Mining Association Rules Events over Data Streams

Aref Faisal Mourtada

A Thesis

in

The Department

of

Concordia Institute for Information Systems Engineering (CIISE)

Presented in Partial Fulfillment of the Requirements

for the Degree of

Master of Applied Science (Information Systems Security) at

Concordia University

Montréal, Québec, Canada

January 2017

CONCORDIA UNIVERSITY

School of Graduate Studies

This is to certify that the thesis prepared

By: **Aref Faisal Mourtada**

Entitled: **Mining Association Rules Events over Data Streams**

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

Master of Applied Science (Information Systems Security)

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the Final Examining Committee:

______________________ Chair
Dr. Chadi Assi

______________________ External Examiner
Dr. Wahab Hamou-Lhadj

______________________ Examiner
Dr. Chung Wang

______________________ Supervisor
Dr. Mourad Debbabi

______________________ Supervisor
Dr. Benjamin Fung

Approved by ______________________
Dr. Rachida Dssouli, Chair
Department of Concordia Institute for Information Systems Engineering (CIISE)

__________ 2017

____
Dr. Amir Asif, Dean
Faculty of Engineering and Computer Science

Abstract

Mining Association Rules Events over Data Streams

Aref Faisal Mourtada

Data streams have gained considerable attention in data analysis and data mining communities because of the emergence of a new classes of applications, such as monitoring, supply chain execution, sensor networks, oilfield and pipeline operations, financial marketing and health data industries. Telecommunication advancements have provided us with easy access to stream data produced by various applications. Data in streams differ from static data stored in data warehouses or database. Data streams are continuous, arrive at high-speeds and change through time. Traditional data mining algorithms assume presence of data in conventional storage means where data mining is performed centrally with the luxury of accessing the data multiple times, using powerful processors, providing offline output with no time constraints. Such algorithms are not suitable for dynamic data streams. Stream data needs to be mined promptly as it might not be feasible to store such volume of data. In addition, streams reflect live status of the environment generating it, so prompt analysis may provide early detection of faults, delays, performance measurements, trend analysis and other diagnostics. This thesis focuses on developing a data stream association rule mining algorithm among co-occurring events. The proposed algorithm mines association rules over data streams incrementally in a centralized setting. We are interested in association rules that meet a provided minimum confidence threshold and have a lift value greater than 1. We refer to such association rules as strong rules. Experiments on several datasets demonstrate that the proposed algorithms is efficient and effective in extracting association rules from data streams, thus having a faster processing time and better memory management.

Acknowledgments

First of all, thanks to Allah for his gracious blessings, for health, for family, for friends and for enabling me to complete this thesis.

I would like to express my sincerest gratitude to my supervisors Dr. Mourad Debbabi and Dr. Benjamin Fung who have supported me throughout my thesis research with knowledge, help and guidance. I am profoundly grateful to them as they have been a great source of encouragement throughout my study and as they have been so patient till this thesis was finished.

I would like to thank all my colleagues in the lab. And a special thanks goes to my colleague and friend Sujoy Ray who has helped me a lot throughout this thesis.

This research is supported by an NSERC National Defence Partnership Grant in Collaboration with MDA Corporation.

Last, but not least, I am deeply grateful to all my family members for their unbounded support and motivation throughout this entire process. I would like to express my love and gratitude to my father who have always believed that education should be among the first priorities in life, to my mother who has always cared for me no matter how old I grow, to my brothers who have always had my back, to my wife who is my love and life partner and finally to the two great blessings from Allah my kids *Hiba* and *Faisal*. I am blessed to have you all in my life.

“Proclaim in the name of thy Lord who created. Proclaim, for thy Lord is the most Bountiful. He who taught by pen. Taught Man what he knew not.” - The Holy Quran

To my dad *Faisal* and my mom *Faizeh*...

To my wife *Nada* and my wonderful kids *Hiba & Faisal*...

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The rapid technological advancement in the last century has changed our society life style. As computing devices have evolved to become mobile, miniaturized, affordable and human independent, they have become a vital part of almost every single task of our daily life. The fact that these devices can be equipped with sensing capabilities, strong processing power and advanced communication modules, allows them to form large monitoring networks that generate a huge volume of data continuously at high speeds and in real time. This is referred to as data streams. Such data represent events triggered by changes in environments or live status reports, and therefore they require online processing and analysis in a swift and prompt manner to be able to extract useful information. The huge volume and high arrival rate of stream data makes data storage infeasible. Hence, delays in processing of stream data could cause data and information loss.

Data mining provides several techniques that can explore hidden information within data. However, the characteristics of data streams pose constrains and challenges that traditional data mining algorithms were not designed to take into consideration. In this thesis, we propose a data mining algorithm to mine association rules events from multiple data streams in an incremental manner. The scope of the proposed algorithm lies in identifying frequent associated events that can generate interesting association rules. We investigate a generic and an efficient stream mining approach to produce selective association rules among dependent events from a sliding window over multiple streams. These association rules are determined incrementally in a centralized setting.

1.1 Motivation

In the recent years, the need to analyze large volumes of data has motivated scholars in the field of data mining to improve the mining processes in order to accommodate large static datasets, minimize the resources required in the analysis and generate mining models that represent the data [28]. As the environment hosting or generating the data evolved much more, it was essential for mining processes to address the problem of continuous and rapid data generation. In addition, the telecommunication technological leaps, in all domains, infrastructure, speed, bandwidth and hardware have encouraged real-time monitoring of different venues of our lives and capturing any possible data for further analysis. Live data is generated in a continuous manner and can be transmitted around the globe rapidly. Some popular examples of data streams include Internet browsing traffic, social media messages, weather information, vehicle guiding systems, road traffic updates and many more.

Several businesses and organizations are converting most of their data infrastructure in to a streaming model because of the potential that streams hold and the increasing demand of real-time analysis of data [67]. The challenges of stream mining are increasing as streams are evolving to have faster arrival rates with huge data volume. Recent statistics estimate daily data generated volume to reach 2.5 billion terabytes. This number is expected to grow to 40 billion terabytes by 2020 [16].

Data streams require the design of advanced and efficient algorithms and frameworks that would process, analyze, aggregate different data sources and respond to any query in a real-time fashion while eliminating any unnecessary operations. The generated mining models need to be updated incrementally upon the presence of new data without re-initiating the whole mining process. The mining output needs to be available upon request such that it would represent the current status of the stream data. Consider the following example. A network of sensors deployed to monitor Tsunami incidents and alert authorities of possible warnings or attacks. Different devices continuously measure weather conditions, earthquake signs, volcanic eruptions and other parameters [3]. If the generated data was to be stored and analyzed later in an offline manner, disasters will strike without being detected. Therefore, data streams require to be mined promptly and in an efficient and incremental manner.

One of the important data mining techniques is *association rule mining (ARM)*. ARM is used to discover interesting relations between data elements in large datasets. Strong association rules are identified using different interestingness measures. Rules can be useful in behavior analysis, situational awareness, event prediction or decision making.

1.2 Objective and Contribution

The concept of ARM was first introduced back in 1993 by Agrawal et al. [3]. Over time, scholars presented several similar approaches. Some approaches enhanced the Agrawal et al.'s algorithm, others proposed new approaches. The ARM process is composed of two steps: frequent itemsets are extracted from data and then association rules are generated from these itemsets. In stream data mining, research initiatives mainly focused on the first step. As a matter of fact when mining a data stream with a sliding window topology, majority of the proposed algorithms, discussed in Section 3.1, only attempt to extract frequent itemsets without generating the association rules. Other algorithms, Section 3.2, which state that they aim to generate the rules, actually generate the rules on demand without taking into consideration the characteristics of data streams. The objectives of this thesis include analyzing the existing gap in the literature for online rule generation over data streams and proposing a new approach of ARM over data streams. In addition, we perform benchmark association rule generation on existing common datasets and compare performance with existing research work. To the best of our knowledge, this is the first work on incremental association rule mining over a sliding window which extracts and maintains the generated rules. Our contributions in this thesis are summarized as follows:

(1) We introduce a novel algorithm called *Mining Association Rules from Event Data Streams (MAREDS)* to extract interesting association rules from evolving data streams over a sliding window model in a centralized setting.

(2) We propose an in-memory efficient data structure, the *partial association enumeration tree* (PAET), which maintains frequent itemsets, potential itemsets, itemsets that might become frequent when the window slides, and the relations among the frequent itemsets.

(3) We propose a generic and scalable framework to generate interesting association rules from

PAET and incrementally maintain the rules as the data stream window slides.

(4) We implement the proposed algorithm and framework and evaluate the performance over several real-life and synthetic datasets, in terms of processing time, memory footprint, responsiveness and scalability. Extensive experiments suggest that the proposed technique performs better than the existing techniques in the majority of the test cases.

