently in the data, while others rarely[2] appear. Moreover, rare items typically have high $periodicity$ (i.e., inter-arrival times) as compared against the frequent items. Hence, if the $support$ and $periodicity$ of items vary a great deal, we will encounter the following two problems:

  - If the $maxPer$ is set too low and/or the $minSup$ is set too high, we will miss the periodic patterns involving rare items.
  - To find the periodic patterns involving both frequent and rare items, we have to set a high $maxPer$ and a low $minSup$. However, this may result in combinatorial explosion, producing too many patterns, because frequent items can combine with one another in all possible ways and many of them may be meaningless.

  This dilemma is known as the "*rare item problem*" [57].

- The basic model evaluates the periodic interestingness of a frequent pattern by determining whether all of its inter-arrival times are within the user-specified $maxPer$ threshold. Therefore, the model cannot assess the partial periodic behavior of a frequent pattern in a database. However, partial periodic-frequent patterns are more common due to the imperfect nature of real-world databases. Discovering partial periodic-frequent patterns in temporal databases has numerous applications.

1.4 Overview of the Proposed Approaches

In this section, we give a brief overview of how we tackle the issues discussed in previous section. We are proposing the following two approaches to tackle the issues discussed.

- Discovering Periodic-Correlated Patterns in Non-Uniform Temporal Databases.

- Discovering Partial Periodic-Frequent Patterns in Non-Uniform Temporal Databases.

[2]Classifying the items into either frequent or rare is a subjective issue that depends upon the user and/or application requirements.

1.4.1 Discovering Periodic-Correlated Patterns in Non-Uniform Temporal Databases

A temporal database is a collection of transactions, ordered by their timestamps. In temporal databases, time gaps can exist in between consecutive transactions and there can be multiple transactions with the same timestamp. A temporal database is said to be non-uniform if it contains items with dissimilar $support$ and $periodicity$. Non-uniform temporal data is naturally produced in many real-world situations. For instance, disasters such as earthquakes and tsunami happen at irregular time intervals. Twitter data related to these disasters is thus non-uniform.

In the first proposed approach, we propose a model to discover periodic-correlated patterns in a non-uniform temporal database. In the literature, correlated pattern model was discussed to address the *rare item problem* in frequent pattern mining [41]. We extend this model to find periodic-correlated patterns in a temporal database to address the *rare item problem* in periodic-frequent pattern mining. The proposed model considers a pattern as interesting if it satisfies the following two conditions: (i) if the $support$ of a pattern is close to the $support$ of its individual items, and (ii) if the $periodicity$ of a pattern is close to the $periodicity$ of its individual items. The *all-confidence* [41] measure is used to determine how close is the $support$ of a pattern with respect to the $support$ of its individual items. To the best of our knowledge, there exists no measure to determine how close is the $periodicity$ of a pattern with respect to the $periodicity$ of its items. So forth, we introduce a new measure, called *periodic-all-confidence*, to determine the interestingness of a pattern. The *periodic-all-confidence* measure is used to determine how close is the $periodicity$ of a pattern with respect to the $periodicity$ of its individual items. These two measures facilitate us to achieve the objective of generating periodic-correlated patterns containing both frequent and rare items yet without causing frequent items to generate too many uninteresting patterns.

A pattern-growth algorithm, called Extended Periodic-Correlated pattern-growth (EPCP-growth), has also been described to find all periodic-correlated patterns. We evaluate the performance of EPCP-growth by conducting extensive experiments on both synthetic and real-world databases. Experimental results demonstrate that the proposed model can tackle the *rare item problem* and EPCP-growth is runtime efficient and highly scalable as well.

1.4.2 Discovering Partial Periodic-Frequent Patterns in Non-Uniform Temporal Databases

Partial periodic-frequent patterns are an important class of regularities that exist in a temporal database. These patterns exhibit partial cyclic repetitions in a database. These patterns are a looser kind of full periodic-frequent patterns, and they exist ubiquitously in the real-world databases. Finding partial periodic-frequent patterns is thus useful to understand data. The proposed patterns can find useful information in many real-life applications.

In the second proposed approach, we have introduced a novel model to discover partial periodic-frequent patterns in non-uniform temporal databases, which considers that patterns may have different *period* and minimum number of cyclic repetitions. The proposed model lets the user specify a different

maximum inter-arrival time ($MIAT$) for each item. An inter-arrival time of a pattern is considered periodic (or cyclic) if it is no more than $period$. Thus, different patterns may satisfy different $period$ depending on their items' $MIAT$ values. This solves the *rare item problem* in partial periodic-frequent pattern mining. A new measure, *Relative Periodic-Support* (RPS), is proposed to determine the (partial) periodic interestingness of a pattern by considering the number of cyclic repetitions in the database. This measure assess the interestingness of a pattern by taking into account both the $support$ and $periodicity$ information of patterns.

A pattern-growth algorithm has been discussed to discover all partial periodic-frequent patterns. Experimental results demonstrate that the proposed model is efficient and tackles the problem of *rare item problem* as well as the problem of discovering partial periodic-frequent patterns. We discuss the usefulness of partial periodic-frequent patterns with a real-world case study and show that the proposed model may be utilized to find prior knowledge about event keywords and their associations in Twitter data.

1.5 Contribution of the thesis

The major contributions of this thesis are as follows:

1. Carried out literature survey on frequent pattern mining and periodic-frequent pattern mining in transactional databases and partial periodic pattern mining in time series.
2. We tackled *rare item problem* in periodic-frequent mining by proposing a novel model to discover *periodic-correlated* patterns.
3. We introduced a novel model to discover *partial periodic-frequent* patterns.
4. Experimental results demonstrate that the proposed algorithms are efficient. We discussed the usefulness of *periodic-correlated* and *partial periodic-frequent* patterns with a real-world case study, respectively.

1.6 Organization of the thesis

The rest of the thesis is organized as follows. Chapter 2 describes related work on frequent pattern mining, periodic-frequent pattern mining in temporal databases and (partial) periodic pattern mining in time series data and show how the proposed approaches are different from other approaches in the literature. Chapter 3 explains the background of frequent pattern mining and periodic-frequent pattern mining in transactional databases. Chapter 4 introduces the proposed model of finding periodic-correlated patterns in a temporal database. Chapter 5 introduces the proposed model of finding partial periodic patterns in a temporal database. Finally, chapter 6 discusses about the summary of thesis and concludes the thesis with future research directions.

Chapter 2

Related Work

Research efforts are being made in the field of data mining to mine interesting knowledge from the transactional databases. Frequent pattern mining was first introduced to extract knowledge from transactional databases and has been extended to various fields based on application. Periodic pattern mining comes under one of the extensions of frequent pattern mining. In this chapter, we first discuss about the research work attempted to address various issues of frequent pattern mining and periodic pattern mining. Next, we discuss the literature on finding (partial) periodic patterns in time series data.

The organization of this chapter is as follows. In Sections 2.1 and 2.2, we describe the literature related to frequent pattern mining and periodic-frequent pattern mining in transactional database, respectively. In Section 2.3, we describe the literature related to finding partial periodic patterns in time series data. In Section 2.4, we discuss about how the proposed approaches are different from the existing approaches. In Section 2.5, we conclude the chapter with summary.

2.1 Frequent pattern mining

Agrawal et al. [3] introduced the problem of finding frequent patterns in a transactional database. Since then, the problem of finding these patterns has received a great deal of attention [64, 24, 55, 23, 44, 15, 14]. The basic model used in most of these studies remained the same. It involves discovering all frequent patterns in a transactional database that satisfy the user-specified minimum support ($minSup$) constraint. The usage of a single $minSup$ for the entire database leads to the *rare item problem* (discussed in Section 1.3). When confronted with this problem in real-world applications, researchers have tried to address it using the concept of multiple $minSups$ [38, 28, 34]. In this concept, each item in the database is specified with a $support$-constraint known as *minimum item support* ($minIS$). Next, the $minSup$ of a pattern is represented with the lowest $minIS$ value of its items. Thus, every pattern can satisfy a different $minSup$ depending upon its items. A major limitation of this concept is that it suffers from an open problem of determining the items' $minIS$ values.

Brin et al. [11] introduced correlated pattern mining to address the *rare item problem*. The statistical measure, χ^2, was used to discover correlated patterns. Since then, several interestingness measures

have been discussed based on the theories in probability, statistics, or information theory. Examples include $all\text{-}confidence$, $any\text{-}confidence$, $bond$ [41] and $kulc$ [10]. Each measure has its own selection bias that justifies the rationale for preferring one pattern over another. As a result, there exists no universally acceptable best measure to discover correlated patterns in any given database. Researchers are making efforts to suggest an appropriate measure based on user and/or application requirements [53, 41, 56, 58, 50].

Recently, *all-confidence* is emerging as a popular measure to discover correlated patterns [37, 31, 60, 67, 68, 30]. It is because this measure satisfies both the *anti-monotonic* and *null-invariance* properties. The former property says that all non-empty subsets of a correlated pattern must also be correlated. This property plays a key role in reducing the search space, which in turn decreases the computational cost of mining the patterns. In other words, this property makes the correlated pattern mining practicable in real-world applications. The latter property discloses genuine correlation relationships without being influenced by the object co-absence in a database. In other words, this property facilitates the user to discover interesting patterns involving both frequent and rare items without generating a huge number of uninteresting patterns.

2.2 Periodic-frequent pattern mining

Frequent itemset mining is an important data mining task. Several $support$ related measures [41, 11] have been discussed to determine the interestingness of an itemset in a transactional database. Each measure has a selection bias that justifies the significance of a knowledge itemset. As a result, there exists no universally acceptable best measure to judge the interestingness of an itemset in any given database. Researchers have proposed criteria to select an interestingness measure based on user and/or application requirements [53]. Unfortunately, current measures cannot be used to determine the periodic interestingness of an itemset in temporal databases. This is because these measures only take the $support$ into account and completely ignore the temporal occurrence information of itemsets in databases. We introduce a new measure that assess the interestingness of itemsets by taking into account both the $support$ and temporal occurrence information of patterns.

Ozden et al. [42] enhanced the transactional database by a time attribute that describes the time when a transaction has appeared, investigated the periodic behavior of the itemsets to discover cyclic association rules. In this study, a database is fragmented into non-overlapping subsets with respect to time. The association rules that are appearing in at least a certain number of subsets are discovered as cyclic association rules. By fragmenting the data and counting the number of subsets in which an itemset occurs greatly simplifies the design of the mining algorithm. However, the drawback is that itemsets (or association rules) that span multiple windows cannot be discovered.

Tanbeer et al. [54] described a model to find full periodic-frequent itemsets in a transactional database. This model eliminates the need for data fragmentation and discovers all frequent itemsets that have exhibited complete (or full) cyclic repetitions in the transactional database that satisfy the

user-specified $minSup$ and $maxPer$ constraints. A pattern-growth algorithm, called Periodic-Frequent Pattern-growth (PFP-growth), was also discussed to find these patterns.

Uday et al. [32] have discussed a greedy search technique to efficiently compute the $periodicity$ of a pattern. Anirudh et al. [6] have proposed an efficient pattern-growth algorithm based on the concept of period summary. In this concept, the tid-list of the patterns are compressed into partial periodic summaries, and later aggregated to find periodic-frequent patterns efficiently. Rashid et al. [45] employed standard deviation of periods as a criterion to assess the periodic behavior of frequent patterns. The discovered patterns are known as regular frequent patterns.

Amphawan et al. [5] investigated the problem of finding top-k full periodic-frequent itemsets in a transactional database. The periodic patterns with the k highest support are know as *top-k* periodic-frequent patterns. This is proposed to make the mining process efficient. It is a single-pass algorithm and uses a best-first search strategy without any thresholds on the support constraint. Another advantage which the authors mention is that selecting the right $minSup$ is also a difficulty, which is resolved in this model.

The popular adoption and successful industrial application of periodic-frequent pattern model suffers from the following two issues: (i) cannot handle databases in which transactions are occurring at irregular time intervals and (ii) the rare item problem (both issues are discussed in Section 1.3).

To address the *rare item problem* in periodic-frequent pattern mining, Uday et al. [33] extended Liu's model [38]. In this model, every item in the database is specified with $minIS$ and $maxIP$ values. The usage of item-specific $minIS$ and $maxIP$ values facilitates every pattern to satisfy a different $minSup$ and $maxPer$ depending on its items. However, the major limitation of this model is the computational cost because the generated periodic-frequent patterns do not satisfy the *anti-monotonic property*.

Surana et al. [51] proposed alternative periodic-frequent pattern using the item-specific $minIS$ and $maxIP$ values. The periodic-frequent patterns discovered by this model satisfy the anti-monotonic property. Henceforth, this model is practicable in real-world applications.

An open problem that is common to above two studies [33, 51] is the methodology to specify items' $minIS$ and $maxIP$ values. Although the model facilitates every item to have different $minIS$ and $maxIP$ values, it suffers from the following limitations: (i) This methodology requires several input parameters from the user. (ii) We have observed that employing this methodology to specify items' $maxIP$ values in the temporal databases where items can have similar $support$ but different $periodicities$ can still lead to the *rare item problem*. We discuss these limitations in detail in Section 4.2.

The problem of finding full periodic-frequent patterns greatly simplifies the design of the model because there is no need of any measure to determine the partial periodic interestingness of a pattern. More importantly, these studies consider transactional database as a symbolic sequence of transactions and ignore the temporal occurrence information of the transactions in a database.

2.3 Partial Periodic pattern mining in time series data

Periodic patterns are an important class of regularities that exist in a time series data. Han et al. [25] divided periodic patterns into two types of patterns, full periodic and partial periodic patterns. The former considers that every point in the period contributes to the cycle behavior of the time series, such as all the hours (days) in a day (year). According to the latter, some but not all points in the period contribute to the cycle behavior of the time series. Thus, partial periodic patterns are a looser kind of full periodic patterns and exist ubiquitously in real-world. The authors have also discussed a model to find partial periodic patterns. The model involves the following two steps [25]:

- Segmenting the given time series data into multiple period-segments such that each period-segment represents a set of events occurred within a particular time interval (or $period$[1]).
- Discovering all partial periodic patterns that satisfy the user-defined *minimum support* ($minSup$). The $minSup$ controls the minimum number of period-segm- ents in which a pattern must appear.

Aref et al. [8] extended the Han's model to incremental mining of partial periodic patterns. Yang et al. [62] used *information gain* as an alternative interestingness measure for $support$ to find partial periodic patterns. Yang et al. [61] studied the change in periodic behavior of a pattern due to the influence of noise, and enhanced the basic model to discover a class of partial periodic patterns known as asynchronous periodic patterns. Zhang et al. [65] enhanced the basic model to discover periodic patterns in character sequences like protein data. The popular adoption and successful industrial application of this basic model suffers from the following obstacles:

- **Computationally expensive model:** Periodic patterns satisfy the *anti-monotonic property* [3]. That is, all non-empty subsets of a periodic pattern are also periodic patterns. However, this property is insufficient to make the periodic pattern mining practical or computationally inexpensive in real-life. The reason is that number of frequent i-patterns shrink slowly (when $i > 1$) as i increases in time series data. The slow speed of decrease in the number of frequent i-patterns is due to a strong correlation between frequencies of patterns and their sub-patterns [24].
- **Rare item problem:** The usage of single $minSup$ for the entire time series leads to the *rare item problem*.
- **Inability to consider temporal information of events:** The basic model of periodic patterns implicitly considers time series as a symbolic sequence and thus as an evenly spaced time series (i.e., all events within a series occur at a fixed time interval). As a result, this model does not take into account the actual temporal information of events within a sequence [39]. This assumption limits the applicability of the model as events in many real-world time series datasets occur at irregular time intervals.

[1] A time series can be segmented using multiple periods

Han et al. [24] introduced "maximum sub-pattern hit set property" to reduce the computational cost of finding these patterns. Chen et al. [13] proposed a pattern-growth algorithm to discover partial periodic patterns.

Recently, Uday et al. [35] discussed a model to find partial periodic-frequent patterns in a transactional database. The partial periodic-frequent patterns do not satisfy the anti-monotonic property [3]. As a result, the model is computationally expensive to use in real-world very large databases. Also, this model suffers from *rare item problem*. Moreover, this model cannot handle temporal databases as multiple transactions can share a common timestamp.

2.4 How proposed approaches are different

To the best of our knowledge, there exists no study that takes into account the temporal occurrence information of the events in a time series. On the contrary, in this thesis, we propose models to find interesting periodic-frequent patterns by taking into account the temporal occurrence information of the transactions in the data. More importantly, since transactions in a temporal database can share a common timestamp, these databases cannot be represented as a time series.

In the first proposed approach of this thesis, we use *all-confidence* to address the *rare item problem* in $support$ dimension. As there exists no measure in the literature to address the *rare item problem* in $periodicity$ dimension, we propose a new measure, *periodic-all-confidence* to extract interesting patterns in periodicity dimension involving rare items. In the second proposed approach, we introduce a novel model to discover partial periodic patterns by taking into account the temporal occurrence information of the transactions in a database.

In both of our approaches, we use temporal database (instead of transactional database) thereby taking into account the temporal occurrence information of the transactions in the data. Overall, both the proposed approaches are novel and distinct from previous studies.

2.5 Summary

In this chapter, we have explained all the existing literature related to frequent pattern mining, periodic-frequent pattern mining in transactional databases and periodic pattern mining in time series data. It can be noted that that the existing approaches for periodic-frequent pattern mining suffer from *rare item problem* and are unable to discover partial periodic-frequent patterns. In the upcoming chapters, we explain the proposed ideas which tackle these problems in existing literature.

Chapter 3

Background

As part of the background, we explain a few approaches in a more elaborated manner. We first explain the model of frequent patterns and then we explain the frequent pattern growth algorithm proposed by Han et al. [26] for mining frequent patterns. Later, we explain the model of periodic-frequent patterns and then we explain the periodic-frequent pattern growth proposed by Tanbeer et al. [54]. In the end, we conclude the chapter with summary.

3.1 Model of frequent patterns

The basic model of frequent patterns [26] is as follows. Let I be the set of items, and $X \subseteq I$ be a **pattern** (or an itemset). A pattern containing β number of items is called a β**-pattern**. A **transaction**, $t_k = (tid, Y)$ is a tuple, where $tid \in \mathbb{R}$ represents the transaction-id at which the pattern Y has occurred. A **transactional database** TDB over I is a set of transactions, $TDB = \{t_1, \cdots, t_m\}$, $m = |TDB|$, where $|TDB|$ can be defined as the number of transactions in TDB. For a transaction $t_k = (tid, Y)$, $k \geq$1, such that $X \subseteq Y$, it is said that X occurs in t_k and such transaction-id is denoted as tid^X. Let $TID^X = \{tid^X_j, \cdots, tid^X_k\}$, $j, k \in [1, m]$ and $j \leq k$, be a set of transaction-ids where X has occurred in TDB. We call this list of transaction-ids of X as **tid-list** of X. The number of transactions containing X in TDB is defined as the **support** of X and denoted as $sup(X)$. That is, $sup(X) = |TID^X|$. The pattern X is a **frequent pattern** if $sup(X) \geq minSup$, where $minSup$ refers to the user-specified *minimum support* constraint. The **problem definition** of frequent pattern mining involves discovering all patterns in TDB that satisfy the user-specified $minSup$ constraint.

Example 1 *Table 3.2 shows the transactional database with the set of items* $I = \{a, b, c, d, e, f, g, h\}$. *The set of items 'a' and 'b,' i.e., 'ab' is a pattern. This pattern contains only two items. Therefore, this is a 2-pattern. In the first transaction,* $t_1 = \{1 : ab\}$, *1 denotes the transaction-id at which the pattern 'ab' has appeared in the database. In the entire database, this pattern appears at the transaction-ids of* $1, 2, 5, 7$ *and* 10. *Therefore,* $TID^{ab} = \{1, 2, 5, 7, 10\}$. *The* support *of 'ab,' i.e.,*

Table 3.1 Nomenclature of various terms used in frequent pattern model

Terminology	Symbol
The complete set of items	I
Transactional database	TDB
The transaction-id of a transaction containing X	tid_i^X
The set of all transaction-ids containing X	TID^X
The $support$ of X	$sup(X)$
The user-defined minimum support	$minSup$

Table 3.2 An example of a transactional database

TID	Items	TID	Items
1	$a,\ b$	7	$a,\ b,\ c,\ e$
2	$a,\ b,\ d$	8	$c,\ d$
3	$c,\ d,\ g$	9	$c,\ d$
4	$c,\ e,\ f$	10	$a,\ b,\ e,\ f$
5	$a,\ b$	11	$c,\ d,\ g$
6	h	12	$a,\ e,\ f$

$sup(ab) = |TID^{ab}| = |1, 2, 5, 7, 10| = 5$. *If the user-specified* $minSup = 2$*, then 'ab' is a frequent pattern as* $sup(ab) \geq minSup$*. Table 3.3 represents the frequent patterns generated using frequent-patterns model at* $minSup = 3$*.*

3.2 Frequent pattern mining using FP-growth

Agarwal et al. [4] first introduced Apriori algorithm to extract frequent patterns from transactional databases. But the time taken by this algorithm is high because it scans the entire database for each of the k-sized patterns ($k = [1, n]$, n is the number of items in the database). Han et al. [26] proposed frequent pattern growth (FP-growth) to extract frequent patterns in an efficient manner. FP-growth requires only two database scans.

FP-growth is based on the *anti-monotonic property*, which says that all non-empty subsets of a frequent pattern must also be frequent. This property plays a key role in reducing the search space,

Table 3.3 Patterns generated using FP-growth Approach for transactional database in Table 3.2

Pat	sup	Pat	sup
a	6	f	3
c	6	ab	5
b	5	cd	4
d	5	ae	3
e	4	ef	3

which in turn decreases the computational cost of mining the patterns. In onter words, this property makes the frequent pattern mining practicable in real-world applications. FP-growth consists of the following two steps: (i) compressing the transactional database into a prefix-tree like structure called frequent pattern tree (FP-tree) and (ii) recursively mining the FP-tree using pattern-growth approach to discover the complete set of frequent patterns. Before we describe these two steps, we first explain the structure of FP-tree.

3.2.1 Structure of FP-tree

The structure of FP-tree contains an FP-list and a prefix-tree. The FP-list consists of $itemname$ (item) and $support$ (sup). Support is the frequency of the item in the transactional database. The items in FP-tree are sorted in descending order of $support$ to facilitate high compactness. Each node in the FP-tree keeps track of the support value and maintains parent-child relationships. The header table maintains the list of frequent items sorted based on support and each item in the header table contain node links to all the occurrences of that item.

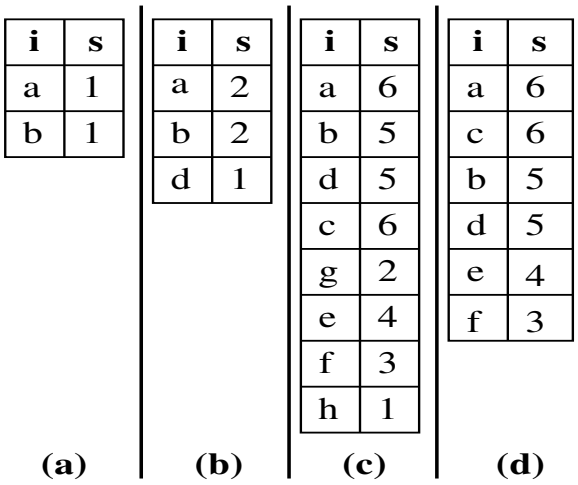
3.2.2 Construction of FP-tree

FP-growth requires two database scans to represent the database in a compact structure, FP-tree. In the first database scan, the FP-list is constructed which contains the frequent items of size 1. Since frequent patterns follow anti monotonic property, frequent items (frequent patterns of unit length) play a key role. Therefore, the first database scan to extract one-sized frequent patterns is an important step. Now only these items in the FP-list will be used for the construction of the prefix tree and the insertion of a branch is done in the same order as in FP-list. Frequent items are discovered by populating the FP-list (lines 1 to 10 in Algorithm 1). Figures 3.1 (a), (b), (c) and (d) show the steps involved in finding frequent items from FP-list.

In the second database scan, the items in the FP-list will take part in the construction of prefix tree (lines 11 to 15 in Algorithm 2). The elements which are not present in the FP-list are not considered while inserting into the prefix tree. While inserting a transaction into the tree, support of the nodes in that branch are incremented by one. Figures 3.2 (a), (b) and (c) show the construction of FP-tree after scanning first, second and every transaction in the transactional database, respectively. In FP-tree, an item header table is built so that each item points to its occurrences in the tree via a chain of node-links, to facilitate tree traversal. For simplicity, we do not show these node-links in trees.

3.2.3 Mining of FP-tree

Algorithm 3 describes the procedure for mining frequent patterns from FP-tree. The FP-tree is mined by calling FP-growth as (FP-tree, *null*). The working of this algorithm is as follows. We proceed to construct the prefix tree for each candidate item in the FP-list, starting from the bottom most item, say i.



i	s
a	1
b	1

(a)

i	s
a	2
b	2
d	1

(b)

i	s
a	6
b	5
d	5
c	6
g	2
e	4
f	3
h	1

(c)

i	s
a	6
c	6
b	5
d	5
e	4
f	3

(d)

Figure 3.1 Construction of FP-list for Table 4.2. (a) After scanning the first transaction (b) After scanning the second transaction (c) After scanning every transaction (d) Final FP-list with sorted list of frequent items

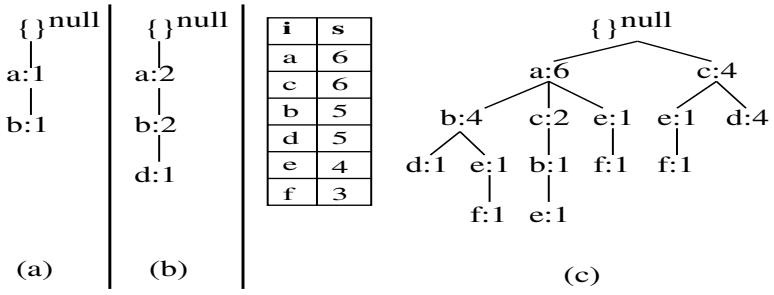


Figure 3.2 Construction of FP-tree for Table 4.2. (a) After scanning first transaction (b) After scanning second transaction (c) After scanning every transaction

Algorithm 1 Construction of FP-tree (TDB: Transactional database, $minSup$: minimum support)

1: Let $t = \{tid_{cur}, X\}$ denote the current transaction with tid_{cur} and X representing the transaction-id of the current transaction and pattern, respectively.
2: **for** each transaction $t \in TDB$ **do**
3: **if** an item i occurs for the first time **then**
4: Insert i into the FP-list with $sup^i = 1$.
5: **else**
6: $sup^i = sup^i + 1$.
7: **end if**
8: **end for**
9: Remove items from the FP-list that do not satisfy $minSup$.
10: Sort the remaining items in FP-list in descending order of their $support$. Let this sorted list of items be FP.
11: Create a root node in FP-tree, T, and label it "$null$."
12: **for** each transaction $t \in TDB$ **do**
13: Sort the items in t in FP order. Let this list of sorted frequent items in t be $[p|P]$, where p is the first item and P is the remaining list.
14: Call $insert\_tree([p|P], T)$.
15: **end for**

Algorithm 2 Insert_tree($[p|P]$, T)

1: **if** T does not have a child N satisfying $p.itemName = N.itemName$ **then**
2: Create a new node N and set its parent as T. Let its node-link be linked to the nodes with the same item via the node-link structure.
3: **end if**
4: Increase the support count of node N by 1.
5: Remove p from P.
6: **if** P is non-empty **then**
7: Call Insert_tree($[P]$, N)
8: **end if**

To construct the prefix-tree for i, the prefix sub-paths of node i are accumulated in a tree-structure, PT_i. Since i is the bottom-most item in the FP-list, each node labeled i in the FP-tree must be a tail node. If an item j in PT_i has $sup \geq minSup$, then we construct its conditional tree and mine it recursively to discover the recurring patterns (lines 3 to 9 in Algorithm 3). All nodes of i in the original FP-tree and is entry in the FP-list are deleted thereafter. (line 10 in Algorithm 3).

The conditional tree CT_i for PT_i is constructed by removing all those items from PT_i that have $sup \leq minSup$. The same process of creating a prefix-tree and its corresponding conditional tree is repeated for further extensions of "ij". The whole process of mining for each item is repeated until FP-list $\neq \emptyset$.

First, we consider item 'f,' which is the bottom-most item in the FP-list, as a suffix pattern. This item appears in three branches of the FP-tree (refer Figure 3.2(c)). The paths formed by these branches are $\{c, e, f : 1\}$, $\{a, b, e, f : 1\}$ and $\{a, e, f : 1\}$ (format of these branches is $\{nodes : support\}$).

Therefore, considering 'f' as a suffix item, its corresponding three prefix paths are $\{c, e : 1\}$, $\{a, b, e : 1\}$ and $\{a, e : 1\}$, which form its conditional pattern base (refer Figure 3.3(a)). Its conditional FP-tree contains only a single path, $\langle e : 3\rangle$; 'a,' 'b' and 'c' are not included because their *sup* do not satisfy $minSup$. Figure 3.3(b) shows the conditional FP-tree of 'f.' The single path generates the pattern $\{ef : 3\}$ (format is {*pattern*: $support$}). The same process of creating prefix-tree and its corresponding conditional tree is repeated for further extensions of 'ef.' Next, 'f' is pruned from the original FP-tree as shown in Figure 3.3(c). All the above processes are once again repeated until FP-list $\neq \emptyset$. Table 3.3 represents the frequent patterns discovered in transactional database in Table 3.2 at $minSup = 3$.

Algorithm 3 FP-growth($Tree$, α)

```
1:  for each a_i in the header of Tree do
2:     Generate pattern β = a_i ∪ α with sup(β) = sup(a_i).
3:     if sup(β) ≥ minSup then
4:        Output β as a frequent pattern as {β, sup(β)}.
5:        Traverse Tree using the node-links of β, and construct β's conditional pattern base and β's
          conditional FP-tree Tree_β.
6:        if Tree_β ≠ ∅ then
7:           call FP-growth(Tree_β, β);
8:        end if
9:     end if
10:    Remove a_i from the Tree.
11: end for
```

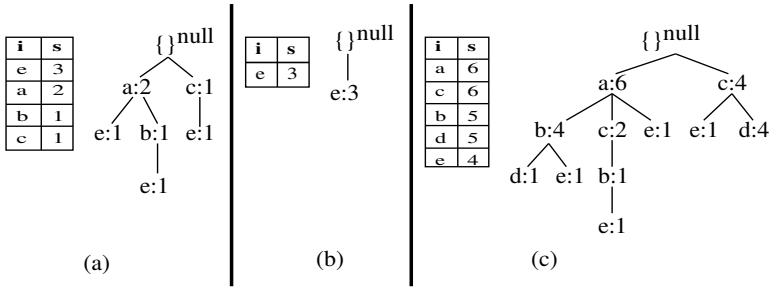


Figure 3.3 Mining of FP-tree for Table 4.2. (a) Prefix-tree of suffix item '*f*,' i.e., PT_f (b) Conditional tree of suffix item '*f*,' i.e., CT_f (c) FP-tree after pruning item '*f*.'

We now explain the model of periodic-frequent patterns followed by mining of periodic-frequent patterns using PF-growth.

3.3 Model of periodic-frequent patterns

The basic model of periodic-frequent patterns [54] is as follows. Let I be the set of items, and $X \subseteq I$ be a **pattern** (or an itemset). A pattern containing β number of items is called a β**-pattern**. A

transaction, $t_k = (tid, Y)$ is a tuple, where $tid \in \mathbb{R}$ represents the transaction-id at which the pattern Y has occurred. A **transactional database** TDB over I is a set of transactions, $TDB = \{t_1, \cdots, t_m\}$, $m = |TDB|$, where $|TDB|$ can be defined as the number of transactions in TDB. For a transaction $t_k = (tid, Y)$, $k \geq 1$, such that $X \subseteq Y$, it is said that X occurs in t_k and such transaction-id is denoted as tid^X. Let $TID^X = \{tid^X_j, \cdots, tid^X_k\}$, $j, k \in [1, m]$ and $j \leq k$, be an **ordered set of transaction-ids** where X has occurred in TDB. We call this list of transaction-ids of X as **tid-list** of X. The number of transactions containing X in TDB is defined as the **support** of X and denoted as $sup(X)$. That is, $sup(X) = |TID^X|$. Let tid^X_q and tid^X_r, $j \leq q < r \leq k$, be the two consecutive transaction-ids in TID^X. The time difference between tid^X_r and tid^X_q is defined as a **inter-arrival times** of X, say p^X_a. That is, $p^X_a = tid^X_r - tid^X_q$. Let $P^X = (p^X_1, p^X_2, \cdots, p^X_r)$ be the set of all *inter-arrival times* for a pattern X. The **periodicity** of X denoted as $per(X) = max(p^X_1, p^X_2, \cdots, p^X_r)$. The pattern X is a **frequent pattern** if $sup(X) \geq minSup$, where $minSup$ refers to the user-specified *minimum support* constraint. The frequent pattern X is said to be **periodic-frequent** if $per(X) \leq maxPer$, where $maxPer$ refers to the user-specified *maximum periodicity* constraint. The **problem definition** of periodic-frequent pattern mining involves discovering all patterns in TDB that satisfy the user-specified $minSup$ and $maxPer$ constraints.

.

Table 3.4 Nomenclature of various terms used in periodic-frequent pattern model

Terminology	Symbol
The complete set of items	I
Transactional database	TDB
The transaction-id of a transaction containing X	tid^X_i
The set of all transaction-ids containing X	TID^X
The $support$ of X	$sup(X)$
The user-defined minimum support	$minSup$
The $periodicity$ of X	$per(X)$
The user-defined maximum period	$maxPer$

Example 2 *Table 3.2 shows the transactional database with the set of items* $I = \{a, b, c, d, e, f, g, h\}$. *The set of items 'a' and 'b,' i.e., 'ab' is a pattern. This pattern contains only two items. Therefore, this is a 2-pattern. In the first transaction,* $t_1 = \{1 : ab\}$, *1 denotes the transaction-id at which the pattern 'ab' has appeared in the database. In the entire database, this pattern appears at the transaction-ids of* $1, 2, 5, 7$ *and* 10. *Therefore,* $TID^{ab} = \{1, 2, 5, 7, 10\}$. *The* support *of 'ab,' i.e.,* $sup(ab) = |TID^{ab}| = |1, 2, 5, 7, 10| = 5$. *If the user-specified* $minSup = 2$, *then 'ab' is a frequent pattern as* $sup(ab) \geq minSup$. *The inter-arrival times for this pattern are:* $p^{ab}_1 = 1\ (= 1 - tid_{ini})$, $p^{ab}_2 = 1\ (= 2 - 1)$, $p^{ab}_3 = 3\ (= 5 - 2)$, $p^{ab}_4 = 2\ (= 7 - 5)$, $p^{ab}_5 = 3\ (10 - 7)$ *and* $p^{ab}_6 = 2\ (=$

Table 3.5 Patterns generated using PF-growth Approach for transactional database in Table 3.2

Pat	sup	per	Pat	sup	per
a	6	3	f	3	6
c	6	3	ab	5	3
b	5	3	cd	4	5
d	5	5	ef	3	6
e	4	4	ce	2	5

$tid_{fin} - 10$), *where* $tid_{ini} = 0$ *represents the transaction-id of initial transaction and* $tid_{fin} = 12$ *represents the transaction-id of final transaction in the database. Therefore,* $P^{ab} = (1, 1, 3, 2, 3, 2)$. *The* $periodicity$ *of 'ab,' i.e.,* $per(ab) = max(1, 1, 3, 2, 3, 2) = 3$. *If the user-defined* $maxPer = 6$, *then the frequent pattern 'ab' is a periodic-frequent pattern because* $per(ab) \leq maxPer$. *Table 3.5 represent the periodic-frequent patterns generated using periodic-frequent model at* $minSup = 2$ *and* $maxPer = 6$.

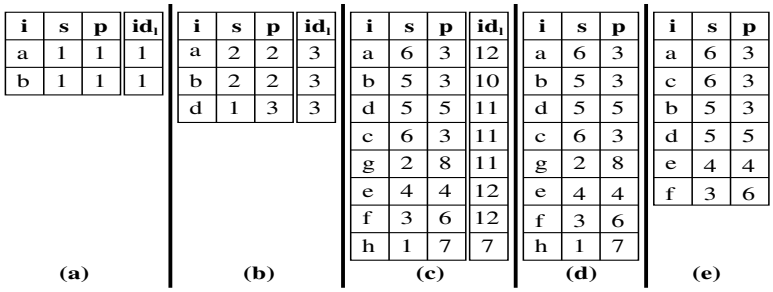
3.4 Periodic-frequent pattern mining using PF-growth

Tanbeer et al. [54] proposed periodic-frequent pattern growth (PF-growth) to extract periodic-frequent patterns from transactional databases. PF-growth consists of the following two steps: (i) compressing the transactional database into a prefix-tree like structure called periodic-frequent pattern tree (PF-tree) and (ii) recursively mining the PF-tree using pattern-growth approach to discover the complete set of periodic-frequent patterns. The algorithm accepts a transactional database (TDB), $minSup$ and $maxPer$ as inputs and outputs a complete set of periodic-frequent patterns.

3.4.1 Structure of PF-tree

The structure of PF-tree contains a PF-list and a prefix-tree. A PF-list consists of three fields - $itemname$ (Item), $support$ (sup) and $periodicity$ (per). The items in PF-tree are sorted in descending order of $support$ to facilitate high compactness. Two types of nodes are maintained in PF-tree - ordinary node and tail-node. The former is the type of node similar to that used in FP-tree, whereas the latter node represents the last item of any sorted transaction. However, the nodes in the PF-tree do not maintain the $support$ count as in FP-tree. Instead, the tail-nodes explicitly maintain the occurrence information for each transaction by maintaining an occurrence transaction-id list, called $tid\text{-}list$, only at the tail node of every transaction.

One can assume that the structure of the prefix-tree in a PF-tree may not be memory efficient since it explicitly maintains transaction-ids of each transaction. However, it has been argued that such a tree



i	s	p	id_l
a	1	1	1
b	1	1	1

(a)

i	s	p	id_l
a	2	2	3
b	2	2	3
d	1	3	3

(b)

i	s	p	id_l
a	6	3	12
b	5	3	10
d	5	5	11
c	6	3	11
g	2	8	11
e	4	4	12
f	3	6	12
h	1	7	7

(c)

i	s	p
a	6	3
b	5	3
d	5	5
c	6	3
g	2	8
e	4	4
f	3	6
h	1	7

(d)

i	s	p
a	6	3
c	6	3
b	5	3
d	5	5
e	4	4
f	3	6

(e)

Figure 3.4 Construction of PF-list for Table 4.2. (a) After scanning the first transaction (b) After scanning the second transaction (c) After scanning every transaction (d) Updated PF-list (e) Final PF-list with sorted list of periodic-frequent items

can achieve memory efficiency by keeping transaction information only at the tail nodes and avoiding the support count field at each node [54].

3.4.2 Construction of PF-tree

PF-growth requires two database scans to represent the database in a compact structure, PF-tree. In the first database scan, the PF-list is constructed which contains the periodic-frequent items of size 1. Since the periodic-frequent patterns generated by the proposed model satisfy the *anti-monotonic property*, periodic-frequent items (or 1-patterns) play a key role in efficient discovery of higher order periodic-frequent patterns. Periodic-frequent items are discovered by populating the PF-list (lines 1 to 18 in Algorithm 4). Figures 3.4 (a), (b), (c), (d) and (e) show the steps involved in finding periodic-frequent items from PF-list.

In the second database scan, the items in the FP-list will take part in the construction of prefix tree (lines 19 to 23 in Algorithm 5). The construction of prefix-tree in PF-tree is similar to the construction of prefix-tree in FP-tree [26]. However, it has to be noted that leaf nodes in FP-tree maintain the transaction-ids of the transactions. The elements which are not present in the PF-list are not considered while inserting into the prefix tree. While inserting a transaction into the tree, support of the nodes in that branch are incremented by one. Figures 3.5 (a), (b) and (c) show the construction of PF-tree after scanning first, second and every transaction in the transactional database, respectively.

3.4.3 Mining of PF-tree

Algorithm 6 describes the procedure for mining periodic-frequent patterns from PF-tree. The PF-tree is mined by calling PF-growth as (PF-tree, *null*). This algorithm resembles FP-growth. However, the key difference is that once the pattern-growth is achieved for a suffix 1-pattern (or item), it is completely pruned from the PF-tree by pushing its tid-list to respective parent nodes.

The working of this algorithm is as follows. We proceed to construct the prefix tree for each candidate item in the PF-list, starting from the bottom most item, say i. To construct the prefix-tree for i, the prefix

Algorithm 4 Construction of PF-tree (TDB: Transactional database, $minSup$: minimum support, $maxPer$: maximum periodicity)

1: Let id_l be a temporary array that records the tid of the last appearance of each item in the TDB. Let $t = \{ts_{cur}, X\}$ denote the current transaction with tid_{cur} and X representing the transaction-id of the current transaction and pattern, respectively.
2: **for** each transaction $t \in TDB$ **do**
3: **if** an item i occurs for the first time **then**
4: Insert i into the PF-list with $sup^i = 1$, $per^i = tid_{ini} - tid_{cur}$ and $id^i_l = 1$.
5: **else**
6: $sup^i = sup^i + 1$.
7: **if** $(tid_{cur} - id^i_l) > per^i$ **then**
8: $per^i = tid_{cur} - id^i_l$.
9: **end if**
10: **end if**
11: **end for**
12: **for** each item i in PF-list **do**
13: **if** $(tid_{fin} - id^i_l) > per^i$ **then**
14: $per^i = tid_{fin} - id^i_l$.
15: **end if**
16: **end for**
17: Remove items from the PF-list that do not satisfy $minSup$ and $maxPer$.
18: Sort the remaining items in PF-list in descending order of their $support$. Let this sorted list of items be PF.
19: Create a root node in PF-tree, T, and label it "$null$."
20: **for** each transaction $t \in TDB$ **do**
21: Sort the items in t in PF order. Let this list of sorted periodic-frequent items in t be $[p|P]$, where p is the first item and P is the remaining list.
22: Call $insert\_tree([p|P], tid_{cur}, T)$.
23: **end for**

sub-paths of node i are accumulated in a tree-structure, PT_i. Since i is the bottom-most item in the PF-list, each node labeled i in the PF-tree must be a tail node. While constructing PT_i, we map the tid-list of every node of i to all items in the respective path explicitly in the temporary array (one for each item). This temporary array facilitates the calculation of sup and per of each item in PT_i (line 2 in Algorithm 6). If an item j in PT_i has $sup \geq minSup$ and $per \leq maxPer$, then we construct its conditional tree and mine it recursively to discover the recurring patterns (lines 3 to 9 in Algorithm 6). Moreover, to enable the construction of the prefix-tree for the next item in the PF-list, the tid-lists are pushed-up to the respective parent nodes in the original PF-tree and in PT_i as well. All nodes of i in the original PF-tree and is entry in the PF-list are deleted thereafter (line 10 in Algorithm 6).