1.3 Thesis Organization

The rest of this thesis is organized as follows. Chapter 2 provides background knowledge of frequent itemset mining, association rule mining and stream data mining. Chapter 3 reviews related work. The problem of mining association rules over data streams is defined in Chapter 4. Chapter 5 introduces the proposed incremental association rule mining algorithm over data streams. It presents a case study demonstrating the flow of the proposed algorithm as well. Experimental results are presented in Chapter 6. Chapter 7 draws the conclusion of the thesis and hints at future research directions.

Chapter 2

Background

2.1 Data Mining

The concept of *data mining* refers to the process of analyzing large datasets in order to extract hidden knowledge and implicit interesting patterns. Data mining outcomes or models are used in different operations such as decision making, status analysis and prediction of behavior outcomes. The wide range of applications where data mining is useful made it an active researched field. Yet, the term “Data Mining” was not introduced until the early 1990s. The roots of data mining can be traced back to three scientific fields: statistical studies, artificial intelligence and machine learning. The vision and understanding on how to extract useful information from data evolved as these different fields evolved. Early data analysis and pattern identification started via different statistics concepts. In 1775, Bayes and Price introduced Bayes theorem [13, 25], which examined current probability to prior probability. Later, in the early 1800s, regression analysis was introduced [13]. Regression was used to estimate relationships among variables. Then, the computer technological boom arrived, which increased data collection, storage and manipulation. Data has grown in size, complexity and availability. Data processing has expanded broadly with several advancements in different computer science fields, such as clustering, neural networks, genetic algorithms in the 1950s, decision trees in the 1960s and support vector machines in the 1990s [29, 22]. Today, the demand for data mining is increasing for all domains whether scientific or commercial. Data mining techniques have comprehensive analytics and powerful approaches that allow the tackling and analysis of more complex data. End users have access to data mining tools with user-friendly interfaces and graphical mining outcomes.

Data mining involves several techniques, each of which provide different data analysis approaches and output. These techniques can be classified into two categories: *descriptive* mining and *predictive* mining. Descriptive mining approach categorizes or extracts general characteristics or relations of a mined dataset [31]. Association mining, sequential mining and clustering are some of the main tasks involved in the descriptive mining techniques' tasks. Predictive mining approach analyzes historical data to extract relations or implications in order to predict future data values or part of it [31]. Examples of such are classification, regression and outlier detection. Some of the major data mining techniques are briefly presented below:

- ***Classification*:** Classification analysis divides a given dataset into distinguished classes or concepts. Classification models are identified by analyzing a training dataset where the class labels are predefined. These models predict categorical class labels over discrete or unordered datasets with unknown class labels [22].

- ***Regression*:** Regression analysis is a statistical approach that divides datasets into classes in a similar manner to classification. Regression uses training datasets to identify distribution trends as well. But unlike classification models, regression models perform numerical prediction rather than discrete labeling [22].

- ***Clustering*:** Cluster analysis shares the same outcome of classification and regression where data is organized into classes. The clustering technique, unlike classification and regression, does not have a defined set of classes. A clustering algorithm discovers proper classes using the principle of "maximizing the intraclass similarity and minimizing the interclass similarity" [22].

- ***Outlier Analysis*:** Outliers do not follow the general flow or behavior of the dataset they belong to. Mining algorithms usually label outliers as noise or exceptions. However, in some cases identifying such rare or unusual data can be useful in many fields. For example, identifying data packets with unusual payloads can prevent a malware spread or a hacking attack [22].

- ***Association Analysis*:** This type studies the frequency of distinct items occurring together in a given dataset and the relation among them. User-defined thresholds, such as itemset minimum support and association rule confidence, are used to limit the mining outcome to interesting results [22, 31]. Association rule mining is described in more details in Section 2.2.

2.2 Association Rule Mining

Association rule mining is one of the descriptive mining techniques that has shown great potential and captured scholars attention since it was first introduced in the early 1990s [31]. Its importance rises from its capabilities to discover unapparent relations, interesting correlations, frequent patterns, associations or casual structures among data in large datasets. These capabilities create a wide range of applications for association rule mining such as marketing, risk management, inventory control, network management and many more. The association rule mining problem is decomposed into two phases.

Phase I: Mining Frequent Itemsets

Mining frequent itemsets is a major part for several data mining techniques such as association rules, sequential patterns, and classification [61]. Its task is to explore combinations of items with minimum frequency threshold occurring within a dataset. Such combinations are termed as frequent itemsets.

Definition 2.1. (*Itemsets*). An itemset X, is a set of items, where $(X \subset I)$ and $I = \{i_1, i_2, \ldots, i_n\}$.

Definition 2.2. (*Support*). The support of an itemset X denotes the frequency of X within a dataset or a sliding window; it may be presented as the ratio of transactions containing all items of X as well.

$$sup(X) = \frac{\text{Number of transactions containing (X)}}{\text{Total number of transactions}(\tau)}$$

Definition 2.3. (*Frequent Itemset*). An itemset X is frequent if its $support(X) \geq s_{min}$, where s_{min} is a user-defined threshold for the least acceptable support value, known as *minimum support*.

Phase II: Association Rule Generation

Association rule generation is defined as forming relationships or correlations among frequent itemsets, extracted in "Phase I", known as association rules. The generated association rules may not necessarily provide meaningful information. Actually, some of the generated rules are misleading. Several measures are proposed by scholars to measure the interestingness of an association rule. The task of determining the interestingness is not simple nor objective. A rule that may be considered interesting for a particular application, may not be so for another. Two key interestingness measures used in ARM are *confidence* and *lift*. Both measures have been incorporated into MAREDS functionality. More highlights on association rules, confidence and lift are discussed below:

Definition 2.4. (*Association Rule*). An association rule is a representation of a relationship between two itemsets in the form of an if/then statement. A rule between itemsets X, antecedent (if-clause), and Y, consequent (then-clause), has the form of $(X \rightarrow Y)$, where $(X \cap Y) = \phi$.

Definition 2.5. (*Confidence*). The confidence of an association rule refers to the probability of both the antecedent and the consequent appearing in the same transaction. The confidence of association rule, $(X \rightarrow Y)$, refers to the percentage of transactions containing X that also contains Y.

$$conf(X \rightarrow Y) = \frac{sup(X \cup Y)}{sup(X)}$$

For an association rule $(X \rightarrow Y)$ to be interesting as per the confidence measure, $(X \cup Y)$ has to be a frequent itemset and $conf(X \rightarrow Y) \geq c_{min}$, where c_{min} is a user-defined threshold for the least acceptable confidence value, known as *minimum confidence*.

Example 2.1. Let us assume we are analyzing a data stream for road traffic. We can note that delay events would often occur with bad weather events. Association analysis might show that 70% of the sliding window data that include bad weather events also include delay events. This relationship could be formulated as follows: Bad weather implies delay with 70% confidence.

Definition 2.6. (*Lift*). The lift of a rule indicates the strength of a rule over the random co-occurrence of an antecedent with the consequent, given their individual support. In probability theory, lift refers to the joint probability of occurrences for two independent itemsets X and Y. In data mining, lift is treated as a ratio between the confidence of the association rule over the unconditional probability of the consequent Y.

$$\mathit{lift}(X \rightarrow Y) = \frac{sup(X \cup Y)}{sup(X) \times sup(Y)}$$

An association rule where the antecedent X and consequent Y are related by a probability that is more than the product of their individual probability of occurrences, or in other words when $\mathit{lift} > 1$, is considered to be interesting. A lift greater than 1 indicates that X and Y are dependent and are not occurring randomly together.

Example 2.2. Referring to the previous example of the road traffic data stream for road traffic. Let us assume that the following are support values for some of the events in the sliding window: $sup(\mathit{Accident}) = 70\%$, $sup(\mathit{Delay}) = 88\%$ and $sup(\mathit{Accident} \cup \mathit{Delay}) = 60\%$. Hence, the rule $\mathit{conf}(\mathit{Accident} \rightarrow \mathit{Delay}) = 85.71\%$. Such a rule sounds interesting as it is having high confidence and support values. However, the rule $\mathit{lift}(\mathit{Accident} \rightarrow \mathit{Delay}) = 0.97$. In such cases, the over all "Delay" events occurrences are actually less likely to happen as a result of that particular accident.

Several traditional association rule mining algorithms are proposed to mining frequent itemsets and association rules from transactional databases. Agrawal and Srikant [4] have proposed, *Apriori*, the first algorithm to mine frequent itemsets and generate association rules from a transactional database. Apriori first extracts single-frequent items. The single frequent items are then extended to dual frequent itemsets, itemsets with two items. The process continues until no more itemsets can be extracted. Each round of frequent itemset mining requires a new scan of the dataset. Afterwards, Apriori examines relations within each of the extracted itemsets. A rule is generated by splitting itemset X into two non-empty sets, Y and $(X - Y)$, represented by $(Y \rightarrow X - Y)$. A frequent itemset X, with n items, generates $(2^n - 2)$ association rules ignoring those with empty antecedents or consequents ($\phi \rightarrow X$ or $X \rightarrow \phi$). Rules that do not satisfy the minimum confidence threshold are discarded.