The conditional tree CT_i for PT_i is constructed by removing all those items from PT_i that have $sup \leq minSup$ or $per \geq maxPer$. If the deleted node is a tail node, its tid-list is pushed-up to its parent node. The contents of the temporary array for the bottom item j in the PF-list of CT_i represent TID_{ij} (i.e., the set of all transaction-ids where items i and j have appeared together in the database).

Algorithm 5 Insert_tree($[p|P]$, tid_{cur}, T)

1: **if** T does not have a child N satisfying $p.itemName = N.itemName$ **then**
2: Create a new node N and set its parent as T. Let its node-link be linked to the nodes with the same item via the node-link structure.
3: **end if**
4: Remove p from P.
5: **if** P is non-empty **then**
6: Call Insert_tree($[P]$, tid_{cur}, N)
7: **else**
8: Add tid_{cur} to T (i.e., leaf node).
9: **end if**

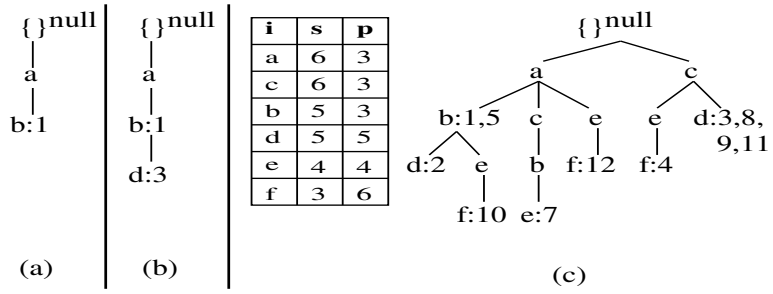


Figure 3.5 Construction of PF-tree for Table 4.2. (a) After scanning first transaction (b) After scanning second transaction (c)After scanning every transaction

The same process of creating a prefix-tree and its corresponding conditional tree is repeated for further extensions of "ij". The whole process of mining for each item is repeated until PF-list $\neq \emptyset$.

First, we consider item 'f,' which is the bottom-most item in the PF-list, as a suffix pattern. This item appears in three branches of the PF-tree (refer Figure 3.5(c)). The paths formed by these branches are $\{cef : 4\}$, $\{abef : 10\}$ and $\{aef : 12\}$ (format of these branches is $\{nodes : transaction - ids\}$). Therefore, considering 'f' as a suffix item, its corresponding three prefix paths are $\{ce : 4\}$, $\{abe : 10\}$ and $\{ae : 12\}$, which form its conditional pattern base (refer Figure 3.6(a)). Its conditional PF-tree contains only a single path, $\langle e : 4, 10, 12\rangle$; '$a$,' '$b$' and '$c$' are not included because their *support* and *periodicity* do not satisfy the $minSup$ and $maxPer$ respectively. Figure 3.6(b) shows the conditional PF-tree of 'f.' The single path generates the pattern $\{ef : 3, 6\}$ (format is {*pattern*: $support, periodicity$}). The same process of creating prefix-tree and its corresponding conditional tree is repeated for further extensions of 'ef.' Next, 'f' is pruned from the original PF-tree and its ts-lists are pushed to its parent nodes, as shown in Figure 3.6(c). All the above processes are once again repeated until PF-list $\neq \emptyset$. Table 3.5 represents the periodic-frequent patterns discovered in transactional database in Table 3.2 at $minSup = 2$ and $maxPer = 6$.

Algorithm 6 PF-growth($Tree$, α)

1: **for** each a_i in the header of $Tree$ **do**
2: Generate pattern $\beta = a_i \cup \alpha$. Construct an array TID^{β}, which represents the set of transaction-ids at which β has appeared in TDB. Next, compute from TID^{β}, $sup(\beta)$ and $per(\beta)$ and compare them with $minSup$ and $maxPer$, respectively.
3: **if** $sup(\beta) \geq minSup$ and $per(\beta) \leq maxPer$ **then**
4: Output β as a periodic-frequent pattern as $\{\beta$: sup, per$\}$.
5: Traverse $Tree$ using the node-links of β, and construct β's conditional pattern base and β's conditional PF-tree $Tree_{\beta}$.
6: **if** $Tree_{\beta} \neq \emptyset$ **then**
7: call PF-growth($Tree_{\beta}$, β);
8: **end if**
9: **end if**
10: Remove a_i from the $Tree$ and push a_i's tid-list to its parent nodes.
11: **end for**

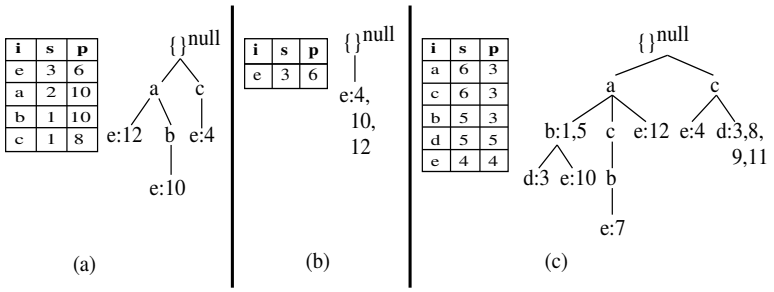


Figure 3.6 Mining of PF-tree for Table 3.2. (a) Prefix-tree of suffix item '*f*,' i.e., PT_f (b) Conditional tree of suffix item '*f*,' i.e., CT_f (c) PF-tree after pruning item '*f*.'

3.5 Summary

In this chapter, we have explained the existing approach to mine frequent patterns and periodic-frequent patterns. In the next chapter, we discuss the *rare item problem* in periodic-frequent pattern mining and propose a novel model to address the problem.

Chapter 4

Discovering Periodic-Correlated Patterns in Non-Uniform Temporal Databases

In this chapter, to solve the *rare item problem* we have proposed a model to discover periodic-correlated patterns in non-uniform temporal databases. The proposed model considers a pattern as interesting if it satisfies the following two conditions: (i) if the support of a pattern is close to the support of its individual items, and (ii) if the periodicity of a pattern is close to the periodicity of its individual items.

The organization of the chapter is as follows. Section 4.1 explains the motivation to the existing problem in periodic-frequent pattern mining. Section 4.3 introduces the basic idea we are proposing to solve the problem. Section 4.4 modifies the existing periodic-frequent pattern model [54] by taking into account the temporal databases. Section 4.5 introduces the proposed model of finding periodic-correlated patterns in a temporal database. Section 4.6 describes EPCP-growth algorithm. Section 4.7 reports on experimental results. Finally, Section 4.8 concludes the chapter with summary.

4.1 Motivation

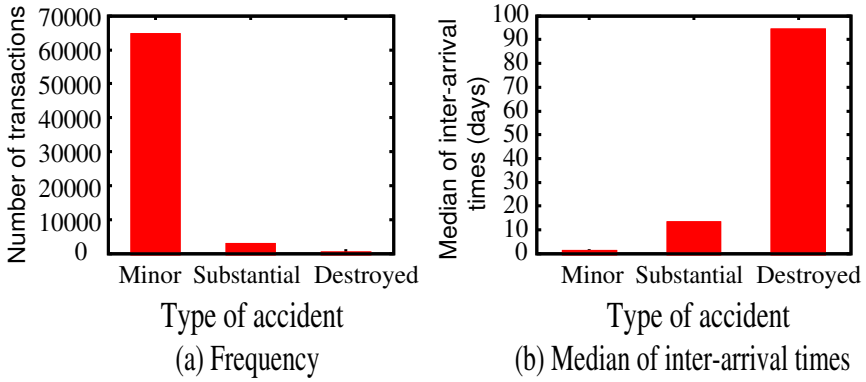
Periodic-frequent pattern[1] mining is an important model in data mining. It involves discovering all patterns in a transactional database that satisfy the user-specified minimum support ($minSup$) and maximum periodicity ($maxPer$) constraints [54]. The $minSup$ controls the minimum number of transactions that a pattern must cover, and the $maxPer$ controls the maximum interval within which a pattern must reoccur in the entire database. Finding periodic-frequent patterns is a significant task with many business applications. A classic application is market-basket analytics. It analyzes how regularly the itemsets are being purchased by the customers. An example of a periodic-frequent pattern is as follows:

$$\{Bat, Ball\} \quad [support = 5\%, \ \ periodicity = 1 \ hour].$$

The above pattern says that 5% of the purchases have the items 'Bat' and '$Ball$,' and the maximum duration between any two consecutive purchases containing both of these items is no more than an hour.

[1] A set of items represents a pattern (or an itemset)

Figure 4.1 Statistics on different damage types in FAA Accidents database.



This predictive behavior of the customers' purchases may facilitate the users in product recommendation and inventory management.

The *support* and $periodicity$ are two dimensions to determine the interestingness of a periodic-frequent pattern. The constraints, $minSup$ and $maxPer$, determine the interestingness of a pattern with respect to these two dimensions. Since only a single $minSup$ and $maxPer$ is used for the whole database, the model works efficiently in the databases in which all the items have uniform $support$ and similar $periodicity$. However, this is often not the case as items are non-uniformly distributed in many real-world databases. That is, some items appear very frequently in the data, while others rarely appear. Moreover, rare items typically have longer *inter-arrival times* as compared with the *inter-arrival times* of the frequent items.

Example 3 *Consider a case of accident database in which reports related to the completely* destroyed *vehicles do not appear as frequently as the reports related to the* minor *damages to a vehicle. As a result, former type of accidents generally have less frequency and longer inter-arrival times as compared against the latter type of accidents. The same can be observed from Figures 4.1(a) and (b), which respectively show the $frequency$ and $median$ of inter-arrival times of three different damage types reported in the Federal Aviation Administration (FAA) database [18]. For a domain expert, any information pertaining to* destroyed *aircrafts may be found useful as it includes materialistic and/or human loss.*

Henceforth, finding periodic-frequent patterns with a single $minSup$ and $maxPer$ leads to the following problems:

- If the $minSup$ is set too high and/or the $maxPer$ is set too short, we will miss the interesting periodic-frequent patterns involving rare items.
- In order to find the interesting periodic-frequent patterns involving rare items, we have to set a low $minSup$ and a long $maxPer$. However, this may result in combinatorial explosion producing

too many patterns, because frequent items can combine with one another in all possible ways and many of them will be meaningless.

This dilemma is known as the *rare-item problem* [57].

Also, since this model accepts the transactional database as an input, the model implicitly assumes that all transactions in a database occur at a fixed time-interval. This assumption limits the applicability of the model as transactions in many real-world databases occur at irregular time intervals. In particular, multiple transactions can share a common timestamp and uneven time gaps can exist in between the consecutive transactions.

In this chapter, we make an effort to address these problems by proposing a novel model to discover periodic-correlated patterns in temporal databases.

4.2 Limitations of Existing Approaches

Prior to our study, researchers have tried to address the *rare item problem* using the concept of multiple $minSup$ and $maxPer$ constraints [33, 51]. In those studies, each item in the database is specified with a *minimum item support* ($minIS$) and *maximum item periodicity* ($maxIP$). Next, the $minSup$ and $maxPer$ of a pattern are specified depending on its items $minIS$ and $maxIP$ values, respectively. Although this concept facilitates every pattern to satisfy a different $minSup$ and $maxPer$ values, it still suffers from an open problem of determining the items' $minIS$ and $maxIP$ values.

Uday et al. [33] extended Liu's model [38] to address the *rare item problem* in periodic-frequent pattern mining. In this model, every item $i_j \in I$ in the database is specified with $minIS$ and $maxIP$ values. The $minSup$ and $maxPer$ for a pattern $X \subseteq I$ are specified as follows:

$$\begin{aligned} minSup(X) &= min(minIS(i_j)|\forall i_j \in X) \\ &and \\ maxPer(X) &= max(maxIP(i_j)|\forall i_j \in X) \end{aligned} \tag{4.1}$$

where, $minSup(X)$ represents the *minimum support* of X, $maxPer(X)$ represents the *maximum periodicity* of X, $minIS(i_j)$ denotes the *minimum item support* of an item $i_j \in X$ and $maxIP(i_j)$ denotes the *maximum item periodicity* of an item $i_j \in X$.

The usage of item-specific $minIS$ and $maxIP$ values facilitates every pattern to satisfy a different $minSup$ and $maxPer$ depending on its items. However, the major limitation of this model is the computational cost because the generated periodic-frequent patterns do not satisfy the *anti-monotonic property*. [51] proposed alternative periodic-frequent pattern using the item-specific $minIS$ and $maxIP$ values. In this model, the $minSup$ and $maxPer$ for a pattern X are specified as follows:

$$\begin{aligned} minSup(X) &= max(minIS(i_j)|\forall i_j \in X) \\ &and \\ maxPer(X) &= min(maxIP(i_j)|\forall i_j \in X) \end{aligned} \tag{4.2}$$

The periodic-frequent patterns discovered by this model satisfy the anti-monotonic property. Henceforth, this model is practicable in real-world applications.

An open problem that is common to above two studies [33, 51] is the methodology to specify items' $minIS$ and $maxIP$ values. [33] have described the following methodology to address this problem:

$$\begin{aligned} minIS(i_j) &= max(\gamma \times S(i_j),\ LS) \\ &\quad and \\ maxIP(i_j) &= max(\beta \times S(i_j) + Per_{max},\ Per_{min}) \end{aligned} \tag{4.3}$$

where $i \in I$ and $S(i)$ is the *support* of the item i. In Equation 4.3, LS is the user-specified lowest *minimum item support* allowed and $\gamma \in [0, 1]$ is a parameter that controls how the $minIS$ values for items should be related to their *supports*. In Equation 4.3, Per_{max} and Per_{min} are the user-specified maximum and minimum *periodicities* such that $Per_{max} \geq Per_{min}$ and $\beta \in [-1, 0]$ is a user-specified constant.

Although Equation 4.3 facilitates every item to have different $minIS$ and $maxIP$ values, it suffers from the following limitations: (i) This methodology requires several input parameters from the user. (ii) Equation 4.3 determines the $maxIP$ of an item by taking into account only its $support$. As a result, this equation implicitly assumes that all items having the same $support$ will also have similar $periodicities$ in a temporal database. However, this is seldom the case as items with similar $support$ can have different $periodicities$. We have observed that employing this methodology to specify items' $maxIP$ values in the temporal databases, where items can have similar $support$ but different $periodicities$ can still lead to the *rare item problem*.

Example 4 *Consider a hypothetical transactional database containing 100 transactions. Let 'x' and 'y' be two items in the database having the same support (say, $sup(x) = sup(y) = 40$), but different periodicities (say, $per(x) = 11$ and $per(y) = 30$). Since Equation 4.3 determines the $maxIP$ values by taking into account only the support of the items, both 'x' and 'y' will be assigned a common $maxIP$ value although their actual* periodicity *is different from one another. This can result either in missing interesting patterns or generating too many patterns. For instance, if we set $\beta = -0.5$, $Per_{min} = 10$ and $Per_{max} = 50$, then $maxIP(x) = maxIP(y) = 20$. In this case, we miss the periodic-frequent patterns containing 'y' because $per(y) \not\leq maxIP(y)$. In order to find the periodic-frequent patterns containing both 'x' and 'y' items, we have to set a high β value. When β is set at -0.375, we derive $maxIP(x) = maxIP(y) = 35$. In this case, we find periodic-frequent patterns containing 'y' because $per(y) \leq maxIP(y)$. However, we may also witness too many patterns containing the item 'x' because its $maxIP$ value is three times higher than its $periodicity$.*

4.3 Basic Idea

To solve the *rare item problem*, we propose a model to discover periodic-correlated patterns in a temporal database. In this thesis, we consider temporal database as a collection of transactions ordered by their timestamps. Further, a temporal database facilitates multiple transactions to share a common timestamp and time gaps in-between consecutive transaction. Temporal data is naturally produced in many real-world situations. For instance, the data produced by social networking applications, such as Twitter and Facebook, represents a temporal database. For example, Table 4.1 shows a part of a temporal database generated from the tweets produced during the Great East Japan Earthquake (GEJE), which occurred on the 11^{th} March 2011.

Considering the input data as a temporal database addresses the first obstacle in periodic-frequent pattern model. Three key properties of a temporal database are:

- All transactions are ordered by their timestamps
- Time gaps exist in-between consecutive transactions
- Multiple transactions share a common timestamp.

These three properties differentiate a temporal database from a widely studied transactional database, which basically represents a collection of transactions. Temporal data is naturally produced in many real-world situations. For instance, the data produced by social networking applications, such as Twitter and Facebook, represents a temporal database. For example, Table 4.1 shows a part of a temporal database generated from the tweets produced during the Great East Japan Earthquake (GEJE), which occurred on the 11^{th} March 2011.

Table 4.1 Some tweets produced during GEJE

timestamp	Tweets
1301575750	#Discrimination n demagoguery 4 foreigners, mainly #Asians by #Japanese in . #Sendai #earthquake #jishin http://htn.to/rhGtmd
1301575750	shhhhh dont tell @jimcramer El-Erian says recent Japanese earthquake is NOT like Kobe, won't have same V shaped recovery - Reuters #newsmkr
1301583131	Some people will never be able to go home. They lost their homes in the tsunami. #VOAAsiachat1
1301583579	Social media had a critical role in Japan during/after the quake/tsunami. Someone else can better assess overall than me. #VOAAsiachat1

Please note that although a temporal database can be transformed into a transactional database by merging its transactions that share a common timestamp, we should avoid such process as finding partial periodic itemsets in such databases leads to the following problems:

- **Type-I error.** Merging the transactions that share a common timestamp can result in losing the actual $support$ of an itemset. This may result in missing an interesting itemset from being discovered as a partial periodic itemset.

  Example 5 *Let two transactions in a market-basket data, say {Bread, Jam} and {Bread, Milk}, happened at a same timestamp (say, 9:00 hours). If we merge these two transactions into a single transaction, say* $\{Bread, Jam, Milk\}$*, then we will lose the actual support of* $Bread$ *in the entire data. This may result in missing the partial periodic itemsets involving the item* $Bread$*.*

- **Type-II error.** Merging the transactions sharing a common timestamp results in the creation of false correlations (or associations) between the items, and thus may generate an uninteresting itemset as a partial periodic itemset.

  Example 6 *Continuing with the previous example, merging the transactions that share a common timestamp induces an incorrect correlation between the items* Jam *and* $Milk$*. This may generate an uninteresting itemset* $\{Jam, Milk\}$ *as a partial periodic itemset.*

In the literature, correlated pattern model was discussed to address the *rare item problem* in frequent pattern mining [41]. We extend this model to find periodic-correlated patterns in a temporal database.

The proposed model considers a pattern as interesting if it satisfies the following two conditions: (i) if the $support$ of a pattern is close to the $support$ of its individual items, and (ii) if the $periodicity$ of a pattern is close to the $periodicity$ of its individual items. The renowned *all-confidence* [41] measure is used to determine how close is the $support$ of a pattern with respect to the $support$ of its individual items. To the best of our knowledge, there exists no measure to determine how close is the $periodicity$ of a pattern with respect to the $periodicity$ of its items. So forth, we introduce a new measure, called *periodic-all-confidence*, to determine the interestingness of a pattern. The *periodic-all-confidence* measure is used to determine how close is the $periodicity$ of a pattern with respect to the $periodicity$ of its individual items. These two measures facilitate us to achieve the objective of generating periodic-correlated patterns containing both frequent and rare items yet without causing frequent items to generate too many uninteresting patterns.

A pattern-growth algorithm, called Extended Periodic-Correlated pattern-growth (EPCP-growth), has also been described to find all periodic-correlated patterns. We evaluate the performance of EPCP-growth by conducting extensive experiments on both synthetic and real-world databases. Experimental results demonstrate that the proposed model can tackle the rare item problem and EPCP-growth is runtime efficient and highly scalable as well.

4.4 Modified Periodic-Frequent Pattern Model

In this section, we modify the periodic-frequent pattern model [54] by taking into account the temporal databases. Care has been taken such that the nomenclature of redefined model is consistent with the nomenclature of the basic model of periodic-frequent patterns.

Let I be the set of items. Let $X \subseteq I$ be a **pattern** (or an itemset). A pattern containing β number of items is called a β**-pattern**. A **transaction**, $t_k = (tid, ts, Y)$ is a tuple, where $tid \in \mathbb{R}$ represents transactional-identifier, $ts \in \mathbb{R}$ represents the timestamp at which the pattern Y has occurred. A **temporal database** TDB over I is a set of transactions, $TDB = \{t_1, \cdots, t_m\}$, $m = |TDB|$, where $|TDB|$ can be defined as the number of transactions in TDB. Let ts_{min} and ts_{max} denote the minimum and maximum timestamps in TDB, respectively. For a transaction $t_k = (tid, ts, Y)$, $k \geq$1, such that $X \subseteq Y$, it is said that X occurs in t_k and such timestamp is denoted as ts^X. Let $TS^X = \{ts^X_j, \cdots, ts^X_k\}$, $j, k \in [1, m]$ and $j \leq k$, be an **ordered set of timestamps** where X has occurred in TDB. We call this list of timestamps of X as **ts-list** of X. The number of transactions containing X in TDB is defined as the **support** of X and denoted as $sup(X)$. That is, $sup(X) = |TS^X|$. Let ts^X_q and ts^X_r, $j \leq q < r \leq k$, be the two consecutive timestamps in TS^X. The time difference between ts^X_r and ts^X_q is defined as a ***inter-arrival time*** of X, say p^X_a. That is, $p^X_a = ts^X_r - ts^X_q$. Let $P^X = (p^X_1, p^X_2, \cdots, p^X_r)$ be the set of all *inter-arrival times* for pattern X. The ***periodicity*** of X, denoted as $per(X) = max(p^X_1, p^X_2, \cdots, p^X_r)$. The pattern X is a **frequent pattern** if $sup(X) \geq minSup$, where $minSup$ refers to the user-specified *minimum support* constraint. The frequent pattern X is said to be **periodic-frequent** if $per(X) \leq maxPer$, where $maxPer$ refers to the user-specified *maximum periodicity* constraint. The redefined **problem definition** of periodic-frequent pattern mining involves discovering all patterns in TDB that satisfy the user-specified $minSup$ and $maxPer$ constraints. The $support$ of a pattern can be expressed in percentage of $|TDB|$. Similarly, the $period$ and $periodicity$ of a pattern can be expressed in percentage of $(ts_{max} - ts_{min})$.

Example 7 *Table 4.2 shows the temporal database with the set of items $I = \{a, b, c, d, e, f, g, h\}$. The set of items 'a' and 'b,' i.e., 'ab' is a pattern. This pattern contains only two items. Therefore, this is a 2-pattern. In the first transaction, $t_1 = (101, 1, ab)$, 101 (from the left hand side) represents the transaction identifier of the transaction, 1 denotes the timestamp at which the transaction has occurred and 'ab' represents the itemset occurring in this transaction. In the entire database, this pattern appears at the timestamps of $1, 2, 5, 7$ and 10. Therefore, $TS^{ab} = \{1, 2, 5, 7, 10\}$. The* support *of 'ab,' i.e., $sup(ab) = |TS^{ab}| = |1, 2, 5, 7, 10| = 5$. If the user-specified $minSup = 5$, then 'ab' is a frequent pattern because $sup(ab) \geq minSup$. The minimum and maximum timestamps of all transactions in this database are 1 and 12, respectively. Therefore, $ts_{min} = 1$ and $ts_{max} = 12$. All inter-arrival times for this pattern are: $p^{ab}_1 = 0$ $(= 1 - ts_{min})$, $p^{ab}_2 = 1$ $(= 2 - 1)$, $p^{ab}_3 = 3$ $(= 5 - 2)$, $p^{ab}_4 = 2$ $(= 7 - 5)$, $p^{ab}_5 = 3$ $(10 - 7)$ and $p^{ab}_6 = 2$ $(= ts_{max} - 10)$. Therefore, $P^{ab} = (0, 1, 3, 2, 3, 2)$. The $periodicity$*

Table 4.2 Running example: A temporal database

tid	ts	Items	tid	ts	Items
101	1	a, b	107	7	a, b, c, e
102	3	a, b, d	108	8	c, d
103	3	c, d, g	109	9	c, d
104	4	c, e, f	110	10	a, b, e, f
105	5	a, b	111	11	c, d, g
106	7	h	112	12	a, e, f

Table 4.3 Periodic-frequent patterns discovered from Table 4.2. The terms Pat, sup, $allConf$, per and $perAllConf$ refer to *pattern*, *support*, *all-confidence*, *periodicity* and *periodic-all-confidence*, respectively. The columns titled **I**, **II** and **III** represent the periodic-frequent patterns generated using basic model, extending *all-confidence* to the basic model and the proposed model, respectively.

Pat	sup	allConf	per	perAllConf	I	II	III
a	6	1	3	1	✓	✓	✓
b	5	1	3	1	✓	✓	✓
c	6	1	3	1	✓	✓	✓
d	5	1	5	1	✓	✓	✓
e	4	1	4	1	✓	✓	✓
f	3	1	6	1	✓	✓	✓
ab	5	0.833	3	1	✓	✓	✓
ef	3	0.75	6	1.5	✓	✓	✓
ce	2	0.4	5	1.67	✓	×	×
cd	4	0.8	5	1.67	✓	✓	×

of 'ab,' i.e., $per(ab) = max(0, 1, 3, 2, 3, 2) = 3$*. If the user-defined* $maxPer = 3$*, then the frequent pattern 'ab' is a periodic-frequent pattern because* $per(ab) \leq maxPer$*. (In our model, the first period of a pattern is calculated using* ts_{min}*, where as the first period of a pattern in Tanbeer's model [54] is calculated with reference to initial time, which is zero.)*

The above redefined model still suffers from the *rare item problem* (refer Example 8). In the next section, we propose a novel model to address this problem.

Example 8 *Consider the rare items 'e' and 'f' in Table 4.2. If we set a high* $minSup$ *and a short* $maxPer$*, say* $minSup = 5$ *and* $maxPer = 3$*, we will miss the periodic-frequent patterns containing these rare items. In order to discover the periodic-frequent patterns containing these rare items, we have to set a low* $minSup$ *and a long* $maxPer$*, say* $minSup = 2$ *and* $maxPer = 6$*. All periodic-frequent patterns discovered at these threshold values are shown in the column titled* ***I*** *in Table 4.3. It can be observed from this table that setting a low* $minSup$ *and a long* $maxPer$ *has not only resulted in finding* $'ef'$ *as a periodic-frequent pattern, but also resulted in generating the uninteresting patterns* $'ce'$ *and*

'cd' as periodic-frequent patterns. The pattern 'ce' is uninteresting (with respect to support *dimension), because the rare item 'e' is randomly occurring with a frequent item 'c' in very few transactions. The pattern 'cd' is uninteresting (with respect to* periodicity *dimension), because it contains the frequent items 'c' and 'd' appearing together at very long inter-arrival times (or $periodicity$).*

4.5 Proposed Model

To address the *rare item problem*, we need an approach that extracts interesting periodic-frequent patterns involving both frequent and rare items yet filtering out uninteresting patterns. After conducting the initial investigation on the nature of interesting patterns found in various databases, we have made a key observation that most of the interesting periodic-frequent patterns discovered in a database have their *support* and *periodicity* close to that of its individual items. The following example illustrates our observation.

Example 9 *In a supermarket, cheap and perishable goods (e.g., bread and butter) are purchased more frequently and periodically than the costly and durable goods (e.g., bed and pillow). Among all the possible combinations of the above four items, we normally consider* {*bread, butter*} *and* {*bed, pillow*} *as interesting patterns, because only these two patterns generally have* support *and* periodicity *close to the* support *and* periodicity *of its individual items. All other uninteresting patterns,* {*bread, bed*}, {*bread, pillow*}, {*butter, bed*} *and* {*butter, pillow*}*, generally have* support *and* periodicity *relatively far away from the* support *and* periodicity *of its individual items as compared against the above two patterns.*

Henceforth, in this chapter we consider a pattern as interesting if its *support* and *periodicity* are close to the *support* and *periodicity* of its individual items. In this context, we need two measures to determine the interestingness of a pattern with respect to both $support$ and $periodicity$ dimensions.

In the literature, researchers have discussed several measures to address the *rare item problem* in *support* dimension [53, 58]. In this chapter, we use *all-confidence* to address the *rare item problem* in support dimension. (The reason for choosing this measure for finding periodic-correlated patterns has been described in Section 2).

Continuing with the basic model of periodic-frequent patterns (discussed in the previous section), the proposed model of periodic-correlated patterns is as follows.

Definition 1 ***(All-confidence of X)*** *The all-confidence of X, denoted as $allConf(X)$, is the ratio of $support$ of X to the maximal support of an item $i_j \in X$. That is, $allConf(X) = \frac{sup(X)}{max(sup(i_j)|\forall i_j \in X)}$.*

For a pattern X, $allConf(X) \in (0, 1]$. As per the *all-confidence* measure, a pattern is interesting in *support* dimension if its *support* is close to the *support* of all of its items. The parameter *minAllConf* indicates the user-specified minimum *all-confidence* threshold value. Based on *minSup* and *minAllConf* thresholds, all the interesting patterns involving rare items in *support* dimension are extracted.

Definition 2 *(**Correlated pattern X**) The pattern X is said to be correlated if $sup(X) \geq minSup$ and $allConf(X) \geq minAllConf$. The terms minSup and minAllConf, respectively represent the user-specified minimum support and minimum all-confidence.*

Example 10 *In Table 4.2, the $support$ of the patterns a, b and ab are 6, 5 and 5, respectively. Therefore, the* all-confidence *of ab, i.e., $allConf(ab) = \frac{sup(ab)}{max(sup(a),sup(b))} = \frac{5}{max(6,5)} = 0.833$. If the user-specified $minSup = 2$ and $minAllConf = 0.6$, then ab is a correlated pattern because $sup(ab) \geq minSup$ and $allConf(ab) \geq minAllConf$.*

The usage of *all-confidence* alone is insufficient to completely address the *rare item problem* in periodic-frequent pattern mining. The reason is this measure does not take into account the $periodicity$ dimension of a pattern.

Example 11 *The column titled **II** in Table 4.3 shows the periodic-frequent patterns discovered when* all-confidence *is used along with $support$ and $periodicity$ measures. The* minSup*,* minAllConf *and* maxPer *values used to find these patterns are 2, 0.6 and 6, respectively. It can be observed from the discovered periodic-frequent patterns that though* all-confidence *is able to prune the uninteresting pattern 'ce,' it has failed to prune another uninteresting pattern 'cd' from the list of periodic-frequent patterns generated by the basic model. Henceforth, the* rare item problem *has to be addressed with respect to both $support$ and $periodicity$ dimensions.*

As there exists no measure in the literature that determines the interestingness of a pattern with respect to the $periodicities$ of all of its items, we propose a new measure, ***periodic-all-confidence***, to extract interesting patterns in *periodicity* dimension involving rare items, which is defined as follows.

Definition 3 *(**Periodic-all-confidence of X**) The periodic-all-confidence of X, denoted as $perAllConf(X)$, is the ratio of $periodicity$ of X to the minimal periodicity of an item $i_j \in X$. That is, $perAllConf(X) = \frac{per(X)}{min(per(i_j)|\forall i_j \in X)}$.*

Example 12 *In Table 4.2, the $periodicity$ of the patterns a, b and ab are 3, 3 and 3, respectively. Therefore, the* periodic-all-confidence *of ab, i.e., $perAllConf(ab) = \frac{per(ab)}{min(per(a),per(b))} = \frac{3}{min(3,3)} = 1$.*

For a pattern X, $perConf(X) \in [1, \infty)$. As per the *periodic-all-confidence* measure, a pattern is interesting in *periodicity* dimension, if the *periodicity* of a pattern is close to the *periodicity* of all of its items. The parameter *maxPerAllConf* indicates the maximum *periodic-all-confidence* threshold set by the user. Based on *maxPer* and *maxPerAllConf* thresholds, the interesting patterns involving rare items in *periodicity* dimension can be extracted.

Henceforth, the periodic-correlated pattern is defined as follows.

Definition 4 ***(Periodic-correlated pattern X)*** *The pattern X is said to be periodic-correlated if $sup(X) \geq minSup$, $allConf(X) \geq minAllConf$, $per(X) \leq maxPer$ and $perAllConf(X) \leq maxPerAllConf$. The terms minSup, minAllConf, maxPer and maxPerAllConf, respectively represent the user-specified minimum support, minimum all-confidence, maximum periodicity and maximum periodic-all-confidence.*

Example 13 *If the user-specified $minSup = 2$, $minAllConf = 0.6$, $maxPer = 6$ and $maxPerAllConf = 1.5$, then the pattern 'ab' is said to be a periodic-correlated pattern, because $sup(ab) \geq minSup$, $allConf(ab) \geq minAllConf$, $per(ab) \leq maxPer$ and $perAllConf(ab) \leq maxPerAllConf$.*

Example 14 *The column titled **III** in Table 4.3 shows the complete set of periodic-correlated patterns discovered from Table 4.2. It can be observed that the proposed model has not only discovered the periodic-correlated patterns containing rare items but also pruned the uninteresting patterns 'cd' and 'ce.'* This clearly demonstrates that the proposed model discovers periodic-correlated patterns containing rare items without generating too many uninteresting patterns.

The discovered periodic-correlated patterns satisfy the *anti-monotonic property*. The correctness is straightforward to prove from Properties 1 and 2. (see Lemma 1).

Property 1 *If $X \subset Y$, then $TS^X \supseteq TS^Y$. Therefore, $sup(X) \geq sup(Y)$ and $allConf(X) \geq allConf(Y)$.*

Property 2 *If $X \subset Y$, then $per(X) \leq per(Y)$. Therefore, $perAllConf(X) \leq perAllConf(Y)$ as $\frac{per(X)}{min(per(i_j) \forall i_j \in X)} \leq \frac{per(Y)}{min(per(i_j) \forall i_j \in Y)}$.*

Lemma 1 *If $X \subset Y$, then $TS^X \supseteq TS^Y$. Therefore, $sup(X) \geq sup(Y)$, $allConf(X) \geq allConf(Y)$, $per(X) \leq per(Y)$ and $perAllConf(X) \leq perAllConf(Y)$.*

Proof 1 *If $X \subset Y$, then $TS^X \supseteq TS^Y$, i.e.*

$$\Rightarrow |TS^X| \geq |TS^Y|$$
$$\Rightarrow sup(X) \geq sup(Y)$$
$$\Rightarrow maximum(sup(i); \forall i \in X) \leq maximum(sup(i); \forall i \in Y)$$
$$\Rightarrow \frac{sup(X)}{maximum(sup(i); \forall i \in X)} \geq \frac{sup(Y)}{maximum(sup(i); \forall i \in Y)}$$
$$\Rightarrow allconf(X) \geq allConf(Y).$$

Similarly, it can be proved for $per(X) \leq per(Y)$ *and* $perAllConf(X) \leq perAllConf(Y)$.

Definition 5 ***Problem Definition:*** *Given the temporal database* (TDB) *and the user-specified minimum support (minSup), minimum all-confidence (minAllConf), maximum periodicity (maxPer) and maximum periodic-all-confidence (maxPerAllConf), the problem of finding periodic-correlated patterns involve discovering all patterns that satisfy the minSup, minAllConf, maxPer and maxPerAllConf thresholds. The support of a pattern can be expressed in percentage of* $|TDB|$. *The periodicity of a pattern can be expressed in percentage of* $(ts_{max} - ts_{min})$.

Please note that, since we are using *confidence*-based measures to prune the uninteresting patterns, we name the discovered patterns as *periodic-correlated* patterns instead of *periodic-frequent* patterns.

4.6 Proposed Algorithm

Tanbeer et al. [54] have proposed Periodic-Frequent pattern-growth (PF-growth) to discover periodic-frequent patterns using $support$ and *periodicity* measures. Unfortunately, this algorithm cannot be directly used for finding the periodic-correlated patterns with our model. The reason is PF-growth does not determine the interestingness of a pattern using *all-confidence* and *periodic-all-confidence* measures. In this chapter, we extend PF-growth to determine the interestingness of a pattern using these two measures. We call the proposed algorithm as Extended Periodic-Correlated pattern-growth (EPCP-growth). The proposed algorithm involves two steps: (i) construction of Extended Periodic-Correlated pattern-tree (EPCP-tree), (ii) recursively mining EPCP-tree to discover periodic-correlated patterns. Before we describe these two steps, we explain the structure of EPCP-tree.

4.6.1 Structure of EPCP-tree

The structure of EPCP-tree consists of a prefix-tree and a EPCP-list. The EPCP-list consists of three fields: item name (*i*), $support$ (*s*) and $periodicity$ (*p*). The structure of prefix-tree in EPCP-tree is similar to that of the prefix-tree in FP-tree [26]. However, to obtain both $support$ and $periodicity$ of

the patterns, the nodes in EPCP-tree explicitly maintain the occurrence information for each transaction by maintaining an occurrence timestamp list, called $ts\text{-}list$, only at the tail node of every transaction. Complete details on prefix-tree are available in [54].

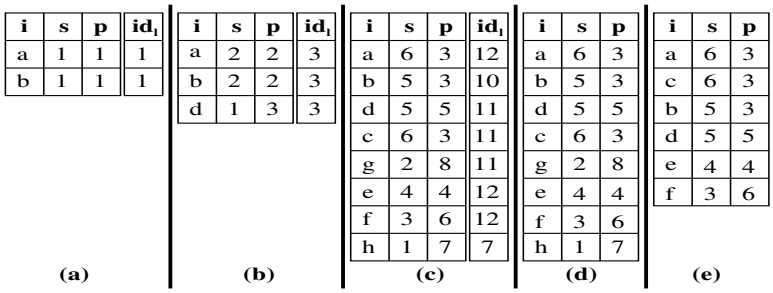
One can assume that the structure of the prefix-tree in an EPCP-tree may not be memory efficient since it explicitly maintains timestamps of each transaction. However, it has been argued that such a tree can achieve memory efficiency by keeping transaction information only at the tail nodes and avoiding the support count field at each node [54].

Algorithm 7 Construction of EPCP-tree (TDB: Temporal database, $minSup$: minimum support, $minAllConf$: minimum all-confidence, $maxPer$: maximum periodicity, $maxPerAllConf$: maximum periodic-all-confidence)

1: Let id_l be a temporary array that records the ts of the last appearance of each item in the TDB. Let $t = \{ts_{cur}, X\}$ denote the current transaction with ts_{cur} and X representing the timestamp of the current transaction and pattern, respectively.
2: **for** each transaction $t \in TDB$ **do**
3: **if** an item i occurs for the first time **then**
4: Insert i into the EPCP-list with $sup^i = 1$, $per^i = ts_{min} - ts_{cur}$ and $id^i_l = 1$.
5: **else**
6: $sup^i = sup^i + 1$.
7: **if** $(ts_{cur} - id^i_l) > per^i$ **then**
8: $per^i = ts_{cur} - id^i_l$.
9: **end if**
10: **end if**
11: **end for**
12: **for** each item i in EPCP-list **do**
13: **if** $(ts_{max} - id^i_l) > per^i$ **then**
14: $per^i = ts_{max} - id^i_l$.
15: **end if**
16: **end for**
17: Remove items from the EPCP-list that do not satisfy $minSup$ and $maxPer$.
18: Sort the remaining items in EPCP-list in descending order of their $support$. Let this sorted list of items be $EPCP$.
19: Create a root node in EPCP-tree, T, and label it "$null$."
20: **for** each transaction $t \in TDB$ **do**
21: Sort the items in t in $EPCP$ order. Let this list of sorted periodic-frequent items in t be $[p|P]$, where p is the first item and P is the remaining list.
22: Call $insert\_tree([p|P], ts_{cur}, T)$.
23: **end for**

4.6.2 Construction of EPCP-tree

Since the periodic-correlated patterns generated by the proposed model satisfy the *anti-monotonic property*, periodic-correlated items (or 1-patterns) play a key role in efficient discovery of higher or-



i	s	p	id_l
a	1	1	1
b	1	1	1

(a)

i	s	p	id_l
a	2	2	3
b	2	2	3
d	1	3	3

(b)

i	s	p	id_l
a	6	3	12
b	5	3	10
d	5	5	11
c	6	3	11
g	2	8	11
e	4	4	12
f	3	6	12
h	1	7	7

(c)

i	s	p
a	6	3
b	5	3
d	5	5
c	6	3
g	2	8
e	4	4
f	3	6
h	1	7

(d)

i	s	p
a	6	3
c	6	3
b	5	3
d	5	5
e	4	4
f	3	6

(e)

Figure 4.2 Construction of EPCP-list for Table 4.2. (a) After scanning the first transaction (b) After scanning the second transaction (c) After scanning every transaction (d) Updated EPCP-list (e) Final EPCP-list with sorted list of periodic-correlated items

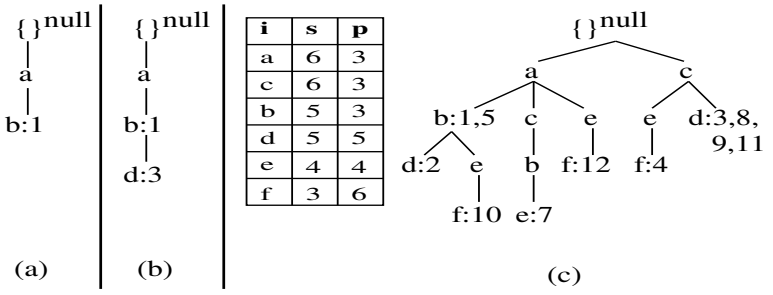


Figure 4.3 Construction of EPCP-tree for Table 4.2. (a) After scanning first transaction (b) After scanning second transaction (c) After scanning every transaction

der periodic-correlated patterns. Periodic-correlated items are discovered by populating the EPCP-list (lines 1 to 18 in Algorithm 7). Figures 4.2 (a), (b), (c), (d) and (e) show the steps involved in finding periodic-correlated items from EPCP-list. The user-specified $minSup$, $minAllConf$, $maxPer$ and $maxPerAllConf$ values are 2, 0.6, 6 and 1.5, respectively.