Example 2.3. Let us assume that the extracted frequent itemset is $X = \{a, b, c\}$. The itemset X contains 3 single items. Hence, the number of generated rules $= 2^3 - 2 = 6$:

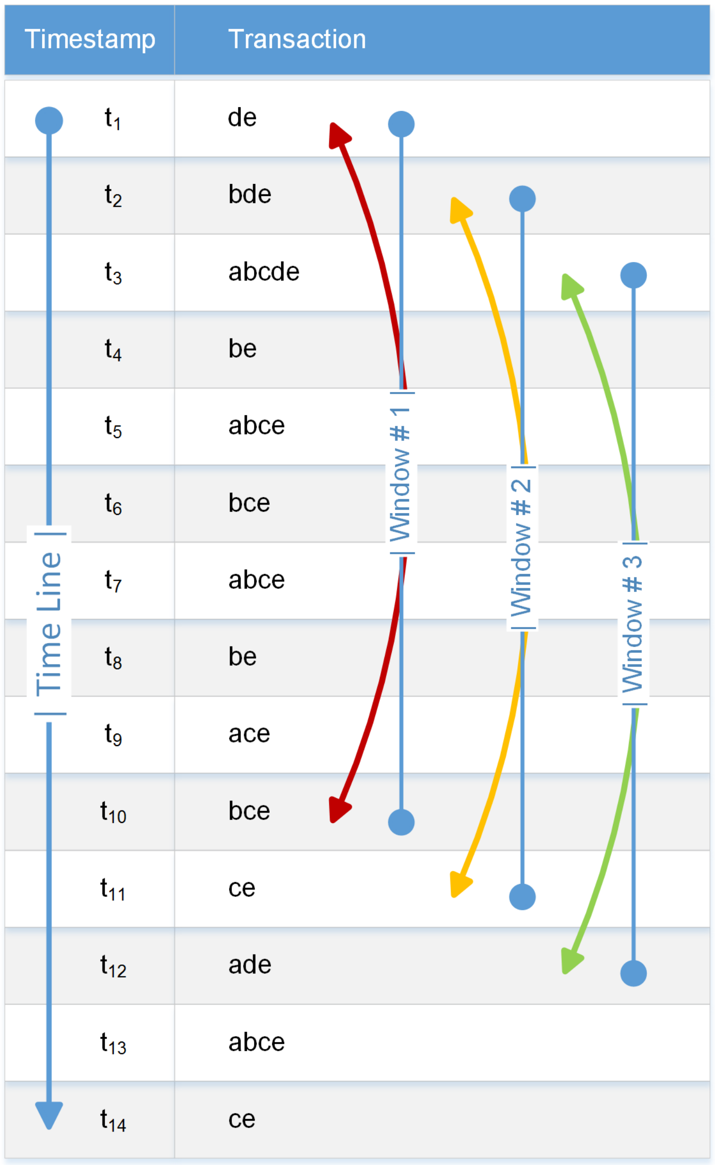
$a \rightarrow bc,\ \ ab \rightarrow c,\ \ ac \rightarrow b,\ \ b \rightarrow ac,\ \ ba \rightarrow a,\ \ c \rightarrow ab.$

Several variations were proposed to enhance performance of Apriori algorithm. Some of the common variations are bucket-hash itemsets, transaction reduction, candidate itemsets partitioning, data subset mining and dynamic itemset counting [29, 22]. Other new algorithms were proposed as well, such as *FP-Growth*, where a pre-fix tree is used to represent the database. Non-frequent items are discarded. Each frequent item is mined to extract the frequent itemsets without the need to re-access the database as the tree already holds the needed information [23]. Another proposed algorithm mines frequent itemsets using vertical data format where the transaction identifications are grouped by the items. All these variations or new proposals only focus on the first phase of the association rule mining, which is frequent itemset extraction. The second phase has not been improved and still has the same complexity which may be acceptable while mining traditional databases. If association rules are to be generated in a stream environment, it may be costly to perform all the second phase operations every time the sliding window is updated.

2.3 Data Streams

A stream of data is defined as a set of consecutive items that arrive in an orderly and timely manner. Streams are usually unbounded and continuous. They arrive at high speed and have dynamic data distribution. These characteristics of data streams make them distinct from traditional static data stored in databases or data warehouses. Therefore, new data analysis tools are needed to tackle data streams accordingly [28].

Streams consisting of distinct events in a general form are often described as *moments* in the research literature [55]. Moments are useful in understanding distribution of items' frequencies, analyzing stream properties and storing stream related knowledge in an optimized manner [55]. Stream data is presented as a set of co-occurring events which can be seen as an itemset where each item represents an event. Thus, a data stream denotes a temporal sequence of itemsets, also known as a *stream transaction*. Figure 2.1 depicts a sequence of data stream transactions. For example, $\{abcde\}$ represents five remote events occurring at timestamp t_3.



Timestamp	Transaction
t_1	de
t_2	bde
t_3	abcde
t_4	be
t_5	abce
t_6	bce
t_7	abce
t_8	be
t_9	ace
t_{10}	bce
t_{11}	ce
t_{12}	ade
t_{13}	abce
t_{14}	ce

Figure 2.1: Central coordinator monitoring data streams over a sliding window model

2.3.1 Data Stream History

Even though the interest in stream data is relatively recent, the concept of streams goes back for more than half a century. The term "*Stream*" was first introduced by Landin, in 1966, to model histories of loop variables when designing unimplemented computing languages [33]. In the next years, the concept of stream has been mainly discussed within the literature of data flow field [64]. Today, stream data analysis and mining have become an active area of research in computer science.

2.3.2 Applications

In this section we discuss some of the real-life data stream application domains in which data mining plays an important role. The data mining in streams helps to flag out unwanted or unique situations from normal behavior, extract patterns, identify adversary actions or customize information [1]. Figure 2.2 illustrates different data stream domains.

- *Networks and Telecommunications:* Network traffic forms one of the largest streams which holds tons of information. Analysis and mining of such streams helps in cyber-attack prevention, prevention of private data accessibility, detection of suspicious behaviors or intrusions, analysis of networks and user online-behavior and many more [20, 58].

- *Financial and Stock Markets:* Financial transaction streams are used to detect possible internet banking and credit card frauds. It is used in bankruptcy prediction for loan takers as well. In stock markets it is used to prevent and detect possible insider trading [18, 20, 52].

- *Roads and Transportation:* Streams from monitored roads and highways provide us with several useful information such as traffic forecast, expected travel times, suggested routes and alternative routes for navigation systems [7, 46].

- *Medical Care:* Some patients are provided with sensors to monitor their health conditions and vital signs. The data stream generated from aggregating the sensors output helps in detecting possible medication reaction, organ failures and internal injuries [1].

- *Social Media:* In the era of social media, social information such as posts, tweets, followers, followed by, likes and shares, generate one of the largest available data streams. Analysis and mining of social streams is used to identify the users' or entities' interests and preferences, predict their behaviors and habits and provide them with personalized services and products [21].

2.3.3 Types and Models

Data streams are categorized into two types according to the nature of the data perceived in the stream, *offline* and *online* [64].

Definition 2.7. (Offline streams.) Offline streams are bulk data that occur at regular time intervals. However, the flow of data is not constant. Data updates arrive at fixed times, daily, weekly, or monthly, with periods of inactivity in between.

Definition 2.8. (Online streams). Online streams refer to real-time data that arrive in an orderly manner.

Analysis and queries on data from offline streams are done in an offline manner and in respect of all previously received saved data [64]. Updating tasks in data warehouses are good examples of this type. Meanwhile, in online streams the data arrival rate and data volume are both high, causing the data storage to be infeasible. Online data streams usually represent a live situation which may require prompt analysis to extract useful live information. Internet-packets represent an online stream, which is impossible to store, yet needs to be analyzed promptly to anticipate any possible attacks.

Furthermore, when addressing data streams in a data analysis context, streams are categorized into three models based on which part of the data stream is used in the analysis or the mining process. Below are the description of each model:

Definition 2.9. (*Landmark Model*). In the landmark model the mining process includes all the stream data starting from a defined point of time, known as the *Landmark*, till the present. The outcome for such a model is practical for applications interested in historical data [54, 53].

Definition 2.10. (*Damped - Time Fading Model*). In the damped model a weight is assigned to each transaction in the stream. The weight value is inversely correlated with the transaction's age. This impacts old transactions to have less effect on the analysis or mining process outcome. Such a model is suitable for applications that require to consider historical data in the analysis process, yet the emphasis should be on newly available data [54, 53].