After finding periodic-correlated items, prefix-tree is constructed by performing another scan on the database (lines 19 to 23 in Algorithm 7). The construction of prefix-tree in EPCP-tree is similar to the construction of prefix-tree in FP-tree [26]. However, it has to be noted that leaf nodes in EPCP-tree maintain the transaction-ids of the transactions. Figures 4.3 (a), (b) and (c) show the construction of EPCP-tree after scanning first, second and every transaction in the temporal database, respectively. In EPCP-tree, an item header table is built so that each item points to its occurrences in the tree via a chain of node-links, to facilitate tree traversal. For simplicity, we do not show these node-links in trees, however, they are maintained as in FP-tree.

Algorithm 8 Insert_tree($[p|P]$, ts_{cur}, T)

1: **if** T does not have a child N satisfying $p.itemName = N.itemName$ **then**
2: Create a new node N and set its parent as T. Let its node-link be linked to the nodes with the same item via the node-link structure.
3: **end if**
4: Remove p from P.
5: **if** P is non-empty **then**
6: Call Insert_tree($[P]$, ts_{cur}, N)
7: **else**
8: Add ts_{cur} to T (i.e., leaf node).
9: **end if**

The EPCP-tree maintains the complete information of all periodic-correlated patterns in a database. The correctness is based on Property 3 and shown in Lemmas 2 and 3. For each transaction $t \in TDB$, EPCP(t) is the set of all candidate items in t, i.e., $EPCP(t) = item(t) \cap EPCP$, and is called the candidate item projection of t.

Property 3 *An EPCP-tree maintains a complete set of candidate item projections for each transaction in a database only once.*

Lemma 2 *Given a TDB and user-defined $minSup$, $minAllConf$, $maxPer$, and $maxPerAllConf$ thresholds, the complete set of all periodic-correlated item projections of all transactions in the TDB can be derived from the EPCP-tree.*

Proof 2 *Based on Property 3, each transaction $t \in TDB$ is mapped to only one path in the tree, and any path from the root up to a tail node maintains the complete projection for exactly n transactions (where n is the total number of entries in the ts-list of the tail node).*

Lemma 3 *The size of the EPCP-tree (without the root node) on a TDB for user-defined $minSup$, $minAllConf$, $maxPer$, and $maxPerAllConf$ thresholds, is bounded by $\sum_{t \in TDB} |EPCP(t)|$.*

Proof 3 *According to the EPCP-tree construction process and Lemma 2, each transaction t contributes at most one path of size $|EPCP(t)|$ to an EPCP-tree. Therefore, the total size contribution of all transactions can be $\sum_{t \in TDB} |EPCP(t)|$ at best. However, since there are usually many common prefix patterns among the transactions, the size of an EPCP-tree is normally much smaller than $\sum_{t \in TDB} |EPCP(t)|$.*

Before we discuss the mining of EPCP-tree, we explore the following important property and lemma of an EPCP-tree.

Property 4 *A tail node in an EPCP-tree maintains the occurrence information for all the nodes in the path (from the tail node to the root) at least in the transactions in its ts-list.*

Lemma 4 *Let $Z = \{a_1, a_2, \cdots, a_n\}$ be a path in an EPCP-tree where node a_n is the tail node carrying the ts-list of the path. If the ts-list is pushed-up to node a_{n1}, then a_{n1} maintains the occurrence information of the path $Z' = \{a_1, a_2, \cdots, a_{n1}\}$ for the same set of transactions in the ts-list without any loss.*

Proof 4 *Based on Property 4, a_n maintains the occurrence information of path Z' at least in the transactions in its ts-list. Therefore, the same ts-list at node a_{n1} maintains the same transaction information for Z' without any loss.*

4.6.3 Mining EPCP-tree

Algorithm 9 describes the procedure for mining periodic-correlated patterns from EPCP-tree. The EPCP-tree is mined by calling EPCP-growth as (EPCP-tree, *null*). This algorithm resembles FP-growth. However, the key difference is that once the pattern-growth is achieved for a suffix 1-pattern (or item), it is completely pruned from the EPCP-tree by pushing its ts-list to respective parent nodes.

The working of this algorithm is as follows. We proceed to construct the prefix tree for each candidate item in the EPCP-list, starting from the bottom most item, say i. To construct the prefix-tree for i, the prefix sub-paths of node i are accumulated in a tree-structure, PT_i. Since i is the bottom-most item in the EPCP-list, each node labeled i in the EPCP-tree must be a tail node. While constructing PT_i, based on Property 4, we map the ts-list of every node of i to all items in the respective path explicitly in the temporary array (one for each item). This temporary array facilitates the calculation of sup, $allConf$, per, $perAllConf$ of each item in PT_i (line 2 in Algorithm 9). If an item j in PT_i has $sup \geq minSup$, $allConf \geq minAllConf$, $per \leq maxPer$ and $perAllConf \leq maxPerAllConf$, then we construct

Table 4.4 Mining EPCP-tree by creating conditional (sub -) pattern bases

Item	sup	per	Cond. Pattern Base	Cond. EPCP-tree	Per. Freq. Patterns
f	3	6	$\{ce:4\}$, $\{abe:10\}$, $\{ae:12\}$	$\langle e:4,10,12\rangle$	$\{ef$: 3, 0.75, 6, 1.5$\}$
e	4	4	$\{c:4\}$, $\{abc:7\}$, $\{ab:10\}$, $\{a:12\}$	–	–
d	5	5	$\{ab:3\}$, $\{c:3,8,9,11\}$	–	–
b	5	3	$\{a:1,2,5,10\}$, $\{ac:7\}$	$\langle a:1,2,5,7,10\rangle$	{ab: 5, 0.833, 3, 1}
c	6	3	$\{a:7\}$	–	–

its conditional tree and mine it recursively to discover the recurring patterns (lines 3 to 9 in Algorithm 9). Moreover, to enable the construction of the prefix-tree for the next item in the EPCP-list, based on Lemma 4, the ts-lists are pushed-up to the respective parent nodes in the original EPCP-tree and in PT_i as well. All nodes of i in the original EPCP-tree and is entry in the EPCP-list are deleted thereafter (line 10 in Algorithm 9).

Using Properties 3 and 4, the conditional tree CT_i for PT_i is constructed by removing all those items from PT_i that have $sup \leq minSup$, or $allConf \leq minAllConf$, or $per \geq maxPer$ or $perAllConf \geq maxPerAllConf$. If the deleted node is a tail node, its ts-list is pushed-up to its parent node. The contents of the temporary array for the bottom item j in the EPCP-list of CT_i represent TS_{ij} (i.e., the set of all timestamps where items i and j have appeared together in the database). The same process of creating a prefix-tree and its corresponding conditional tree is repeated for further extensions of "ij". The whole process of mining for each item is repeated until EPCP-list $\neq \emptyset$.

Table 4.4 summarizes the working of this algorithm. First, we consider item 'f,' which is the bottom-most item in the EPCP-list, as a suffix pattern. This item appears in three branches of the EPCP-tree (refer Figure 4.3(c)). The paths formed by these branches are $\{cef:4\}$, $\{abef:10\}$ and $\{aef:12\}$ (format of these branches is $\{nodes:timestamps\}$). Therefore, considering 'f' as a suffix item, its corresponding three prefix paths are $\{ce:4\}$, $\{abe:10\}$ and $\{ae:12\}$, which form its conditional pattern base (refer Figure 4.4(a)). Its conditional EPCP-tree contains only a single path, $\langle e:4,10,12\rangle$; 'a,' 'b' and 'c' are not included because their *all-confidence* and *periodic-all-confidence* do not satisfy the $minAllConf$ and $maxPerAllConf$ respectively. Figure 4.4(b) shows the conditional EPCP-tree of 'f.' The single path generates the pattern $\{ef:3,0.75,6,1.5\}$ (format is {*pattern*: $support$, *all-confidence*, $periodicity$, *periodic-all-confidence*}). The same process of creating prefix-tree and its corresponding conditional tree is repeated for further extensions of 'ef.' Next, 'f' is pruned from the original EPCP-tree and its ts-lists are pushed to its parent nodes, as shown in Figure 4.4(c). All the above processes are once again repeated until EPCP-list $\neq \emptyset$.

Algorithm 9 EPCP-growth($Tree$, α)

1: **for** each a_i in the header of $Tree$ **do**
2: Generate pattern $\beta = a_i \cup \alpha$. Construct an array TS^{β}, which represents the set of timestamps at which β has appeared in TDB. Next, compute from TS^{β}, $sup(\beta)$, $allConf(\beta)$, $per(\beta)$ and $perAllConf(\beta)$ and compare them with $minSup$, $minAllConf$, $maxPer$ and $maxPerAllConf$, respectively.
3: **if** $sup(\beta) \geq minSup$, $allConf(\beta) \geq minAllConf$, $per(\beta) \leq maxPer$ and $perAllConf(\beta) \leq maxPerAllConf$ **then**
4: Output β as a periodic-correlated pattern as {β: sup, allConf, per, perAllConf}.
5: Traverse $Tree$ using the node-links of β, and construct β's conditional pattern base and β's conditional EPCP-tree $Tree_{\beta}$.
6: **if** $Tree_{\beta} \neq \emptyset$ **then**
7: call EPCP-growth($Tree_{\beta}$, β);
8: **end if**
9: **end if**
10: Remove a_i from the $Tree$ and push a_i's ts-list to its parent nodes.
11: **end for**

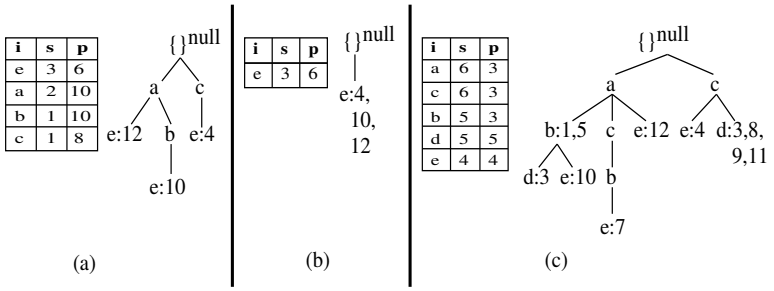


Figure 4.4 Mining of EPCP-tree for Table 4.2. (a) Prefix-tree of suffix item '*f*,' i.e., PT_f (b) Conditional tree of suffix item '*f*,' i.e., CT_f (c) EPCP-tree after pruning item '*f*.'

4.7 Experimental Results

In this section, we show that the proposed model discovers interesting patterns pertaining to both frequent and rare items by pruning uninteresting patterns. We also evaluate the proposed model against the existing models of periodic-correlated patterns [33, 51, 54].

The algorithms, *PF-growth*, *MCPF-growth*, *MaxCPF-growth* and *EPCP-growth* are written in C++ and run with Fedora 22 on a 2.66 GHz machine with 8 GB of memory. We have conducted experiments using both synthetic (**T10I4D100K**) and real-world (**Retail** and **FAA-accidents**) databases. The T10I4D100K data-base is generated using the IBM data generator [3]. This database contains 878 items with 100,000 transactions. The **Retail** database contains the market basket data from a Belgian retail store. This database contains 16,471 items with 88,162 transactions. The **FAA-accidents** database is constructed from the accidents data recorded by FAA from 1-January-1978 to 31-December-2014. This database contains 9,290 items with 98,864 transactions.

4.7.1 Patterns Generated by the Proposed Model

Figure 4.5 (a)-(c) shows the number of patterns generated at different *minAllConf* and *maxPerAllConf* values in T10I4D100K, Retail and FAA-Accidents databases. *The* $minSup$ *and* $maxPer$ *are set at* 0.01% *and* 40%. The following observations can be drawn: (i) The increase in $minAllConf$ results in decrease of periodic-correlated patterns. The reason is that as $minAllConf$ increases, the $support$ threshold value of a pattern increases. (ii) The increase in $maxPerAllConf$ results in increase of patterns. The reason is that as *maxPerAllConf* increases, the *periodicity* threshold value of a pattern increases.

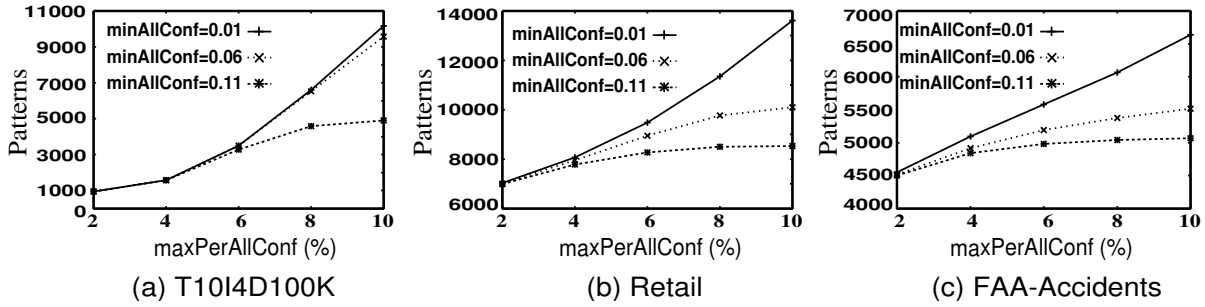


Figure 4.5 Periodic-correlated patterns generated at different *maxAllConf* and *maxPerAllConf* values

Figure 4.6 show the runtime requirements of EPCP-growth at different *maxPerAllConf* and *minAllConf* values in T10I4D100K, Retail and FAA-Accidents databases. The changes in *minAllConf* and *maxPerAllConf* threshold values shows a similar effect on runtime requirement as in the generation of periodic-correlated patterns.

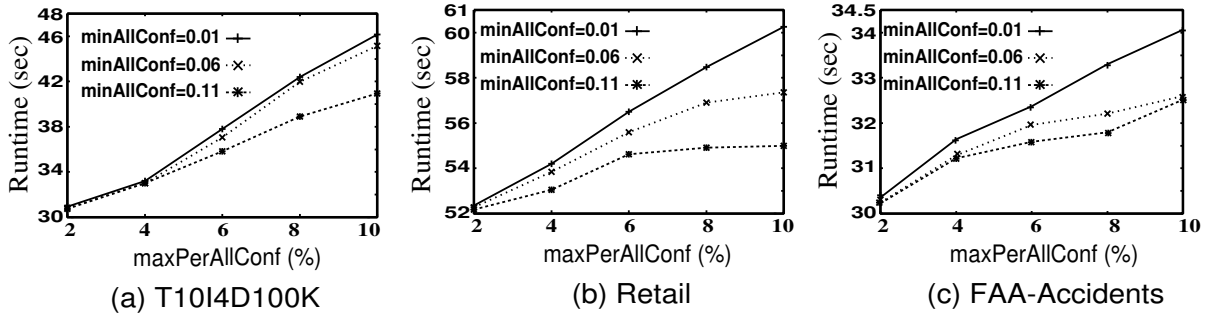


Figure 4.6 Runtime requirements of EPCP-growth at different *maxAllConf* and *maxPerAllConf* values

4.7.2 A case study: evaluation of periodic patterns discovered from FAA-Accidents data

Table 4.5 shows some of the interesting patterns discovered in FAA database. The $minSup$, *minAllConf*, $maxPer$ and *maxPerAllConf* values used are 0.01%, 0.01, 40% and 9, respectively. *It can be*

Table 4.5 Some of the interesting patterns discovered in FAA-accidents database

S. No.	Patterns	sup	allConf	per	perAllConf
1	{Pilot Not Certificated, Destroyed}	13	0.06	4756	7.77
2	{Student, Substantial}	136	0.02	893	29.77
3	{Boeing, Substantial}	214	0.02	214	26.35
4	{Private-Pilot, Cessna, CE-172, Minor}	1,661	0.03	117	23.4
5	{General Operating Rules, Commercial Pilot, Minor}	10,399	0.15	32	6.4

observed from their $support$ values that our model has discovered interesting patterns involving both frequent and rare items effectively. Please note that the *periodicity* (per) is expressed in days.

The first pattern in this table reveals interesting information that 13 aircrafts have been 'destroyed' when piloted by a non-certified pilot. The $periodicity$ of this event is 4756 days (≈13 years). The second pattern indicates 136 aircrafts driven by student pilots have suffered substantial damages at least once in every ≈2.5 years. The third pattern indicates that Boeing aircrafts have suffered substantial damages at least once in every ≈7 months. The fourth pattern reveals the information that Cessna airlines CE-172 driven by private pilots have encountered minor damages at least once in every ≈4 months. The last pattern reveals the information that at least once in every 32 days, an aircraft driven by commercial pilots has witnessed minor damages during general operating rules.

4.7.3 Comparison of Proposed Model against the Existing Models

For *MCPF-growth* and *MaxCPF-growth*, we use Equation 4.3 to specify items' $minIS$ and $maxIP$ values. Setting the α and β values in this equation has been a non-trivial task as the patterns discovered by these algorithms can be different from the patterns discovered by *EPCP-growth*. After conducting several experiments, we have empirically set the following values for *MCPF-growth* and *MaxCPF-growth* algorithms, such that both algorithms discover almost all periodic-frequent patterns discovered by *EPCP-growth*.

Figure 4.7 shows the number of periodic-frequent patterns generated at different $minSup$ values (Y-$axis$ is plotted on logscale). For *EPCP-growth*, we have fixed $minAllConf = 0.01$, $maxPer = 40\%$ and $maxPerAllConf = 9$ and vary $minSup$ values. For *MCPF-growth* and *MaxCPF-growth*, we have set $\gamma = 0.01$, $LS = minSup$, $\beta = -0.4$, $Per_{max} = 40\%$ and $Per_{min} = 10\%$. For *PF-growth*, we have set $maxPer = 40\%$ and vary $minSup$ values.

Figure 4.8 shows the number of periodic-frequent patterns generated at different $maxPer$ values (Y-$axis$ is plotted on logscale). For *EPCP-growth*, we have fixed $minSup = 0.01\%$, $minAllConf = 0.01$ and $maxPerAllConf = 9$ and vary $maxPer$ values. For *MCPF-growth* and *MaxCPF-growth*, we have set $\gamma = 0.01$, $LS = 0.01\%$, $\beta = -0.4$, $Per_{max} = maxPer$ and $Per_{min} = 10\%$. For *PF-growth*, we have set $minSup = 0.01\%$ and vary $maxPer$ values.

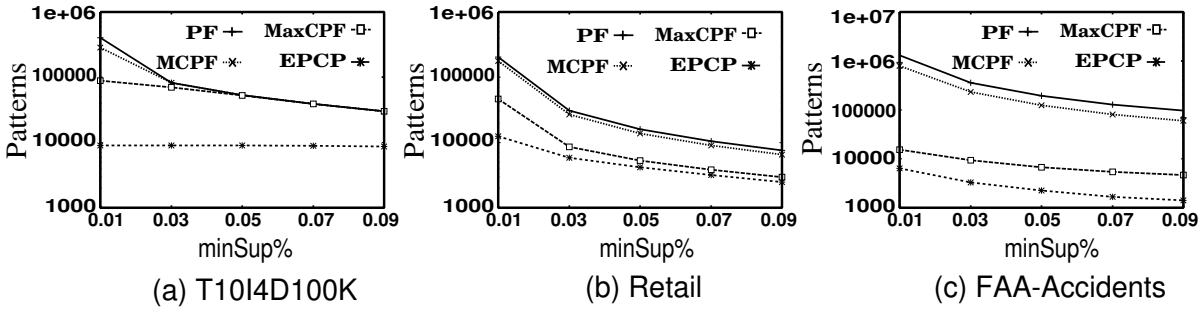


Figure 4.7 Periodic-Correlated patterns generated at different minSup values

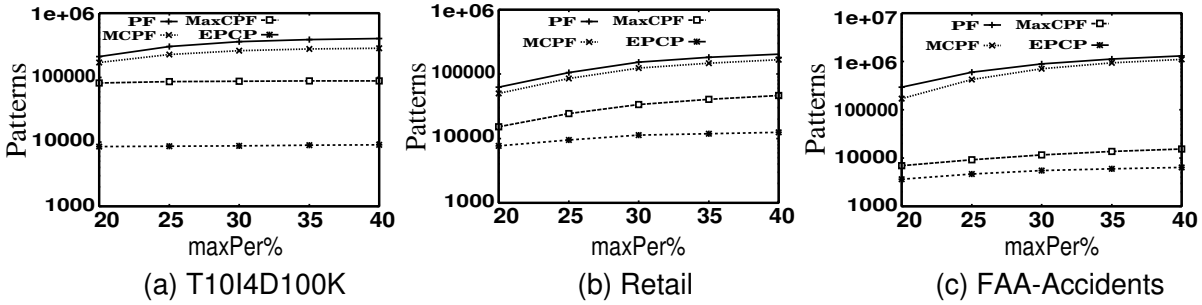


Figure 4.8 Periodic-Correlated patterns generated at different maxPer values

From Figures 4.7 and 4.8, it can be observed that the proposed model has generated lesser number of periodic-frequent patterns than the other models, because the existing models have suffered from the *rare item problem*.