Definition 2.11. (*Sliding Window Model*). The sliding window model maintains the most recent stream transactions in a buffer, called a window. The window has a fixed size that may vary according to the application and/or system resources. When new transactions arrive, the oldest transactions are removed from the window. The same concept applies to any analysis or mining process. Analysis or mining is only applied on the transactions in the current window. Once new transactions are available and old ones are retired, the outcome is updated accordingly. Outcome of this model is desirable for applications seeking recent information from data streams [54, 53]. This process is known as a forgetting process, as it limits the amount of processed data and allows the mining process to forget old data and adapt to changes [7].

Usually the offline streams use a landmark model in the data analysis process, while online stream use either the damped model or the sliding window model as the outcome is expected to reflect the live information from the stream.

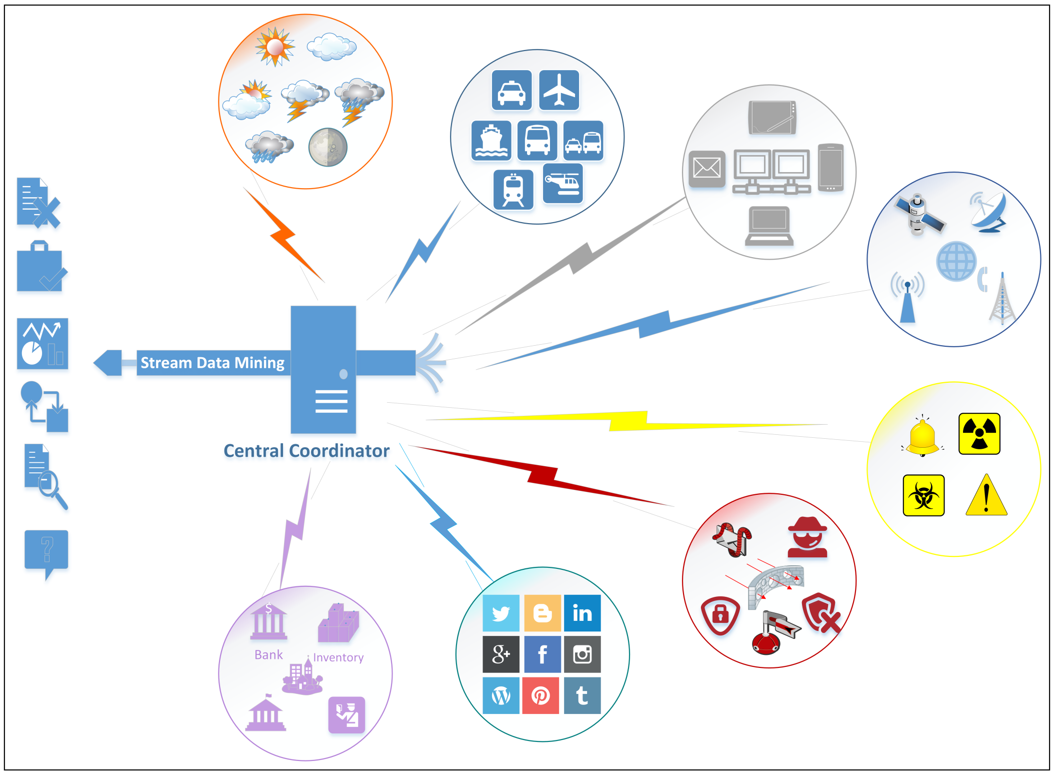


Figure 2.2: Miscellaneous data stream application domains

2.3.4 Challenges and Requirements

The stream data environments, unlike conventional databases and data warehouses, impose a set of challenges that make traditional analysis and mining algorithms fail or perform properly. The following elaborates on the major challenges and requirements of data stream analysis and mining algorithms [28, 30, 64, 1, 20].

- *Unbounded Data Feed:* Traditional data mining techniques analyze data in an offline fashion, where the data is stored in databases which offers the mining algorithm the luxury of multiple accesses. However, data streams are generated in an unbounded manner which makes storing all the data infeasible. Old data needs to be discarded to free space for newly arriving data. Hence, a stream mining algorithm is required to perform the mining task after a single data scan only.

- *Real-Time Mining:* A key characteristic of a stream is that the data's arrival is rapid. The mining algorithm has to process the data in a real-time manner while taking into consideration the datas arrival rate. A relaxed processing time can cause a bottleneck or even data loss as the data arrival rate is faster than the processing rate.

- *Resource Management:* A traditional mining algorithm is considered to be finite. It performs its assigned mining task and eventually terminates. On the contrary, a stream mining algorithm runs continuously and does not pause or end unless manually terminated. It constantly requires some of the systems resources from processing power, memory space and sometimes power energy. A stream mining algorithm requires proper resource management such that it would not exhaust the systems resources or block other processes from having them. Advanced scheduling and memory management techniques are important to take into consideration when designing a stream mining algorithm.

- *Data Structure Choice:* The choice of a proper data structure is an important element in any algorithm design. It is even more important for stream mining. Data structures are used to store the stream data and the mining outcome. In addition, data structures are accessed to update both incoming data and the mining outcome, or to retrieve information in response to users' queries. Hence, a stream mining algorithm needs to choose an efficient and a compact data structure that has a small memory footprint where data can be accessed with as minimum operations as possible. Storing any data on the disk has an additional overhead of I/O operations increasing the processing and data access time.

- *Update and Aggregation of Knowledge:* After mining the stream data in a timely manner, as mentioned in the previous point, several challenges rise regarding the extracted knowledge. Newly discovered knowledge has to be added or merged with the previous knowledge in an incremental manner. In addition, old knowledge has to be either degraded or retired.

- *Visualization:* Visualization is a powerful tool to understand and illustrate the data mining outcome. In some data stream applications, such as monitoring applications, visualization facilitates the analysis process. For example, a use of a graph that shows the relationship between mined association rules makes any action taking or decision making process easier and faster.

- *Data Evolution:* Stream data is generated in a rapid manner and represents real-world applications and environments. As the conditions change or differ in the environments, the underlying distribution of a data stream changes over time. This change may require a change in the user defined parameters, such as the minimum support or minimum confidence. A well designed stream mining algorithm should have some flexibility to interact with such requirement without the need to start the mining process all over again.

2.3.5 Challenges of Mining Data Streams

Data streams require real-time mining, which refers to mining the data as soon as it is generated and available. Traditional data mining techniques mine data in an offline fashion, where the data is stored and the algorithm has easy and multiple accesses to it. However, as stream data is generated in a huge, rapid and timely manner, storing all the data is infeasible. A stream mining algorithm has to extract information from generated data using only one data scan while taking into consideration the data's arrival rate. Additional mining operations, such as a second data scan, is resource consuming and can cause a bottleneck as data may arrive faster than the processing rate [54].

2.3.6 Concept Drift

The distribution of the generated data in a stream may change over time. This concept is known as temporal evolution, covariate shift, non-stationarity, or concept drift [7]. Concept drift occurs from unforeseen or unpredictable changes that may affect the sources generating stream data. The distribution of data in traditional databases is assumed to be static. However, this is not the case for real-time stream applications. The data in streams is being generated continuously. Hence, it is typical that different parameters and conditions reflected by such data to change over time. The distribution of data will change eventually as well. Traditional data mining models and algorithms may have poor performance or may produce inaccurate information if applied on stream data. A stream mining algorithm should be able to capture the change in data distribution and reflect it in the mining outcome. Concept drift in data streams urged to introduce new concepts such as the sliding window model, which was discussed earlier in Section 2.3.3.

Chapter 3

Literature Review

The motivation of this thesis is to build a generic, fast and scalable data stream mining technique to extract associations among events during decision making in various application areas. In this regard, a prescriptive data mining technique like ARM has shown great potential and captured scholars' attention since it was introduced in the early 1990s [32]. The fist frequent itemset mining over data streams was proposed by Manku and Motwani in 2002 [44]. The authors proposed an algorithm to mine single frequent items from a data stream. Later, they extended their scope to mine frequent itemsets. And since then several algorithms were introduced to tackle ARM over streams. Data stream mining algorithms are often categorized by two dimensions:

- *Centralized versus Distributed:* In a centralized setting, items are collected from various streams in a central location, known as a sink or coordinator, where the mining process takes place. In a distributed setting, data streams are mined at distinct locations. Each individual mining output is aggregated to form one global mining outcome.

- *Frequent Itemset Mining versus Association Rule Mining:* Frequent itemset mining extracts itemsets that meet a minimum support threshold in a dataset. As mentioned earlier, this is considered as the first phase of the two-phase ARM technique. In its second phase, association rules are formed from the extracted frequent itemsets of the first phase. Rules are filtered afterwards as per user or application requirements.

Most stream mining algorithms focus on the first phase. However, the second phase is equally important. Without the rule extraction, the information gained in the first phase may be misleading and can not be used directly in decision making. Hereafter, we review relevant related literature in data stream association rule mining categorized as stated above.

3.1 Centralized-Stream Frequent Itemset Mining

In the sliding window model, we focus on algorithms that mine whole and closed frequent itemsets with exact outcome as it is more relevant to our proposed algorithm, MAREDS. Algorithms that mine maximal itemsets or provide approximate outcome is information lossy. They may also have marginal errors or produce some false positive or false negative results. Hence, such algorithms are not considered.

Leung and Khan [37] propose one of the first algorithms to mine exact frequent itemsets from a data stream. The algorithm adopts a batch processing model with non-overlapping sliding windows. Incoming data is stored in a canonical ordered prefix-tree structure, *DSTree*. The tree nodes store current items support values and some particulars from the previous batch. Previous batch information is used to prevent any tree traversal during the update process. FP-growth mining technique [23] is used to extract the frequent itemsets from the DSTree. Tanbeer et al. [62] also use FP-growth mining technique but over a transactional based sliding window. The window transactions are maintained in an FP-tree-like structure, known as *CPS-tree*. CPS-tree is restructured occasionally to keep the nodes in descending order based on their support values. The restructuring process provides rapid tree accessibility and keeps its size minimal. Li and Lee [40] introduce *MFI-TransSW* algorithm which uses a bit sequence data structure to store the sliding window items. Left bit-shifting is used to add new transactions and retire the old ones while the AND operation is used to extract the frequent itemsets. *LDS* algorithm by Deypir and Sadreddini [17] uses three different forms of lists to store the sliding window items. The first list maintains items by the transactions they are present in. The second list maintains items through the transactions they are absent from. In the third list, item occurrences are stored as a bit string. Each item is maintained in the most optimum list type based on its frequency. Frequent itemsets are extracted from the lists upon user request using either Eclat[76], dEclat[77] or bEclat[6] algorithms. The choice of algorithm depends on the most common list type.

Chi et al. [12] propose *Moment* algorithm, the first algorithm to mine closed frequent itemsets over a data stream using a sliding window. Moment stores window transactions and extracts closed frequent itemsets in an inverse FP-Tree and a prefix tree structure, named *CET*, respectively. CET maintains additional nodes, known as boundary nodes, to address state changes such as: infrequent itemset becoming frequent and vice versa. The proposed concept of boundary nodes is so intuitive, yet it has a major flow. The number of maintained boundary nodes is relatively high compared to the number of closed frequent itemsets nodes especially with low minimum support value. This would cause slow tree traversal and memory exhaustion. *NewMoment* by Li et al. [39] and *TMoment* by Nori et al. [49] propose variations of the Moment algorithm. The sliding window transactions and the mined closed frequent itemsets are maintained in a prefix tree in both algorithms. In addition, a copy of the closed frequent itemsets are stored in a separate hash table to ease any update or query task. The first level nodes in the tree are used to store all items along with their occurrence information. This information is used to track the items' supports and extract the frequent itemsets. NewMoment represents the occurrences using a bit string while TMoment uses an integer array of transaction unique IDs. The other tree nodes hold the closed frequent itemsets and their support. Jiang and Gruenwald [27] propose *CFI-Stream* that stores all the sliding window transactions, frequent and infrequent, in a prefix tree in a closed itemsets format. Frequent itemsets are extracted upon user request by applying minimum support threshold. *CloStream* by Yen et al. [74] maintain all the sliding window transactions as well. CloStream creates two tables to store current transactions and single items separately along with a list of closed itemsets. *QMINE* algorithm [48] also uses two similar tables. However, the second table in QMINE holds a set of bit victors to keep track of each item's presence in the first table. Both CloStream and QMINE algorithms generate frequent itemsets upon request and by applying desired minimum support value. Keming Tang et al. [63] propose *Stream_FCI* which uses an FP-tree like structure, called *DFP-tree*, to store the sliding window transactions. Frequent items and their support are stored in an external head table. Each frequent item points to its first occurrence in the tree. In addition, the algorithm creates links across matching items in different tree branches. The extracted closed frequent itemsets are saved in a separate table to ease update and query operations.

The landmark model analyzes all data in a stream starting from a defined point of time. Data volume grows to infinity as time goes on. Some data needs to be discarded and hence the mining outcome will not represent exact frequent itemsets. In this model, Li et al. [41] propose *DSM-FI* algorithm to mine frequent itemsets from a data stream by batch. Batch data is stored in a prefix-tree. Tree pruning is applied periodically to remove infrequent or irrelevant itemsets and keep the tree size minimal. Frequent itemsets are extracted from the tree periodically or upon request. Zhi-Jun et al. [81] propose dividing frequent itemsets into equivalent classes. Each class itemsets, support values and border itemsets are maintained in an enumeration tree. The border itemsets are used to filter frequent itemsets and hence keep tree size under control. Liu et al. [42] propose *FP-CDS* algorithm to mine closed frequent itemsets over the same model. Potential frequent itemsets in each batch are stored in a prefix tree. Frequent itemsets are extracted from the tree in real time upon user request. Yu et al. [75] propose a false-negative based algorithm to extract approximate frequent itemsets. *Chernoff Bound* [11] is used to prune off infrequent itemsets as more data arrive.

Over a stream damped model, Chang and Lee propose *estDec* algorithm [9] to mine frequent itemsets. estDec maintains itemsets that have potential to become frequent in the near future in a lexicographic tree. Decay element is represented by a weight value assigned to each of the nodes in a reverse chronological order. Frequent itemsets are extracted from the tree upon user request. Woo and Lee [71] extend estDec to *estMax* algorithm to mine maximal frequent itemsets. In estMax, after adding potential itemsets to the lexicographic tree, the tree is restructured to keep only maximal frequent itemsets. estMax uses two thresholds, *Maximality Mark* and *Maximum Lifetime*, to improve mining performance. The Maximality Mark identifies new nodes and eliminates the need to fully traverse the tree upon update. Maximum lifetime is used to opt out old frequent itemsets. Leung and Jiang [36] propose *DUF-streaming* algorithm to mine frequent items over a damped model by batches. DUF-streams uses *UF-growth* algorithm [35] to extract frequent itemsets and stores them in an FP-like tree. An "*Expected Support*" value, representing the decay element, is computed incrementally after processing each batch to eliminate older itemsets.

3.2 Centralized-Stream Association Rule Mining

Aggarwal and Yu [2] are among the first scholars to propose a framework for online mining of frequent itemsets and generating association rules. The proposed concept of online mining provides end users with capabilities of directly querying a database of generated association rules. Queries have the flexibility of using different support and confidence values without any additional computational cost. An adjacency lattice is used to maintain extracted frequent itemsets. The lattice structure allows easy association rules generation, as well as rule redundancy removal. It eliminates the need to re-access original data for support queries. The algorithm does not take into consideration any dataset update or any transaction insertion and deletion, which makes it not suitable for data streams. Shin and Lee [57] propose an algorithm to mine association rules over a damped stream model. Frequent itemsets are mined using *estDec* algorithm [9]. Afterwards, a stack traversal approach is used to generate association rules. The generation process divides rules into ordered rules and unordered rules. Ordered rules indicate that all items on the left-hand side are lexicographically greater than those on the right-hand side. However, the algorithm does not keep track of generated rules. It requires to generate rules from scratch upon each user request by traversing the estDec tree. Thakkar et al. [65] propose a data stream management system which mines association rules over a sliding window model. Frequent itemsets are mined using *Verification* algorithm [47] and maintained in an FP-like tree. Association rules are generated after a predefined number of elapsed transactions. Optional pruning is applied over the extracted rules to eliminate duplicate and uninteresting rules. Association rules are saved in a database for further analysis and rule comparison. However, the saved rules are not used in subsequent rounds of rule generation. Su et al.[10] propose *FFI_Stream* to mine association rules from stream data containing quantitative attributes. Stream data is divided into fuzzy sets using *SWEM* clustering algorithm [15]. Afterwards, frequent itemsets are acquired from the sets using modified version of *UF_streaming* [35]. A "*Membership Function Bias*", known as *MFB_measure*, is proposed to measure interesting frequent itemsets that could generate interesting association rules. Yet, rules are not actually generated. Thool and Voditel [66] propose *Streaming-Rules* algorithm to mine association rules over a landmark window model. Frequent itemsets are mined using *Space-Saving* algorithm [45] and maintained in a list structure.

One-to-one association rules are generated by rescanning the current window. The rule generation process is not incremental. It is done from scratch upon every data update. Corpinar and Gundem [14] propose *PNRMXS* algorithm to mine positive and negative association rules from *XML* streams over a landmark model. A modified version of FP-growth [23] is used to mine frequent itemsets from each stream batch. The rule generation process extracts one-to-one association rules from scratch in a non-incremental way. Paik et al. [50] propose to mine maximal frequent items from *XML* streams. Association rules are generated for each batch separately. Then, they are filtered by a minimum confidence threshold. Association rules from each batch are accumulated for the entire stream in a landmark model fashion. Yet, the rule extraction for each batch is performed from scratch every time. Vijayarani and Prasannalakshmi [68] conduct an analysis on association rule generation over data streams using traditional mining approaches. The objective of the experiments was to examine the number of extracted rules and the execution time with various data arrival rates. The experiments adopt a batch mining approach. The mining task starts from scratch for each batch. Such behavior does not reflect stream mining environment.