Figures 4.9 shows the runtime taken by various models at different $minSup$ values (Y-$axis$ is plotted on logscale). It can be observed that, in all the databases the proposed model takes lesser runtime to find periodic-frequent patterns than *PF-growth* and *MCPF-growth*. But the proposed model takes slightly more runtime than *MaxCPF-growth*. So the proposed model is not adding any significant overhead in mining periodic-correlated patterns.

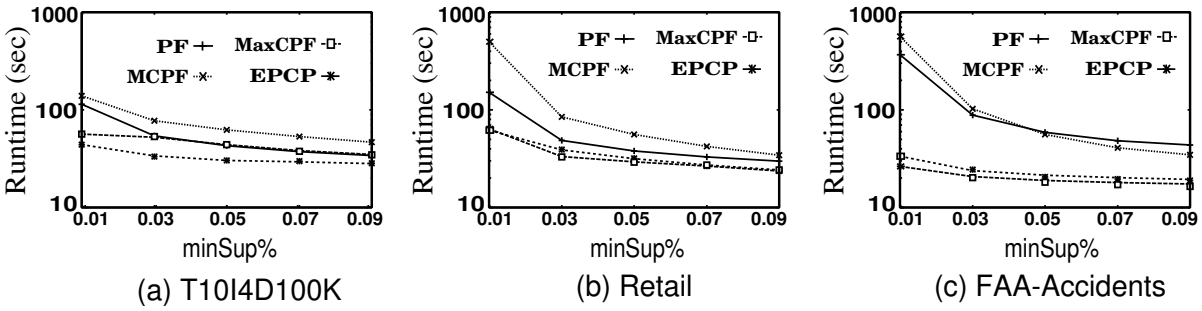


Figure 4.9 Runtime requirements of various models at different minSup values

Figure 4.10 shows the runtime taken by various models at different $maxPer$ values ($Y\text{-}axis$ is plotted on logscale). Similar observations to that of varying $minSup$ can be drawn.

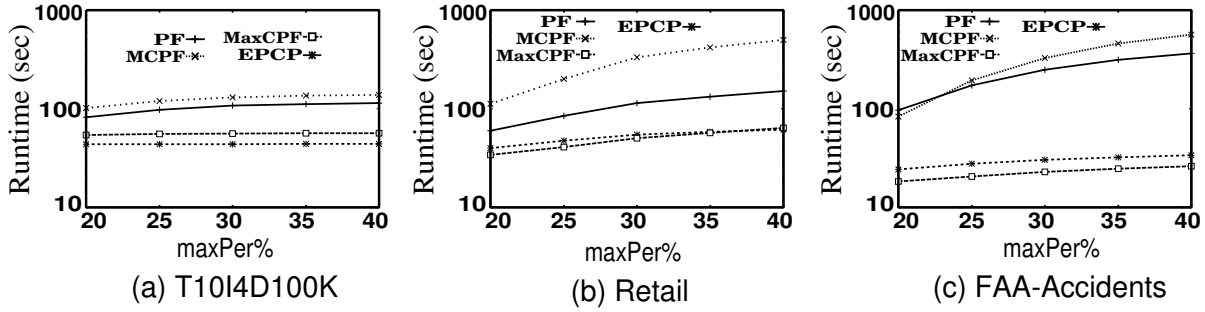


Figure 4.10 Runtime requirements of various models at different maxPer values

4.7.4 Scalability

We studied the scalability of EPCP-growth on execution time by varying the number of transactions in a database. We used Kosarak, T10I4D1000K and T25I6D1000K datasets for this experiment. We divided the database into five equal parts, i.e., 20% transactions in each part. Then we investigated the performance of EPCP-growth by accumulating each part with previous parts and running EPCP-growth each time. Figure 4.11 (a), (b) and (c) show the graph of runtime requirements of EPCP-growth in Kosarak, T10I4D1000K and T25I6D1000K databases, respectively. It is clear from the graphs that as the database size increases, overall tree construction and mining time increase. However, it shows stable performance of about linear increase in runtime with respect to the database size.

Figure 4.12 (a), (b) and (c) show the graph of memory requirements of EPCP-growth in Kosarak, T10I4D1000K and T25I6D1000K databases, respectively. Similar observations to that of runtime requirements can be drawn. Therefore, it can be observed from the scalability test that EPCP-growth can mine periodic-correlated patterns in large databases with considerable amount of runtime and memory.

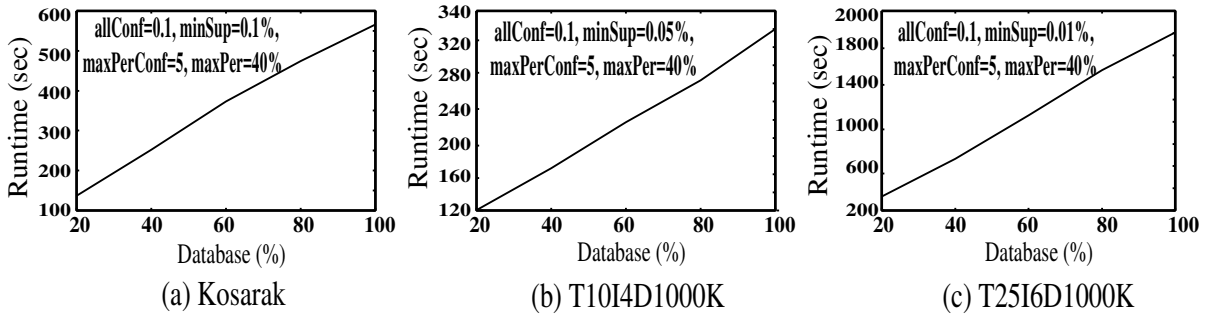
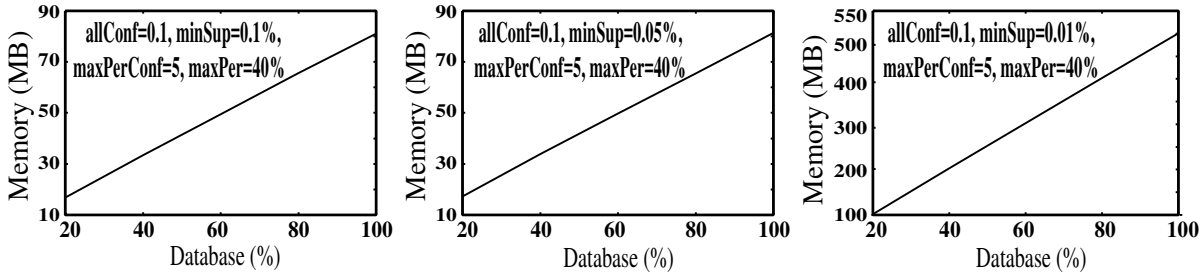


Figure 4.11 Runtime requirements of *EPCP*-growth in various databases



(a) Kosarak (b) T10I4D1000K (c) T25I6D1000K

Figure 4.12 Memory requirements of *EPCP*-growth in various databases

4.8 Summary

This chapter introduces a model to address the rare item problem in both *support* and *periodicity* dimensions. A new interestingness measure, *periodic-all-confidence*, is proposed to address the problem in *periodicity* dimension. An efficient pattern-growth algorithm has been proposed to discover all periodic-correlated patterns in a database. Experimental results demonstrate that the proposed model is efficient. We discussed the usefulness of periodic-correlated patterns with a real-world case study and showed that the proposed model may be utilized to discover interesting periodic-correlated patterns involving both frequent and rare items from FAA-Accidents database effectively.

Chapter 5

Discovering Partial Periodic-Frequent Patterns in Non-Uniform Temporal Databases

In this chapter, we have proposed a novel model to tackle *rare item problem* as well as to find partial periodic-frequent patterns in temporal databases. The proposed model enables every pattern to satisfy different $period$ and minimum number of cyclic repetitions depending on its items. A novel measure *relative periodic-support*, is being proposed to determine the partial periodic interestingness of a pattern in a database.

The organization of the chapter is as follows. Section 5.1 explains the motivation to the existing problem in periodic pattern mining. Section 5.2 introduces the basic idea we are proposing to solve the problem. Section 5.3 introduces the proposed model of finding periodic-correlated patterns in a temporal database. Section 5.4 describes EPCP-growth algorithm. Section 5.5 reports on experimental results. Section 5.6 makes a comparison between the model proposed in this chapter and earlier chapter (Chapter 4). Finally, section 5.7 concludes the chapter with summary.

5.1 Motivation

Temporal databases are commonly used in many domains. A temporal database is a collection of transactions, ordered by their timestamps. A temporal database is said to be non-uniform if it contains items with dissimilar $support$ and $periodicity$. Non-uniform temporal data is naturally produced in many real-world situations. For instance, disasters such as earthquakes and tsunami happen at irregular time intervals. Twitter data related to these disasters is thus non-uniform. We discussed about temporal databases in detail in Section 4.3.

Current periodic-frequent pattern models have focused on discovering full periodic-frequent patterns, i.e., finding all patterns that have exhibited complete cyclic repetitions throughout the entire database. These models evaluate the periodic interestingness of a frequent pattern by determining whether all of its inter-arrival times are within the user-specified $maxPer$ threshold. Therefore, the model cannot assess the partial periodic behavior of a frequent pattern in a database. However, partial periodic-frequent

patterns are more common due to the imperfect nature of real-world databases. Discovering partial periodic-frequent patterns in temporal databases has numerous applications.

Partial periodic-frequent patterns exhibit partial cyclic repetitions in a database. Partial periodic-frequent patterns are a looser kind of full periodic-frequent patterns, and they exist ubiquitously in the real-world databases. Finding partial periodic-frequent patterns is thus useful to understand the data. For example, it was revealed in our present study on Twitter data related to the GEJE that over 80% of the event keywords found by a supervised event detection algorithm [47] can also be discovered as partial periodic-frequent patterns. The proposed study thus may be used as an unsupervised learning technique to generate some prior knowledge about event keywords and their associations in Twitter data. There is no existing work which has considered mining partial periodic-frequent patterns in non-uniform temporal databases, where items have non-uniform *support* and *periodicity*, despite that this type of data is very common in real-life.

5.2 Basic Idea

In this chapter, we have introduced a novel model to discover partial periodic-frequent patterns in non-uniform temporal databases, which considers that patterns may have different *period* and minimum number of cyclic repetitions. The proposed model lets the user specify a different *maximum inter-arrival time* ($MIAT$) for each item. An inter-arrival time of a pattern is considered periodic (or cyclic) if it is no more than $period$. Thus, different patterns may satisfy different $period$ depending on their items' $MIAT$ values. This solves the *rare item problem* in partial periodic-frequent pattern mining. A new measure, *Relative Periodic-Support* (RPS), is proposed to determine the (partial) periodic interestingness of a pattern by considering the number of cyclic repetitions in the database. This measure assess the interestingness of a pattern by taking into account both the $support$ and $periodicity$ information of patterns.

This measure satisfies the *null-invariant property* [53]. Thus, the usage of item specific $MIAT$ values and RPS allows the proposed model to capture the non-uniform distribution of items in a database. We also propose a pattern-growth algorithm that discovers the complete set of partial periodic-frequent patterns. This measure also satisfies the *convertible anti-monotonic property* [43]. This property plays a key role in reducing the search space, which in turn decreases the computational cost of mining the patterns. Thus, enabling the proposed algorithm in efficient mining of partial periodic-frequent patterns.

5.3 Proposed Model - Discovering Partial Periodic-Frequent Patterns

Let $I = \{i_1, i_2, \cdots, i_n\}$ be the set of 'n' items appearing in a database. A set of items $X \subseteq I$ is called an itemset (or **a pattern**). A pattern containing k items is called a k-pattern. The length of this pattern is k. A transaction is a triplet $tr = (tid, ts, Y)$, where tid represents the transactional identifier, $ts \in \mathbb{R}$ represents the transaction time (or timestamp) and Y is an itemset. A temporal database TDB

Table 5.1 Running example: temporal database

tid	ts	items
100	1	ab
101	3	$acdg$
102	3	$abef$
103	4	cd
104	6	$abcd$
105	7	efg
106	8	$abcd$
107	9	ce
108	10	$abef$
109	11	abf
110	12	$abcd$
111	14	$acdeg$

is an ordered set of transactions, i.e. $TDB = \{tr_1, tr_2, \cdots, tr_m\}$, where $m = |TDB|$ represents the database size (the number of transactions). Let ts_{min} and ts_{max} denote the minimum and maximum timestamps in TDB, respectively. For a transaction $tr = (tid, ts, Y)$, such that $X \subseteq Y$, it is said that X occurs in tr and such a timestamp is denoted as ts^X. Let $TS^X = (ts^X_a, ts^X_b, \cdots, ts^X_c)$, $a \leq b \leq c$, be the **ordered list of timestamps** of transactions in which X appears in TDB. The number of transactions containing X in TDB (i.e., the size of TS^X) is defined as the $support$ of X and denoted as $sup(X)$. That is, $sup(X) = |TS^X|$.

Example 15 *Table 5.1 shows a temporal database with $I = \{abcdefg\}$. The set of items 'a' and 'b,' i.e., 'ab' is a pattern. This pattern contains 2 items. Therefore, it is a 2-pattern. The length of this pattern is 2. In the first transaction, $tr_1 = (100, 1, ab)$, '100' represents the tid of the transaction, '1' represents the timestamp of this transaction and 'ab' represents the items occurring in this transaction. Other transactions in this database follow the same representation. The size of the database is $m = 12$. The minimum and maximum timestamps in this database are 1 and 14, respectively. Therefore, $ts_{min} = 1$ and $ts_{max} = 14$. The pattern 'ab' appears in the transactions whose timestamps are 1, 3, 6, 8, 10, 11 and 12. Therefore, $TS^{ab} = \{1, 3, 6, 8, 10, 11, 12\}$. The $support$ of 'ab,' i.e., $sup(ab) = |TS^{ab}| = 7$.*

Definition 6 ***(Period of a pattern X)*** *Let $MIAT(i_j)$ be the user-defined* maximum inter-arrival time *$(MIAT)$ specified for an item $i_j \in I$. The $period$ of a pattern X, denoted as $PER(X)$, represents the largest $MIAT$ value of all items in X. That is, $PER(X) = max(MIAT(i_j) | \forall i_j \in X)$. The items' $MIAT$ values can also be expressed in percentage of $(ts_{max} - ts_{min})$.*

Example 16 *Let the $MIAT$ values for the items a, b, c, d, e, f and g be 2, 2, 2, 2, 3, 4 and 4, respectively. The $period$ of the pattern 'ab,' i.e., $PER(ab) = max(2, 2) = 2$.*

The usage of items' $MIAT$ values enable us to achieve the goal of having lower $periods$ for patterns that only involve frequent items, and having higher $periods$ for patterns that involve rare items. The items' $MIAT$ values may be derived using the $period$ determining functions, such as Fast Fourier Transformations (FFTs) and auto-correlation.

Definition 7 ***(Periodic occurrence of a pattern X)*** *Let ts^X_j, $ts^X_k \in TS^X$, $1 \leq j < k \leq m$, denote any two consecutive timestamps in TS^X. The time difference between ts^X_k and ts^X_j is referred*

*as an **inter-arrival time** of X, and denoted as iat^X. That is, $iat^X = ts^X_k - ts^X_j$. Let $IAT^X = \{iat^X_1, iat^X_2, \cdots, iat^X_k\}$, $k = sup(X) - 1$, be the list of all inter-arrival times of X in TDB. An inter-arrival time of X is said to be **periodic** (or cyclic) if it is no more than $PER(X)$. That is, a $iat^X_i \in IAT^X$ is said to be **periodic** if $iat^X_i \leq PER(X)$.*

Example 17 *The pattern 'ab' has initially appeared at the timestamps of 1 and 3. The difference between these two timestamps gives an inter-arrival time of 'ab.' That is, $iat^{ab}_1 = 2\ (= 3 - 1)$. Similarly, other inter-arrival times of 'ab' are $iat^{ab}_2 = 3\ (= 6 - 3)$, $iat^{ab}_3 = 2\ (= 8 - 6)$, $iat^{ab}_4 = 2\ (= 10 - 8)$, $iat^{ab}_5 = 1\ (= 11 - 10)$ and $iat^{ab}_6 = 1\ (= 12 - 11)$. Therefore, $IAT^{ab} = \{2, 3, 2, 2, 1, 1\}$. If $PER(ab) = 2$, then iat^{ab}_1, iat^{ab}_3, iat^{ab}_4, iat^{ab}_5 and iat^{ab}_6 are considered as the periodic occurrences of 'ab'. The iat^{ab}_2 is considered as an aperiodic occurrence of 'ab' because $iat^{ab}_2 \not\leq PER(ab)$.*

Definition 8 ***(Relative periodic-support of a pattern X)** Let $\widehat{IAT^X}$ be the set of all inter-arrival times in IAT^X that have $iat^X \leq PER(X)$. That is, $\widehat{IAT^X} \subseteq IAT^X$ such that if $\exists iat^X_k \in IAT^X : iat^X_k \leq PER(X)$, then $iat^X_k \in \widehat{IAT^X}$. The relative periodic-support of X, denoted as $RPS(X) = \frac{|\widehat{IAT^X}|}{|IAT^{i_j}|}$, where i_j is an item that has the lowest $support$ and maximum $MIAT$ value among all items in X. This measure satisfies the null-invariant property [53].*

Example 18 *Continuing with the previous example, $\widehat{IAT^{ab}} = \{2, 2, 2, 1, 1\}$, the item 'b' in the pattern 'ab' has the lowest $support$ and maximum $MIAT$ value. Therefore, the* relative periodic-support *of 'ab,' i.e., $RPS(ab) = \frac{|\widehat{IAT^{ab}}|}{|IAT^b|} = \frac{5}{6} = 0.83$.*

For brevity, we call $\widehat{IAT^X}$ as ***periodic-frequency***. The *periodic-frequency* determines the number of cyclic repetitions of a pattern in the data. Thus, the proposed measure assess the interestingness of a pattern by taking into account both the $support$ and $periodicity$ information of patterns. The proposed measure enables us to achieve the goal of specifying a higher number of cyclic repetitions for patterns that only involve frequent items, and a lower number of cyclic repetitions for patterns that involve rare items. Hence, the proposed model tackles *rare item problem.*

For a pattern X, $RPS(X) \in [0, 1]$. If all inter-arrival times of X are more than $PER(X)$, then $RPS(X) = 0$. In other words, X is an irregular pattern. If all inter-arrival times of X are within $PER(X)$, then $RPS(X) = 1$. In other words, X is a full periodic-frequent pattern.

In the proposed model, we have considered an inter-arrival time of X as interesting if $iat^X \leq PER(X)$. However, our model is flexible and allows other ways to consider an inter-arrival time of a pattern as interesting. For instance, we can consider an inter-arrival time of a pattern as interesting if $iat^X \leq PER(X) \pm \Omega$, where $\Omega > 1$ is a constant that denotes time tolerance. We stick to the above definition for brevity.

Definition 9 *(**Partial Periodic-Frequent pattern X**) The pattern X is a partial periodic-frequent pattern if $RPS(X) \geq minRPS$, where $minRPS$ is the user-specified minimum relative periodic-support.*

Example 19 *Continuing with the previous example, if the user-specified $minRPS = 0.6$, then 'ab' is a partial periodic-frequent pattern because $RPS(ab) \geq minRPS$.*

Definition 10 *(**Problem definition**) Given a temporal database (TDB), set of items (I), user-defined minimum interval times of the items ($MIAT$) and minimum relative periodic-support ($minRPS$), the problem of finding partial periodic-frequent patterns involve discovering all patterns in TDB that have* relative periodic-support *no less than $minRPS$.*

The partial periodic-frequent patterns generated by the proposed model satisfy the **convertible anti-monotonic property** [43].

Property 5 *Let $Z = \{i_1, i_2, \cdots, i_k\}$, $1 \leq k \leq |I|$, be a pattern with $MIAT(i_1) \geq MIAT(i_2) \geq MIAT(i_k)$. If $Y \subset Z$ and $i_1 \in Y$, then $RPS(Y) \geq RPS(Z)$ as $\frac{|\widehat{IAT^Y}|}{|IAT^{i_1}|} \geq \frac{|\widehat{IAT^Z}|}{|IAT^{i_1}|}$.*

5.4 Partial Periodic-Frequent Pattern-Growth Algorithm

In this section, we describe the proposed PP-growth algorithm that discovers the complete set of partial periodic-frequent patterns. Our algorithm involves the following two steps: (i) compress the database into a partial periodic-frequent pattern tree (PP-tree) and (ii) recursively mine the PP-tree to find all partial periodic-frequent patterns. Before we discuss these two steps, we describe the PP-tree structure.