3.3 Distributed-Stream Association Rule Mining and Frequent Itemset Mining

Park and Kargupta [51], Sawant and Shah[56] and Zeng et al. [78] conduct surveys that investigate approaches for distributed data mining. Majority of surveyed algorithms assume that the datasets are stored in distributed locations across the network. In addition, they deem to have the luxuries of traditional data mining techniques mentioned earlier. These algorithms mainly focus on frequent itemset mining and do not examine any mechanism for rule generation. Moreover, only few articles can be seen on distributed ARM over stream data.

Manjhi et al. [43] propose to extract frequent items from multiple distributed streams. Distinct monitors maintain single items support for each stream. Frequent items are communicated periodically to a central monitor in a hierarchical manner. A local monitor communicates its frequent items to an upper level monitor, which merges it with its own items and pass it to the next level. This process goes on until the all frequent items are gathered at the central monitor. The hierarchical architecture minimizes communication and computation cost at the central monitor. In addition, it allows more frequent items to be extracted minimizing the error. Sun et al. [60] propose a framework to extract frequent patterns from several distributed data streams. Frequent patterns are mined from each stream using adaptive filtering techniques. Global patterns are extracted after aggregating local patterns. Then, they are communicated back to local streams to refine and verify the newly extracted outcome. Huang et al. [26] propose a distributed sequential pattern mining algorithm that uses two *Map-Reduce* functions over a *Hadoop* platform [5]. The first function extracts candidate patterns locally at different Hadoop nodes and generates a summary. The second function aggregates the generated different summaries to produce a final summary. The global summary is updated incrementally to incorporate new candidates and remove expired ones. Wang and Chen [70] propose a frequent itemset mining framework over distributed data streams using a landmark model. *hSynopsis* algorithm [69] is used to mine local frequent itemsets. A central coordinator aggregates local streams synopsises to form a global synopsis. The framework poses communication strategies and constraints that minimize communication with the coordinator. Zhang and Mao [79] use a combination of decision trees and *naïve Bayes* classifier [34] to mine frequent patterns from distributed data streams. Local streams build decision trees to generate statistical summaries. A statistical summary approximates items' support values of the current stream batch. Then, local patterns are formed from both statistical summaries and key attributes in the decision tree. The naïve Bayes classifier is used to aggregate local patterns to form global patterns. Cesario and Mastroianni [8] propose a hybrid single-pass/multiple-pass framework for mining frequent items and itemsets from distributed data streams. The framework consists of multiple layers of mining. The mining outcome is communicated forward and backward, locally and globally, across the different layers to refine the mining output and minimize the error. Finally, Wu et al. [72] propose a decentralized approach to mine event association rules over multiple streams. Frequent stream events are filtered locally and communicated to a central location where they are merged through an Apriori-based map-reduce function. Association rules are generated through another map-reduce function upon user request. Generated rules are not stored nor are used in subsequent requests.

Table 3.1 presents an overview of related work discussed above. As elaborated, scholars in centralized setting focus on frequent itemset mining. Association rule generation is not discussed thoroughly in data streams. Available stream rule-generating algorithms mainly extract all frequent itemsets and then apply traditional techniques to extract the rules. They do not take into consideration any stream characteristics. This leads to several unnecessary computations that are so costly in a stream environment. In distributed setting, most of the techniques require reliable and extensive communication of information to share the information. The techniques do not consider any information that would be lost because of the lack of a global data view. In the proposed algorithm, MAREDS maintains interesting associations incrementally from data streams without the need for any further computations.

Table 3.1: Survey of algorithms from the literature review

Algorithm	**Setting**	**Window Model**	**Itemsets**	**Freqeunt Itemset**	**Update**	**Rules**	**Dataset**
DSTREE[37]	C	Sliding	All FIS	On req.	B	No	Mixed
CPS[62]	C	Sliding	All FIS	On req.	T	No	Mixed
MFI-TransSW[40], LCS[17]	C	Sliding	All FIS	On req.	T	No	Synthetic
Moment[12], Tmoment[49]	C	Sliding	Closed	Stored	T	No	Mixed
NewMoment[39]	C	Sliding	Closed	Stored	T	No	Synthetic
CFI-Stream[27], CloStream[74], QMINE[48]	C	Sliding	Closed	On req.	T	No	Synthetic
Stream_FCI[63]	C	Sliding	Closed	Stored	T	No	Synthetic
DSM-FI[41]	C	Sliding	All FIS	On req.	B	No	Synthetic
(Zhi-Jun et al.)[81]	C	Landmark	All FIS	Stored	T	No	Mixed
FP-CDS[42]	C	Landmark	Closed	On req.	B	No	Synthetic
(Yu et el)[75]	C	Landmark	All FIS	Stored	B	No	Synthetic
estDec[9]	C	Landmark	All FIS	On req.	T	No	Mixed
estMax[71]	C	Decay	Maximal	On req.	T	No	Mixed
DUF-streaming[36]	C	Decay	All FIS	On req.	B	No	Mixed
(Aggarwal and Yu)[2]	C	Decay	All FIS	Stored	NA	Yes (On req.)	Synthetic
(Shin and Lee)[57]	C	NA	All FIS	On req.	T	Yes (On req.)	Mixed
SWIM[65], FFI_Stream[10]	C	Decay	All FIS	Stored	T	Yes	Mixed
Streaming-Rules[66]	C	Sliding	Top-K	Stored	T	[1-1]	Synthetic
PNRMXS[14]	C	Landmark	All FIS	Stored	B	[1-1]	Synthetic
(Paik et al.)[50]	C	Landmark	Maximal	Stored	B	Yes	NA
(Manjhi et al.)[43]	D	Landmark	All FIS	Stored	B	No	Mixed
(Sun et al.)[60]	D	Decay	All FIS	NA	B	No	Synthetic
(Huang et al.)[26]	D	Decay	NA	Stored	NA	No	Synthetic
(Wang and Chen)[70]	D	NA	Maximal	On req.	T	No	Mixed
(Zhang and Mao)[79]	D	Decay	All FIS	Stored	B	No	Mixed
(Cesario and Mastroianni)[8]	D	Decay	All FIS	Stored	B	No	Real
(Wu et al.)[72]	D	Sliding	1 & 2 Itemsets	Stored	T	[1-1]	Synthetic

**(Setting: C:Centralized, D:Distributed) (Update: B:by Batch, T: per Transaction)*

Chapter 4

Problem Description

In this chapter, we formally define the association rule mining problem over data streams. In Section 4.1, we review the foundations of association rule mining in a data stream environment. The problem statement is stated in Section 4.2.

4.1 Initial Definitions

In a centralized setting, we assume the set of all possible items, also referred as the *Alphabet*, generated by a data stream is represented by $\mathcal{A} = \{e_1, e_2, \ldots, e_n\}$. A coordinator would receive co-occurring events as a set of items at each defined moment or timestamp t_j. The set of items arriving at the same time stamp t_j form one itemset and is referred to as a transaction. Relations between items with itemsets are examined to identify association rules.

Definition 4.1. (*Strong Association Rule*). The relationship among two mutually exclusive itemsets X and Y ($X, Y \subset \mathcal{A}$) is deemed to be strong if it follows two conditions. First, the support value of $(X \cup Y)$ should be at least s_{min} within a predefined number of recent transactions τ up to the current moment. Second, the ratio of $(X \cup Y)$'s support compared to X's support should be at least c_{min} within the last τ transactions. Such relation between X and Y is defined as a *strong association rule* and is denoted by $(X \rightarrow Y)$. In data mining X and Y are called as antecedent and consequent respectively.

The variables s_{min} and c_{min} are two application-defined thresholds named as *minimum support* and *minimum confidence*, respectively. In MAREDS, we investigate an additional condition. We are interested in association rules where antecedent X and consequent Y are related by a probability more than the product of their individual occurrence's probability, hence the association rule lift should be greater than one. The notion of lift was discussed in Section 2.2. We denote association rules that satisfy these three conditions as *interesting rules*.

Definition 4.2. (*Interesting Association Rule*). An interesting association rule is a strong association rule that has lift value greater than 1.

As the window slides, a new transaction is added to the window while the oldest transaction is removed. All the relationships among itemsets require to be re-investigated as some of the itemsets' support values may change. Wu et al. [73] defines the support and confidence values of an itemset over a timestamped data stream using a *lifetime* function l_j.

Definition 4.3. (*Lifetime function*). At a moment t_j, a lifetime function l_j is defined over a sliding window of size τ as $l_j : \mathcal{A} \to T$. T is a set of timestamps expressed as: $\{t : j - \tau < t \leq j; t \in T\}$.

Example 4.1. In Figure 2.1, let us assume that the sliding window size $\tau = 10$. Then, $l_{10}(a) = \{t_3, t_5, t_7, t_9\}$ and $l_{10}(ab) = \{t_3, t_5, t_7\}$.