5.4.1 Structure of PP-tree structure

A PP-tree has two components: a PP-list and a prefix-tree. The PP-list consists of each distinct *item* (i) with *minimum interval time* $(MIAT)$, *support* (S), *periodic-frequency* (PF) and a pointer pointing to the first node in the prefix-tree carrying the item. The prefix-tree in a PP-tree resembles that of the prefix-tree in a FP-tree [23]. However, to capture both $support$ and inter-arrival times of the patterns, the nodes in the PP-tree explicitly maintain the occurrence information for each transaction by keeping an occurrence timestamp list, called a ***ts*-list**. To achieve memory efficiency, only the last node of every transaction maintains the ts-list. We now explain the construction and mining of PP-tree.

Algorithm 10 Construction of PP-Tree(TDB: Temporal database, I: Set of items, $MIAT$: minimum interval time, $minRPS$: minimum relative periodic-support)

1: Insert all items in TDB into the PP-list with their $MIAT$ values. Set the $support$ and *periodic-frequency* values of all these items to 0. The $timestamps$ of the last occurring transactions of all items in the PP-list are explicitly recorded for each item in a temporary array, called ts_l.
2: Let $t = \{ts_{cur}, X\}$ denote the current transaction with ts_{cur} and X representing the timestamp and pattern, respectively.
3: **for** each transaction $t \in TDB$ **do**
4: **for** each item $i \in X$ **do**
5: $S(i) + +$;
6: **if** $((ts_l(i) \neq 0) \&\& (ts_{cur} - ts_l(i)) \leq MIAT(i))$ **then**
7: $PF(i) + +$;
8: **end if**
9: $ts_l(i) = ts_{cur}$;
10: **end for**
11: **end for**
12: All items in PP-list are sorted in ascending order of their $MIAT$ values. The items having a common $MIAT$ value are sorted in descending order of their $support$.
13: Measure the RPS value for the bottom most item in the PP-list. If the RPS value of this item is less than $minRPS$, then prune this item from the PP-list and repeat the same step for the next bottom most item in the PP-list. Stop this pruning process once the RPS value of the bottom most item in PP-list is no less than $minRPS$. Let CI denote this sorted list of items.
14: Create a root node in the prefix-tree, T, and label it as "$null$."
15: **for** each transaction $t \in TDB$ **do**
16: Sort the items in X according to the order of CI. Let the sorted candidate item list in t be $[p|P]$, where p is the first item and P is the remaining list. Call $insert\_tree([p|P], ts_{cur}, T)$, which is performed as follows. If T has a child N such that $N.item\text{-}name \neq p.item\text{-}name$, then create a new node N, Let its parent link be linked to T. Let its node-link be linked to nodes with the same $item\text{-}name$ via the node-link structure. Remove p from P. If P is nonempty, call $insert\_tree(P, ts_{cur}, N)$ recursively; else add ts_{cur} to the leaf node.
17: **end for**

Algorithm 11 PP-growth($Tree$, α)

1: **for** each a_i in the header of $Tree$ **do**
2: **if** $\frac{PF(a_i)}{S(a_i)-1} \geq minRPS$ **then**
3: Generate pattern $\beta = a_i \cup \alpha$. Traverse $Tree$ using the node-links of β, and construct an array, TS^{β}, which represents the timestamps at which β has appeared in TDB. Construct β's conditional pattern base and β's conditional PP-tree $Tree_{\beta}$ by calling calculateRPS(β, TS^{β}, $MIAT(a_i)$). The calculateRPS function calculates the *periodic-frequency* of β from TS^{β}, and returns RPS value by dividing the *periodic-frequency* with $S(a_i) - 1$.
4: **if** $Tree_{\beta} \neq \emptyset$ **then**
5: call PP-growth($Tree_{\beta}$, β);
6: **end if**
7: **end if**
8: Remove a_i from the $Tree$ and push the a_i's ts-list to its parent nodes.
9: **end for**

(a)

i	MIAT	S	PF	tsl
a	2	0	0	0
b	2	0	0	0
c	2	0	0	0
d	2	0	0	0
e	3	0	0	0
g	4	0	0	0
f	4	0	0	0

(b)

i	MIAT	S	PF	tsl
a	2	1	0	1
b	2	1	0	1
c	2	0	0	0
d	2	0	0	0
e	3	0	0	0
g	4	0	0	0
f	4	0	0	0

(c)

i	MIAT	S	PF	tsl
a	2	9	7	14
b	2	7	5	12
c	2	7	5	14
d	2	6	4	14
e	3	5	2	14
f	4	4	3	11
g	4	3	0	14

(d)

i	MIAT	S	PF
a	2	9	7
b	2	7	5
c	2	7	5
d	2	6	4
e	3	5	2
f	4	4	3
g	4	3	0

(e)

i	MIAT	S	PF
a	2	9	7
b	2	7	5
c	2	7	5
d	2	6	4
e	3	5	2
f	4	4	3

Figure 5.1 Construction of PP-List. (a) Before scanning the database (b) After scanning the first transaction (c) After scanning the entire database (d) Updated PP-list (e) Final PP-list with sorted list of items

5.4.2 Construction of PP-tree.

The procedure for constructing a PP-tree is shown in Algorithm 10. We illustrate the working of this algorithm using the database shown in Table 5.1. (**Please note that we ignore the *tid* information of transactions for brevity**).

For the construction of PP-list, we insert all items into the PP-list with their $MIAT$ values. The $support$ and *periodic-frequency* of all these items are simultaneously set to 0. Figure 5.1 (a) shows the PP-list generated before scanning the database (line 1 in Algorithm 10). The scan on the first transaction, "1:ab," updates the $support$ and ts_l values of a and b to 1 and 1, respectively. Figure 5.1 (b) shows the PP-list generated after scanning the first transaction. This process is repeated for other transactions in the database and PP-list is updated accordingly. Figure 5.1 (c) shows the PP-list generated after scanning the entire database (lines 2 to 8 in Algorithm 10). The items in PP-list are sorted in ascending order of their $MIAT$ values. Items having a common $MIAT$ value are sorted in descending order of their $support$ (to achieve memory efficiency). Figure 5.1 (d) shows the sorted PP-list (line 9 in Algorithm 10). We calculate RPS for the item 'g,' which is the bottom-most item in the PP-list. As $RPS(g) \ngeq minRPS$, the item 'g' is pruned from the PP-list. Next, we calculate the RPS value for the item f, which is

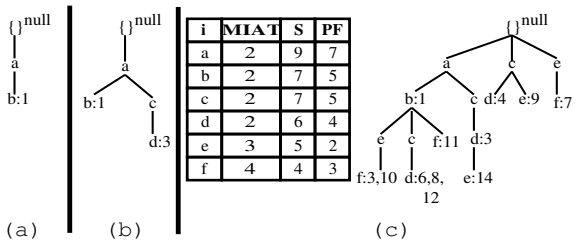


Figure 5.2 Construction of PP-Tree. (a) After scanning the first transaction (b) After scanning the second transaction (c) After scanning the entire database

the current bottom-most item in the PP-list. As $RPS(f) \geq minRPS$, we consider f as a periodic 1-pattern and stop the process of pruning other aperiodic 1-patterns from the PP-list. Figure 5.1 (e) shows the final PP-list after pruning some of the aperiodic 1-patterns whose supersets can never produce any periodic-frequent pattern (line 10 in Algorithm 10). Let CI denote the sorted list of items in PP-list. That is, $CI = \{a, b, c, d, e, f\}$. Next, we create a root node in the prefix-tree of PP-tree, and label it as "null" (line 11 in Algorithm 10).

In the next step, we update the PP-tree by performing another scan on the database. The items in the first transaction, "$1 : ab$," are sorted in CI order and a first branch is constructed with two nodes $\langle a \rangle$ and $\langle b : 1 \rangle$, where 'a' is linked as a child of the root and 'b' is linked as the child node of 'a'. As 'b' represents the leaf node of the first transaction, this node carries the timestamp of 1. Figure 5.2(a) shows the PP-tree updated after scanning the first transaction. This process is repeated for the remaining transactions in the database and the PP-tree is updated accordingly. Figure 5.2(b) shows the PP-tree generated after scanning the second transaction. Figure 5.2(c). shows the PP-tree generated after scanning the entire database (lines 12 and 13 in Algorithm 10).

5.4.3 Recursive mining of PP-tree.

The PP-tree is mined as follows. Start from length-1 pattern (as an initial suffix pattern). If the RPS value of this pattern satisfies the $minRPS$, then consider this pattern as a periodic item (or 1-pattern), construct its conditional pattern base (a sub-database, which consists of the set of prefix paths in the PP-tree with the suffix pattern), then construct its conditional PP-tree, and recursively mine that tree. Pattern-growth is achieved by concatenating the suffix pattern with the periodic-frequent patterns generated from a conditional PP-tree. Next, the initial suffix pattern is pruned from the original PP-tree by moving its ts-lists to the corresponding parent nodes.

Algorithm 11 describes the procedure for finding periodic-frequent patterns in a PP-tree. We do not discuss this algorithm in detail as it is straightforward to understand. Mining the PP-tree is summarized in Table 5.2. It can be observed that conditional pattern bases have not been constructed for the item 'e,' because it is an aperiodic 1-pattern with $RPS(e) \ngeq minRPS$. The above bottom-up mining technique is efficient, because it shrinks the search space dramatically as the mining process progresses. Some of the improvements discussed for FP-growth [23] can be straight forward extended to PP-growth.

Table 5.2 Mining the PP-tree by creating conditional (sub-)pattern bases

$item$	$support$	$MIAT$	Conditional Pattern Base	Conditional PP-tree	Partial Periodic-frequent patterns
f	4	4	$\{abe:3,10\}$, $\{ab:11\}$, $\{e:7\}$	$\langle e:3,7,10\rangle$	$\{ef:0.66\}$
e	5	3	–	–	–
d	6	2	$\{abc:6,8,12\}$, $\{ac:3,14\}$, $\{c:4\}$	$\langle c:3,4,6,8,12,14\rangle$	$\{cd:0.8\}$
c	7	2	$\{ab:6,8,12\}$, $\{a:3,14\}$	–	–
b	7	2	$\{a:1,3,6,8,10,11,12\}$	$\langle a:1,3,6,8,10,11,12\rangle$	$\{ab:0.83\}$

5.5 Experimental Results

In this section, we evaluate the proposed algorithm and show that our algorithm is memory and runtime efficient. We also show that PP-tree consumes less memory than the FP-tree for many databases. Finally, we discuss the usefulness of the proposed model by demonstrating that over 80% of the event keywords found by a supervised event detection system [47] in Twitter data can also be discovered as partial periodic-frequent patterns. (Similar to FP-growth [43, 23], PP-growth also scales linearly with the increase of database size).

The algorithms PP-growth and FP-growth are written in GNU C++ and run on a 2.66 GHz machine having 16 GB of memory. Ubuntu 14.04 is the operating system of our machine. The event detection system is written in python and java, and available for download at `https://github.com/aritter/twitter_nlp`. The experiments have been conducted using both synthetic (**T10I4D100K**) and real-world (**FAA-accidents** and **Twitter**) databases. The synthetic database, **T10I4D100K**, is generated by using the IBM data generator [3]. This data generator is widely used for evaluating association rule mining algorithms. The **T10I4D100K** database contains 870 items with 100,000 transactions. The **FAA-accidents** database is constructed from the accidents data recorded by the Federal Aviation Authority (FAA) from 1-January-1970 to 31-December-2014. Only categorical attributes have been taken into account while constructing the database. This database contains 9,290 items and 98,864 transactions. The Twitter database constitutes of 2,680,896 tweets collected from 10-march-2011 to 31-march-2011. These tweets are related to GEJE. We have created temporal database by considering top 4000 frequent english words.

Figure 5.3(a)-(c) present scatter plots about the inter-arrival times of items in the T10I4D100K, FAA-accidents and Twitter databases, respectively. The X-axis represents the items ranked in descending order of their $support$ and Y-axis represents the median of inter-arrival times of an item in a database. The thick line in these figures denote the trend line. The equations of these trend lines and R^2 values are shown in Table 5.3. It can be observed from the trend lines that rare items not only have low $support$, but also have high inter-arrival times as compared against the frequent items. This experiment clearly demonstrates the importance of enabling every pattern to satisfy a different $period$ and minimum number of cyclic repetitions to be a (partia)l periodic-frequent pattern.

The performance of PP-growth has to be evaluated by varying the items' $MIAT$ values. Unfortunately, popular $period$ identification functions (e.g. FFTs and auto-correlation) do not help us vary items' $MIAT$ values. In this context, we employ the following methodology to specify the items'

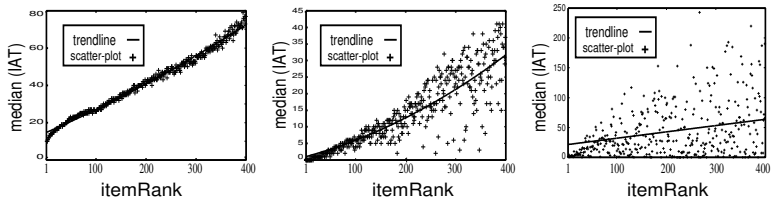


Figure 5.3 The median of inter-arrival times of items in a database

Table 5.3 Trend line Equations for various databases

Database	Equation of trend line ($f(x)$)	R^2
T10I4D100K	y = $7.06E$-$05x^2 + 0.12x + 14.70$	0.9892
FAA-Accidents	y = -$9.67E$-$05x^2 + 0.04x + 1$	0.9122
Twitter	y = $3.32E$-$06x^2 + 0.11x + 21.39$	0.0777

$MIAT$ values. For each database, we use the equation of trend line as a reference, and specify the items' $MIAT$ values by multiplying the equation of the trend line with a constant β. That is, $MIAT(i_j) = \beta \times f(x)$, where $\beta \geq 1$ is a user-specified constant and $f(x)$ is the equation of trend line in which x denotes the rank of an item in support descending order.

Figure 5.4(a)-(c) shows the number of partial periodic-frequent patterns generated for different $minRPS$ and β values in T10I4D100K, FAA-accidents and Twitter databases, respectively. The following two observations can be drawn from these figures: (i) Increase in β value may increase the number of partial periodic-frequent patterns. The reason is that higher β values tend to increase $MIAT$ values of items. (ii) Increase in $minRPS$ may decrease the number of periodic patterns. The reason is that increasing $minRPS$ increases the minimum number of cyclic repetitions necessary for a pattern to be a partial periodic-frequent pattern.

Figure 5.5(a)-(c) show the runtime requirements of PP-growth at different $minRPS$ and β values in T10I4D100K, FAA-accidents and Twitter databases, respectively. It can be observed that varying the β and $minRPS$ values has similar influence on runtime than on the generation of periodic-frequent patterns.

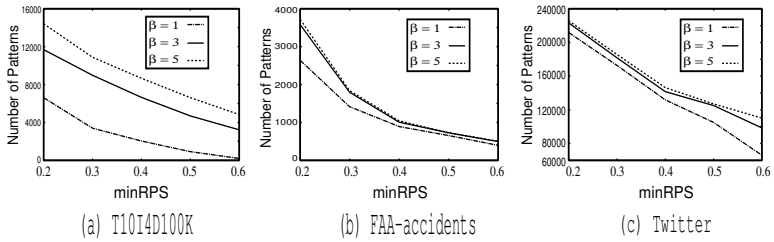


Figure 5.4 The partial periodic-frequent patterns generated at different $minRPS$ and β values

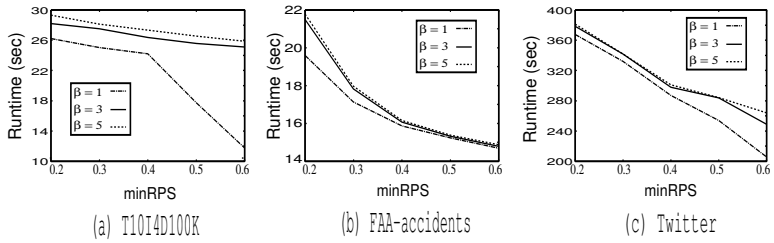


Figure 5.5 Runtime requirements of PP-growth at different $minRPS$ and β values

Table 5.4 Memory comparison of FP-tree and PP-tree

Data set	FP-tree (in MB)	PP-tree (in MB)	No. of Nodes
T10I4D100K	10.906	8.561	714,739
FAA-accidents	5.898	4.801	316,935
Twitter	7.172	15.606	470,040

Table 5.4 lists the maximum memory usage of PP-tree and FP-tree on T10I4100K, FAA-accidents and Twitter databases, respectively. Both trees are constructed with every item in the database. It can be observed from the results that PP-tree consumes less memory than FP-tree if number of nodes in a $tree$ exceed the database size, otherwise, PP-tree consumes more memory than FP-tree.

5.5.1 A case study: evaluation of partial periodic-frequent patterns discovered from Twitter data

While investigating the usefulness of partial periodic-frequent patterns discovered from Twitter data, we have observed that many generated periodic-frequent 1-patterns (and their associations) were interesting as they were referring to the event GEJE. This motivated us to study the following: (i) Do event keywords in Twitter exhibit periodic behavior? and (ii) If event keywords exhibit periodic behavior, then what would be their percentage? The significance of this study is that if we find many event key-

Table 5.5 Some of the interesting partial periodic-frequent patterns and tweets containing the patterns

Pattern	RPS	Tweets
still,death,alive,-rumors,celeb	0.83	S Yuko Yamaguchi (Hello Kitty) & Satoshi Tajiri (Pokemon) are still alive. Please stop spreading J-celeb death rumors. #earthquake
jishin,helpme,anpi,-hinan,nosg	0.86	twitter社より。統一のハッシュタグなどが発表になりました。情報の統合に協力しましょう。 #jishin: 地震一般に関する情報 #j_j_helpme :救助要請 #hinan :避難 #anpi :安否確認 #311care: 医療系被災者支援情報” #NOSG **(summary: users were tweeting the list of hashtags provided by Twitter for GEJE)**
tsunami,earthquake,-warning,massive,-widened	0.83	RT @bbcbreaking: #Tsunami warning is widened to incl rest of Pacific coast, incl #Australia and #South America massive #earthquake in #Japan

words exhibiting periodic behavior, then one can use the proposed model as an unsupervised learning technique to derive some prior knowledge about event keywords and their associations in Twitter data.

Ritter et al. [47] discussed a supervised learning model to discover event keywords from tweets. We use this model for our experiment. This model annotates tweets using natural language processing techniques, generates a model from the training set of tweets and uses the model to extract event keywords from the test set of tweets. As the authors have already trained their model to identify event keywords in tweets, we have simply provided our Twitter data as the test set and extracted event keywords. A total of 325 event keywords have been extracted from the Twitter data. (We found that only 106 event keywords have appeared in top 500 frequent words. This clearly demonstrates that $frequency$ has less influence in determining a word as an event keyword.) When we compared these event keywords against the periodic 1-patterns generated at $\beta = 3$ and $minRPS = 0.6$, we found that 267 event keywords have been generated as periodic 1-patterns. In other words, 82.15% (=$\frac{267 \times 100}{325}$) of keywords have exhibited periodic behavior in Twitter data. This clearly demonstrates that periodic pattern mining can be used to find prior knowledge about event keywords and their associations in Twitter data. Table 5.5 lists some of the generated periodic patterns and their associated tweets.

5.6 Discussions

In this section, we compare the *partial periodic frequent* model (proposed in this chapter) and *periodic-correlated* model (proposed in Chapter 4). As demonstrated from the experimental results in Section 4.7, the model of *periodic-correlated* pattern mining in temporal databases tackles the *rare item problem* effectively. However, this model does not mine partial periodic-frequent patterns. That is, frequent patterns with partial periodicity are not mined. Because, this model mines only those frequent patterns which are periodic throughout the database, i.e. all the inter-arrival times of the pattern should be periodic.

In this chapter, we tackled this issue by proposing a novel model to discover partial periodic-frequent patterns in temporal databases. As demonstrated from the experimental results in Section 5.5, the proposed model solves the problem of mining *partial periodic-frequent* patterns as well as *rare item problem* effectively. However, this proposed model does not provide sufficient/complete information when a user wants to mine full periodic-frequent patterns, i.e., frequent patterns which are periodically occurring throughout the database. For mining full periodic-frequent patterns using this proposed model, we have to set $minRPS = 1$. For better understanding, let us consider an hypothetical example.

Example 20 *Consider the figures 5.6, 5.7 and 5.8. The figures depict different hypothetical cases showing timestamps at which a pattern X is occurring. In Case-1, pattern X is occurring at timestamps 2, 4 and 6. In Case-2, pattern X is occurring at timestamps 8, 10 and 12. In Case-3, pattern X is occurring*

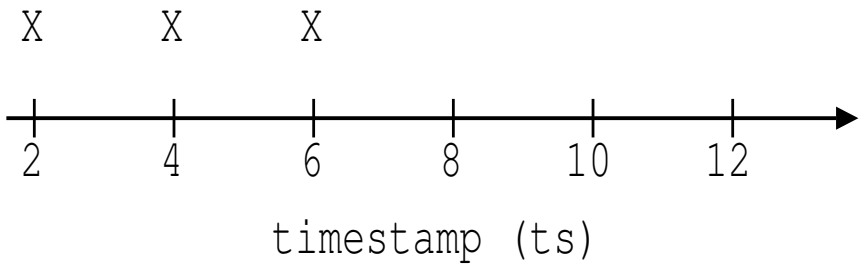


Figure 5.6 Case-1 : Occurrence timeline for pattern X

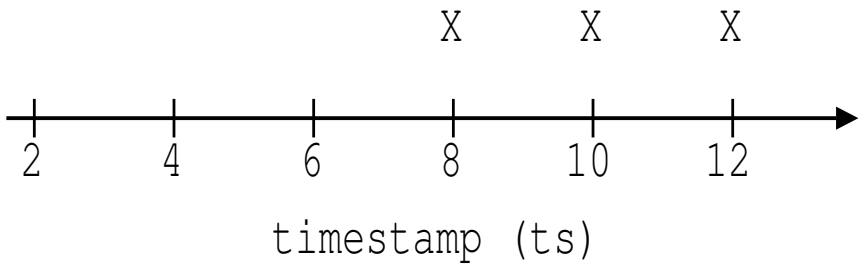


Figure 5.7 Case-2 : Occurrence timeline for pattern X

at timestamps 2, 4, 6, 8, 10 and 12. In all the three cases, let PER(X)=2. Therefore, RPS(X)=1 in all the three cases.

In Example 20, pattern *X* has $RPS(X) = 1$ in all the three cases, namely Fig 5.6, 5.7 and 5.8. As we can observe, if $minRPS$ is set as 1, the proposed model will mine patterns of all the three cases. Because of the same value of RPS, this proposed model cannot distinguish between the patterns of Case-1 and Case-2 from the patterns of Case-3. So, if a user wants to mine exclusively full periodic-frequent patterns, i.e., frequent patterns which are periodically occurring throughout the database (patterns of Case-3), then the proposed model cannot be used.

However, we can use the proposed model of periodic-correlated patterns (discussed in Chapter 4) for mining the patterns of Case-3 exclusively.

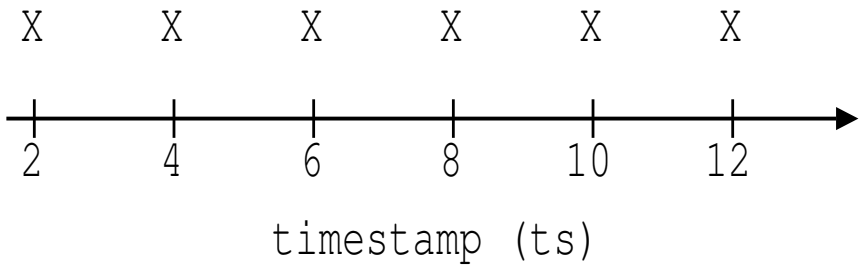


Figure 5.8 Case-3 : Occurrence timeline for pattern X

5.7 Summary

We have proposed a model that solves the problem of mining *partial periodic-frequent* patterns as well as *rare item problem* effectively. It enables every pattern to satisfy different $period$ and minimum number of cyclic repetitions depending on its items. A null-invariant and convertible anti-monotonic measure, *relative periodic-support*, was discussed to determine the partial periodic interestingness of a pattern in a database. A pattern-growth algorithm has also been presented to find periodic-frequent patterns. Experimental results show that the proposed model can find useful information and that the algorithm is efficient.

Chapter 6

Conclusions

Data mining transforms data into business intelligence and has become an important part of the modern organizations. Thus, data mining is being used in wide range of industry applications, such as marketing, surveillance, fraud detection, customer relationship management, bio-informatics, etc. Tanbeer et al. [54] have proposed a periodic-frequent pattern mining model to extract periodic-frequent patterns from transactional databases. This thesis is focused on the limitations of this model and proposed two improved models/approaches to address the limitations. In this last chapter of the thesis, we summarize the approaches proposed in this thesis and also give possible future directions in the field of periodic pattern mining.

6.1 Summary

The popular adoption and successful industrial application of this basic model [54] suffers from the following three obstacles:

- Since the input data is a *transactional database*, the model cannot handle datasets in which multiple transactions can share a common timestamp and/or when transactions occur at irregular time intervals.
- Since only a single $minSup$ and $maxPer$ are used to find the patterns in the entire database, the model implicitly assumes that all items in the database have uniform $support$ and $periodicity$. However, in reality, databases have a non-uniform item distribution, which considers that items have different $support$ and $periodicity$ values. If the items in a database vary widely, then finding periodic-frequent patterns using a single $minSup$ and $maxPer$ leads to the *rare item problem*.
- Current studies on periodic pattern mining have focused on discovering full periodic-frequent patterns, i.e., finding all frequent patterns that have exhibited complete cyclic repetitions throughout the entire database. However, partial periodic patterns are more common due to the imperfect nature of real-world databases. Therefore, the model cannot assess the partial periodic behavior of a frequent pattern in a database.However, partial periodic-frequent patterns are more common due

to the imperfect nature of real-world databases. Discovering partial periodic-frequent patterns in temporal databases has numerous applications.

Considering the input data as a temporal database addresses the first obstacle in periodic-frequent pattern model. In this thesis, we have proposed the following two approaches in non-uniform temporal databases to address the second and third obstacles:

- Discovering Periodic-Correlated Patterns in Non-Uniform Temporal Databases.
- Discovering Partial Periodic-Frequent Patterns in Non-Uniform Temporal Databases.

As part of our first contribution, to address the second obstacle, we have proposed a novel model to discover periodic-correlated patterns in a non-uniform temporal database. A new interestingness measure, *periodic-all-confidence*, has been proposed to address the problem in *periodicity* dimension. An efficient pattern-growth algorithm has been proposed to discover all periodic-correlated patterns in a database. Experimental results demonstrate that the proposed model is efficient. We discussed the usefulness of periodic-correlated patterns with a real-world case study and showed that the proposed model may be utilized to discover interesting periodic-correlated patterns involving both frequent and rare items from FAA-Accidents database effectively.

As part of our second contribution, we have introduced a novel model to address the problem of finding partial periodic-frequent patterns in non-uniform temporal databases. It enables every pattern to satisfy different $period$ and minimum number of cyclic repetitions depending on its items. A novel measure, *relative periodic-support*, was proposed to determine the partial periodic interestingness of a pattern in a database. A pattern-growth algorithm has also been presented to find partial periodic-frequent patterns. Experimental results demonstrate that the proposed model is efficient. We discussed the usefulness of partial periodic-frequent patterns with a real-world case study and showed that the proposed model may be utilized to find prior knowledge about event keywords and their associations in Twitter data.

6.2 Future Work

There are still many interesting problems which are worth continuing research. Some of them are:

- The proposed algorithms require 2 scans of the temporal database. Research can be further done on how to reduce the number of scans to 1
- Our study has been confined to mining periodic patterns from a static temporal database. The proposed approaches can be extended to incremental mining of temporal databases.
- It is interesting to investigate alternative measures of *periodic-all-confidence* and *relative periodic-support* to satisfy user and/or application requirements.
- Research can be done on the change in periodic behavior of rare items due to noise.

Chapter 7

List of Publications

7.1 Related Publications

1. *Venkatesh, J.N.*, Uday Kiran, R., Krishna Reddy, P., Kitsuregawa, M.: Discovering periodic-correlated patterns in temporal databases. In: Transactions on Large-Scale Data and Knowledge-Centered Systems, **TLDKS 2018** (Accepted with minor revision. In Press.)

2. Uday Kiran, R., *Venkatesh, J.N.*, Fournier-Viger, P., Toyoda, M., Krishna Reddy, P., Kitsuregawa, M., 2017c. Discovering periodic patterns in non-uniform temporal databases. In: Advances in Knowledge Discovery and Data Mining - 21st Pacific-Asia Conference, **PAKDD 2017**, Jeju, South Korea, May 23-26, 2017, Proceedings, Part II. pp. 604-617. doi:10.1007/978-3-319-57529-2_47

3. *Venkatesh, J.N.*, Uday Kiran, R., Krishna Reddy, P., Kitsuregawa, M.: Discovering periodic-frequent patterns in transactional databases using all-confidence and periodic-all-confidence. In: Hartmann, S., Ma, H. (eds.) **DEXA 2016**. LNCS, vol. 9827, pp. 55-70. Springer, Cham (2016). doi:10.1007/978-3-319-44403-1_4

7.2 Other Publications

1. Uday Kiran, R., *Venkatesh, J.N.*, Toyoda, M., Kitsuregawa, M., Krishna Reddy, P. In: Discovering partial periodic-frequent patterns in a transactional database. **Journal of Systems and Software** 125, 170-182. doi:10.1016/j.jss.2016.11.035

Bibliography

[1] G. Adomavicius and A. Tuzhilin. Expert-driven validation of rule-based user models in personalization applications. *Data Min. Knowl. Discov.*, 5(1-2):33–58, Jan. 2001.

[2] G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Trans. on Knowl. and Data Eng.*, 17(6):734–749, June 2005.

[3] R. Agrawal, T. Imieliński, and A. Swami. Mining association rules between sets of items in large databases. In *SIGMOD*, pages 207–216, 1993.

[4] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *Proceedings of the 20th International Conference on Very Large Data Bases*, VLDB '94, pages 487–499, San Francisco, CA, USA, 1994. Morgan Kaufmann Publishers Inc.

[5] K. Amphawan, P. Lenca, and A. Surarerks. Mining top-k periodic-frequent pattern from transactional databases without support threshold. In *Advances in Information Technology*, pages 18–29, 2009.

[6] A. Anirudh, R. U. Kiran, P. K. Reddy, and M. Kitsuregaway. Memory efficient mining of periodic-frequent patterns in transactional databases. In *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1–8, Dec 2016.

[7] A. Anirudh, R. U. Kiran, P. K. Reddy, M. Toyoda, and M. Kitsuregawa. An efficient map-reduce framework to mine periodic frequent patterns. In *International Conference on Big Data Analytics and Knowledge Discovery (DaWaK)*, Aug 2017.

[8] W. G. Aref, M. G. Elfeky, and A. K. Elmagarmid. Incremental, online, and merge mining of partial periodic patterns in time-series databases. *IEEE TKDE*, 16(3):332–342, Mar. 2004.

[9] P. Berkhin. *A Survey of Clustering Data Mining Techniques*, pages 25–71. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[10] J. Bradshaw. Yams - yet another measure of similarity. In *EuroMUG*, 2001. `http://www.daylight.com/meetings/emug01/Bradshaw/Similarity/YAMS.html`.

[11] S. Brin, R. Motwani, and C. Silverstein. Beyond market baskets: generalizing association rules to correlations. In *SIGMOD*, pages 265–276, 1997.

[12] S. Brindha, K. Prabha, and S. Sukumaran. A survey on classification techniques for text mining. In *2016 3rd International Conference on Advanced Computing and Communication Systems (ICACCS)*, volume 01, pages 1–5, Jan 2016.

[13] S.-S. Chen, T. C.-K. Huang, and Z.-M. Lin. New and efficient knowledge discovery of partial periodic patterns with multiple minimum supports. *J. Syst. Softw.*, 84(10):1638–1651, Oct. 2011.

[14] Z.-H. Deng. Diffnodesets: An efficient structure for fast mining frequent itemsets. *Applied Soft Computing*, 41:214 – 223, 2016.

[15] Z.-H. Deng and S.-L. Lv. Prepost+: An efficient n-lists-based algorithm for mining frequent itemsets via children-parent equivalence pruning. *Expert Systems with Applications*, 42(13):5424 – 5432, 2015.

[16] H. Ding, G. Trajcevski, P. Scheuermann, X. Wang, and E. Keogh. Querying and mining of time series data: Experimental comparison of representations and distance measures. *Proc. VLDB Endow.*, 1(2):1542–1552, Aug. 2008.

[17] G. Dong and J. Li. Efficient mining of emerging patterns: Discovering trends and differences. In *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '99, pages 43–52, 1999.

[18] FAA. *Federal Aviation Authority*, 2015. Available at `http://www.asias.faa.gov/pls/apex/f?p=100:1:7565565412795:::::`.

[19] P. Fournier-Viger, J. C.-W. Lin, Q.-H. Duong, and T.-L. Dam. *PHM: Mining Periodic High-Utility Itemsets*, pages 64–79. Springer International Publishing, Cham, 2016.

[20] M. M. Gaber, A. Zaslavsky, and S. Krishnaswamy. Mining data streams: A review. *SIGMOD Rec.*, 34(2):18–26, June 2005.

[21] K. Gouda and M. J. Zaki. Efficiently mining maximal frequent itemsets. In *Proceedings of the 2001 IEEE International Conference on Data Mining*, ICDM '01, pages 163–170, 2001.

[22] J. Gray. The next database revolution. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, SIGMOD '04, pages 1–4, New York, NY, USA, 2004. ACM.

[23] J. Han, H. Cheng, D. Xin, and X. Yan. Frequent pattern mining: Current status and future directions. *DMKD*, 14(1), 2007.

[24] J. Han, G. Dong, and Y. Yin. Efficient mining of partial periodic patterns in time series database. In *ICDE*, pages 106–115, 1999.

[25] J. Han, W. Gong, and Y. Yin. Mining segment-wise periodic patterns in time-related databases. In *KDD*, pages 214–218, 1998.

[26] J. Han, J. Pei, Y. Yin, and R. Mao. Mining frequent patterns without candidate generation: A frequent-pattern tree approach. *Data Min. Knowl. Discov.*, 8(1):53–87, Jan. 2004.

[27] J. Han, J. Wang, Y. Lu, and P. Tzvetkov. Mining top.k frequent closed patterns without minimum support. In *Proceedings of the 2002 IEEE International Conference on Data Mining*, ICDM '02, pages 211–218, 2002.

[28] Y.-H. Hu and Y.-L. Chen. Mining association rules with multiple minimum supports: a new mining algorithm and a support tuning mechanism. *Decision Support Systems*, 42(1):1–24, 2006.

[29] N. Jiang and L. Gruenwald. Research issues in data stream association rule mining. *SIGMOD Rec.*, 35(1):14–19, Mar. 2006.

[30] S. Kim, M. Barsky, and J. Han. Efficient mining of top correlated patterns based on null-invariant measures. In *PKDD*, pages 177–192, 2011.

[31] W.-Y. Kim, Y.-K. Lee, and J. Han. Ccmine: Efficient mining of confidence-closed correlated patterns. In *Advances in Knowledge Discovery and Data Mining*, pages 569–579, 2004.

[32] R. U. Kiran and M. Kitsuregawa. Novel techniques to reduce search space in periodic-frequent pattern mining. In *DASFAA (2)*, pages 377–391, 2014.

[33] R. U. Kiran and P. K. Reddy. Towards efficient mining of periodic-frequent patterns in transactional databases. In *DEXA (2)*, pages 194–208, 2010.

[34] R. U. Kiran and P. K. Reddy. Novel techniques to reduce search space in multiple minimum supports-based frequent pattern mining algorithms. In *EDBT*, pages 11–20, 2011.

[35] R. U. Kiran, J. Venkatesh, M. Toyoda, M. Kitsuregawa, and P. K. Reddy. Discovering partial periodic-frequent patterns in a transactional database. *Journal of Systems and Software*, 125:170 – 182, 2017.

[36] H.-P. Kriegel, M. S hubert, and A. Zimek. Angle-based outlier detection in high-dimensional data. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '08, pages 444–452, New York, NY, USA, 2008. ACM.

[37] Y. K. Lee, W. Y. Kim, D. Cao, and J. Han. Comine: efficient mining of correlated patterns. In *ICDM*, pages 581–584, 2003.

[38] B. Liu, W. Hsu, and Y. Ma. Mining association rules with multiple minimum supports. In *KDD*, pages 337–341, 1999.

[39] S. Ma and J. Hellerstein. Mining partially periodic event patterns with unknown periods. In *ICDE*, pages 205–214, 2001.

[40] P. Magdalinos, C. Doulkeridis, and M. Vazirgiannis. Enhancing clustering quality through landmark-based dimensionality reduction. *ACM Trans. Knowl. Discov. Data*, 5(2):11:1–11:44, Feb. 2011.

[41] E. R. Omiecinski. Alternative interest measures for mining associations in databases. *IEEE Trans. on Knowl. and Data Eng.*, 15:57–69, January 2003.

[42] B. Özden, S. Ramaswamy, and A. Silberschatz. Cyclic association rules. In *ICDE*, pages 412–421, 1998.

[43] J. Pei, J. Han, and L. V. Lakshmanan. Pushing convertible constraints in frequent itemset mining. *Data Mining and Knowledge Discovery*, 8:227–252, 2004.

[44] G. Pyun, U. Yun, and K. H. Ryu. Efficient frequent pattern mining based on linear prefix tree. *Knowledge-Based Systems*, 55:125 – 139, 2014.

[45] M. M. Rashid, M. R. Karim, B. S. Jeong, and H. J. Choi. Efficient mining regularly frequent patterns in transactional databases. In *DASFAA (1)*, pages 258–271, 2012.

[46] M. Renz, R. Cheng, and H.-P. Kriegel. Similarity search and mining in uncertain databases. *Proc. VLDB Endow.*, 3(1-2):1653–1654, Sept. 2010.

[47] A. Ritter, Mausam, O. Etzioni, and S. Clark. Open domain event extraction from twitter. In *KDD*, pages 1104–1112, 2012.

[48] G. A. Seber and A. J. Lee. In *Linear regression analysis*, volume 936 of *John Wiley & Sons*, 2012.

[49] H. Stormer. Improving e-commerce recommender systems by the identification of seasonal products. In *Twenty second Conference on Artificial Intelligence*, pages 92–99, 2007.

[50] A. Surana, R. U. Kiran, and P. K. Reddy. Selecting a right interestingness measure for rare association rules. In *International Conference on Management of Data*, pages 105–115, 2010.

[51] A. Surana, R. U. Kiran, and P. K. Reddy. An efficient approach to mine periodic-frequent patterns in transactional databases. In *PAKDD Workshops*, pages 254–266, 2011.

[52] M. K. Swamy, P. K. Reddy, and S. Srivastava. Extracting diverse patterns with unbalanced concept hierarchy. In *Advances in Knowledge Discovery and Data Mining - 18th Pacific-Asia Conference, PAKDD 2014, Tainan, Taiwan, May 13-16, 2014. Proceedings, Part I*, pages 15–27, 2014.

[53] P.-N. Tan, V. Kumar, and J. Srivastava. Selecting the right interestingness measure for association patterns. In *Knowl. Discovery and Data Mining*, pages 32–41, 2002.

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

[55] T. Uno, M. Kiyomi, and H. Arimura. Lcm ver.3: Collaboration of array, bitmap and prefix tree for frequent itemset mining. In *Proceedings of the 1st International Workshop on Open Source Data Mining: Frequent Pattern Mining Implementations*, OSDM '05, pages 77–86, New York, NY, USA, 2005. ACM.

[56] B. Vaillant, P. Lenca, and S. Lallich. *A Clustering of Interestingness Measures*, pages 290–297. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

[57] G. M. Weiss. Mining with rarity: A unifying framework. *SIGKDD Explorations*, 6(1):7–19, 2004.

[58] T. Wu, Y. Chen, and J. Han. Re-examination of interestingness measures in pattern mining: a unified framework. *DMKD*, 21(3):371–397, 2010.

[59] W. Wu and L. Gruenwald. Research issues in mining multiple data streams. In *Proceedings of the First International Workshop on Novel Data Stream Pattern Mining Techniques*, StreamKDD '10, pages 56–60, New York, NY, USA, 2010. ACM.

[60] H. Xiong, P.-N. Tan, and V. Kumar. Hyperclique pattern discovery. *Data Mining and Knowledge Discovery*, 13(2):219–242, 2006.

[61] J. Yang, W. Wang, and P. S. Yu. Mining asynchronous periodic patterns in time series data. *IEEE Trans. Knowl. Data Eng.*, pages 613–628, 2003.

[62] R. Yang, W. Wang, and P. Yu. Infominer+: mining partial periodic patterns with gap penalties. In *ICDM*, pages 725–728, 2002.

[63] H. Yao, H. J. Hamilton, and C. J. Butz. A foundational approach to mining itemset utilities from databases. In *SDM*, pages 482–486. SIAM, 2004.

[64] M. J. Zaki, S. Parthasarathy, M. Ogihara, and W. Li. New algorithms for fast discovery of association rules. Technical report, Rochester, NY, USA, 1997.

[65] M. Zhang, B. Kao, D. W. Cheung, and K. Y. Yip. Mining periodic patterns with gap requirement from sequences. *ACM Trans. Knowl. Discov. Data*, 1(2), Aug. 2007.

[66] W. Zhang, T. Yoshida, X. Tang, and Q. Wang. Text clustering using frequent itemsets. *Knowledge Based Systems*, 23(5):379–388, July 2010.

[67] Z. Zhou, Z. Wu, C. Wang, and Y. Feng. Efficiently mining mutually and positively correlated patterns. In *Advanced Data Mining and Applications, Second International Conference, ADMA 2006, Xi'an, China, August 14-16, 2006, Proceedings*, pages 118–125, 2006.

[68] Z. Zhou, Z. Wu, C. Wang, and Y. Feng. Mining both associated and correlated patterns. In *Computational Science - ICCS 2006, 6th International Conference, Reading, UK, May 28-31, 2006, Proceedings, Part IV*, pages 468–475, 2006.