The association rules are examined in the same manner as they are dependent on support and confidence values. In this context, we introduce *momentary association rule*.

Definition 4.4. (*Momentary Association Rule* $\mathcal{R}_j$). The association $\mathcal{R}_j$ is examined among two sets of items X and Y in the most recent τ transactions ending at timestamp t_j of the sliding Window. $\mathcal{R}_j$ is characterized through three functions: momentary support (sup_j), momentary confidence ($conf_j$) and momentary lift ($lift_j$) of itemset X as follows:

$$sup_j(X) = |\bigcap_{e \in X} l_j(e)|; X \subseteq \mathcal{A} \tag{1}$$

$$conf_j(X \to Y) = \frac{sup_j(X \cup Y)}{sup_j(X)}; X, Y \subseteq \mathcal{A} \tag{2}$$

$$lift_j(X \to Y) = \frac{\tau \times sup_j(X \cup Y)}{sup_j(X) \times sup_j(Y)}; X, Y \subseteq \mathcal{A} \tag{3}$$

From equation (1), it is observed that $sup_j(X) \geq sup_j(X \cup Y)$. This is also known as the *Apriori* property [32]. Similarly, from equation (2), the confidence is also related as $conf_j(X \rightarrow Y) \geq conf_j(X \rightarrow Y \cup Z)$ and $conf_j(X \rightarrow Y) \leq conf_j(X \cup Z \rightarrow Y \setminus Z)$ where $Z \subset Y$.
Based on the above functions, we define the following related terms:

Definition 4.5. (*Stream Frequent Itemset*). An itemset X with momentary support no less than s_{min}, i.e., $sup_j(X) \geq s_{min}$, is considered *frequent*. Otherwise, the itemset X is *infrequent*.

Definition 4.6. (*Omnipresent Itemset*). An itemset X that is present in all the window transactions, i.e., $sup_j(X) = \tau$, is considered *omnipresent*.

Definition 4.7. (*Intermediate and Closed Itemsets*). If a frequent itemset X has a superset $X \cup Y$ that has the exact same support of X and there is no superset of $X \cup Y$ that has the same support, then X and $X \cup Y$ are called *intermediate itemsets* and *closed itemset*, respectively. For simplicity, $X \cup Y$ is represented as XY in the rest of this thesis.

Association rules can be categorized based on the number of items in the antecedent and the consequent as follows:

(i) $[1 - 1]$: named one-to-one rule. It refers to association rules where only single items are present in each of antecedent and the consequent.

(iii) $[n - 1]$: named many-to-one rule. It refers to association rules where multiple items are present in the antecedent, while the consequent holds only single items.

(ii) $[1 - n]$: named one-to-many rule. It refers to association rules where only a single item is present in the antecedent, while the consequent holds multiple items.

(iv) $[n - n]$: named many-to-many rule. It refers to association rules where multiple items are present in both of antecedent and the consequent.

4.2 Problem Statement

In a centralized setting, a coordinator collects event data streams generated from various monitored environments. The problem is to extract and maintain $[n - n]$ momentary association rules

$\mathcal{R}_j$ of our interest from the most recent τ transactions ending at t_j from the data stream. The interesting association rule $\mathcal{R}_j$ is incrementally generated using previously stored outcome information during the mining of $\mathcal{R}_{j-1}$. An interesting association rule is denoted as $(X \xrightarrow{l} Y)$ and satisfies the following constraints:

(i) $sup_j(X \cup Y) \geq s_{min}$

(ii) $conf_j(X \to Y) \geq c_{min}$

(iii) $lift_j(X \to Y) > 1$

* $X, Y \subset \mathcal{A}$ and $X \cap Y = \phi$.

Figure 2.1 is an example of a central coordinator analyzing an event data stream with alphabet $\mathcal{A} = \{a, b, c, d, e\}$ over a sliding window of size $\tau = 10$. We use this example throughout the thesis with minimum support $s_{min} = 3$ and minimum confidence $c_{min} = 0.7$.

Chapter 5

Proposed Algorithm

Rajaraman and Ullman [55] point out that the core challenge of stream mining lies in handling the rapid speed of the data stream with the complexity of the mining algorithm(s). They recommend in-memory, single-pass and real-time data processing. We propose, MAREDS, an in-memory mining algorithm of a selected set of association rules in a centralized data stream setting. The mining procedure captures stream data using a bit matrix and a prefix tree, the partial association enumeration tree, over a sliding window model. The proposed approach enables us to answer the query "What are the current interesting associations rules? " at any time.

In the next Section 5.1, we analyze different association rule generation properties that would help in reducing the solution space.

5.1 Association Rule Property Analysis

The associations among itemsets in the sliding window may change their status from relevant to irrelevant, and vice versa, upon the arrival of each new transaction. This happens due to changes in the items' support values and their co-occurrence with other events. In this regard, we examine a set of properties that would help in reducing the solution search space and thereby improve the efficiency of the association rule generation algorithm. We extract the following properties from the aforementioned constraints stated in the problem statement Section 4.2:
Given a frequent itemset $\{abcd\}$, it is noticed that the confidence values for the following generated

rules are: $conf_j(abc \rightarrow d) \geq conf_j(ab \rightarrow cd) \geq conf_j(a \rightarrow bcd)$. Hence, the confidence values of association rules generated from the same itemset have an anti-monotonic property. The confidence function $conf_j$ is anti-monotonic with respect to the number of items in the antecedent. If $conf_j(abc \rightarrow d)$ does not hold, there exists no association rules among the items in $\{abcd\}$ at the higher order $[n-2]$, $[n-3]$, etc, where the consequent is a superset of d. This significantly reduces the scope of the rule search. Property 1 mathematically captures our interest.

Property 1. Given frequent itemsets X, XY, XZ and XYZ are related through an anti-monotonic relationship such that if $(X \rightarrow Y)$ or $(X \rightarrow Z)$ or $(XY \rightarrow Z)$ or $(XZ \rightarrow Y)$ then $(X \rightarrow YZ)$.

Proof. In order for association rule $(X \rightarrow YZ)$ to fulfill constraint (ii): $[\frac{sup_j(XYZ)}{sup_j(X)} \geq c_{min}]$, then using Apriori property:
$\frac{sup_j^p(XY)}{sup_j^p(X)} \geq c_{min}$, $\frac{sup_j^p(XZ)}{sup_j^p(X)} \geq c_{min}$, $\frac{sup_j^p(XYZ)}{sup_j^p(XY)} \geq c_{min}$ and $\frac{sup_j^p(XYZ)}{sup_j^p(XZ)} \geq c_{min}$.
□

Any itemset that is present in every transaction of the current sliding window cannot generate interesting association rules. For example, a special weather condition may persist for a whole day. Property 2 helps in pruning the scope of search for interesting association rules among frequent itemsets. It signifies that omnipresent data is not relevant to be included in the interesting associations among two set of recent events.

Property 2. An interesting association rule can not have an antecedent and/or consequent itemset that is omnipresent; otherwise, the lift value is less or equal to 1 (*lift* ≤ 1).

Proof. Assume $(X \rightarrow Y)$ is an association rule where itemset X is present in all transactions. $lift_j(X \rightarrow Y) = \frac{\tau \times sup_j(XY)}{sup_j(X) \times sup_j(Y)}$ and $sup_j(x) = \tau$, then $lift_j(X \rightarrow Y) = \frac{sup_j(XY)}{sup_j(Y)}$; and since $sup_j(XY) \leq sup_j(Y)$, then $lift_j(X \rightarrow Y) \leq 1$. The same can applied if itemset Y is omnipresent.
□

An intermediate itemset X which is subset of a closed frequent itemset XY indicates that these itemsets occur together. Therefore, as stated in Property 3, the relationship between such itemsets will form an interesting association rule, unless they are omnipresent. The generated rule, $(X \xrightarrow{l} Y)$, does not require any confidence or lift evaluation .

Property 3. Given a frequent closed itemset XY and an intermediate itemset X, there exists an interesting association rule $(X \xrightarrow{l} Y)$ if itemset Y is not omnipresent.

Proof. Since, $sup_j(X) = sup_j(XY)$:

- X, Y and XY are all frequent (Apriori property), constraint (i) is met.

- $conf_j(X \to Y) = \frac{sup_j(XY)}{sup_j(Y)} = 100\%$, constraint (ii) is met.

- $lift_j(X \to Y) = \frac{\tau}{sup_j(Y)}$. So, if Y is not omnipresent $(sup_j(Y) < \tau)$, then $lift_j(X \to Y) > 1$, which meets constraint (iii). $\square$

The following properties, Property 4, Property 5 and Property 6, extend the general concept of Property 3. They provide various conditions to mine interesting association rules. Our main interest in this context is to make use of co-occurring events through these properties to improve the performance of the incremental association rule mining algorithm by reducing the search space.

Property 4. Given a non-omnipresent frequent closed itemset XYZ and intermediate itemsets X, XY , then $(X \xrightarrow{l} YZ)$, $(XY \xrightarrow{l} Z)$, $(XZ \xrightarrow{l} Y)$, $(X \xrightarrow{l} Y)$ and $(X \xrightarrow{l} Z)$ are all valid interesting association rules except in the case(s) where the consequent(s) are omnipresent.

Proof. Using both Property 1 and Property 3. $\square$

Property 5. Given a frequent itemset X, an intermediate itemset XY and a non-omnipresent frequent closed itemset XYZ such that $sup_j(XY) = sup_j(XYZ)$, the following is true:

a. if $(X \xrightarrow{l} Y) \Rightarrow (X \xrightarrow{l} YZ)$

b. if $(X \overset{l}{\nrightarrow} YZ) \Rightarrow (X \overset{l}{\nrightarrow} Y)$

Proof. Proof as follows:

a. Given $(X \xrightarrow{l} Y)$ is an interesting association rule, it meets constraints (i), (ii) and (iii).

   - Since $sup_j(XY) = sup_j(XYZ)$ then implication $(X \to YZ)$ meets constraints (i) and (ii).

   - Since $sup_j(Y) \geq sup_j(YZ)$, then $\frac{\tau \times sup_j(XY)}{sup_j(X) \times sup_j(Y)} \leq \frac{\tau \times sup_j(XYZ)}{sup_j(X) \times sup_j(YZ)}$. Thus, the implication $(X \xrightarrow{l} YZ)$ meets constraint (iii).

b. Given $(X \overset{l}{\nrightarrow} YZ)$, it does not meet constraint (i), (ii) or (iii).

- If $(X \overset{l}{\nrightarrow} YZ)$ does not meet constraint (i) or (ii) and since $sup_j(XYZ) = sup_j(XY)$ then $(X \overset{l}{\nrightarrow} Y)$ also does not meet the same constraint.

- If $(X \overset{l}{\nrightarrow} YZ)$ does not satisfy constraint (iii) and since $\frac{\tau \times sup_j(XYZ)}{sup_j(X) \times sup_j(YZ)} \geq \frac{\tau \times sup_j(XY)}{sup_j(X) \times sup_j(Y)}$, then $(X \overset{l}{\nrightarrow} Y)$ also does not satisfy the same.

□

From Eq. 1, Eq. 2 and Eq. 3, it is clear that the evaluations of different thresholds require retrieving support values for the involved itemsets. In this context, Property 6-a. significantly reduces the load of support evaluation for some itemsets. The other properties, Property 6-b and Property 6-c, help is minimizing the rule evaluations during the $[n - n]$ association rule generation.

Property 6. Given a frequent closed itemset XY and an intermediate itemset X, the following is true for itemset XYZ:

a. $sup_j(XZ) = sup_j(XYZ)$

b. if $(XY \xrightarrow{l} Z) \Rightarrow (X \xrightarrow{l} YZ)$

c. if $(X \overset{l}{\nrightarrow} YZ) \Rightarrow (XY \overset{l}{\nrightarrow} Z)$

Proof. Proof as follows:

a. Since $sup_j(X) = sup_j(XY)$, Y occurs in every transaction X occurs in over the whole sliding window j. Now, if X and Z jointly occur, it will be among a subset of the XY transactions, hence $sup_j(XZ) = sup_j(XYZ)$.

b. Given $(XY \xrightarrow{l} Z)$ is an interesting association rule, it meets constraints (i), (ii) and (iii).

- Since $sup_j(X) = sup_j(XY)$, then implication $(X \xrightarrow{l} YZ)$ meets constraints (i) and (ii).

- Since $sup_j(Z) \geq sup_j(YZ)$, then $\frac{\tau \times sup_j(XYZ)}{sup_j(XY) \times sup_j(Z)} \leq \frac{\tau \times sup_j(XYZ)}{sup_j(X) \times sup_j(YZ)}$. Thus, implication $(X \xrightarrow{l} YZ)$ meets constraint (iii) as well.

c. Given $(X \overset{l}{\nrightarrow} YZ)$, it does not meet constraint (i), (ii) or (iii).

- If $(X \overset{l}{\nrightarrow} YZ)$ does not meet constraint (i) or (ii) and since $sup_j(XY) = sup_j(X)$, then $(X \overset{l}{\nrightarrow} Y)$ also does not meet the same constraint(s).
- If $(X \overset{l}{\nrightarrow} YZ)$ does not satisfy constraints (iii) and since $\frac{\tau \times sup_j(XYZ)}{sup_j(X) \times sup_j(YZ)} \geq \frac{\tau \times sup_j(XYZ)}{sup_j(XY) \times sup_j(Z)}$, then $(XY \overset{l}{\nrightarrow} Z)$ also does not meet the same.

$\square$

An itemset that has a support value less than the product of the sliding window size by the minimum confidence can participate as a consequent in an interesting association rule. As stated in Property 7, this can help in determining whether a rule is lifted from its consequent value without any additional evaluation.

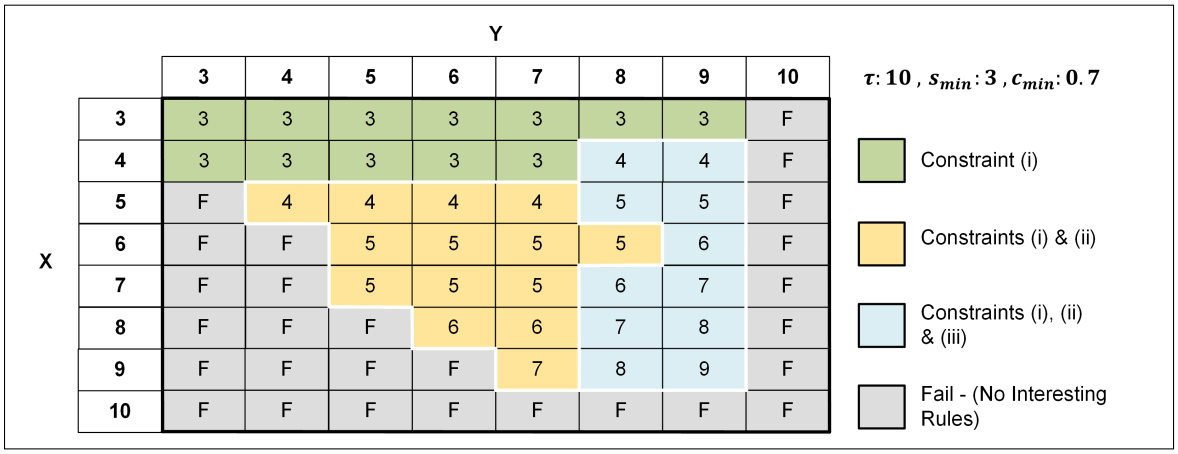
Property 7. Given frequent itemsets X, Y and XY, a rule $(X \overset{l}{\rightarrow} Y)$ is valid if itemset X is not omnipresent, $conf_j(X \rightarrow Y) \geq c_{min}$ and $sup_j(Y) < \tau \times c_{min}$.

Proof. Proof as follows:

- X, Y and XY are frequent itemsets, hence constraint (i) is met.

- $conf_j(X \rightarrow Y) \geq c_{min}$, constraint (ii) is met.

- Given $sup_j(Y) < \tau \times c_{min}$ and $sup_j(Y) < \tau \times \frac{sup_j(XY)}{sup_j(X)}$ since $\frac{sup_j(XY)}{sup_j(X)} \geq c_{min}$, then, $\frac{\tau \times sup_j(XY)}{sup_j(X) \times sup_j(Y)} > 1$ as support of $sup_j(X) < \tau$. constraint (iii) is met. $\square$



X \ Y	3	4	5	6	7	8	9	10
3	3	3	3	3	3	3	3	F
4	3	3	3	3	3	4	4	F
5	F	4	4	4	4	5	5	F
6	F	F	5	5	5	5	6	F
7	F	F	5	5	5	6	7	F
8	F	F	F	6	6	7	8	F
9	F	F	F	F	7	8	9	F
10	F	F	F	F	F	F	F	F

$\tau: 10$, $s_{min}: 3$, $c_{min}: 0.7$

🟩	Constraint (i)
🟨	Constraints (i) & (ii)
🟦	Constraints (i), (ii) & (iii)
⬜	Fail - (No Interesting Rules)

Figure 5.1: Minimum requirement of $sup_j(XY)$ for supports of X and Y

5.1.1 Interesting Association Rules Identification

At a given time slot t_j, Figure 5.1 provides an example for the minimum requirements for the joint support of itemset XY to generate an interesting association rule $(X \xrightarrow{l} Y)$ over a sliding window of size $\tau = 10$, minimum support $s_{min} = 3$ and minimum confidence $c_{min} = 0.7$ (or 70%). The support values are presented in four distinct regions of colors. The gray region, blocks marked with the value 'F', implies that interesting rules can not be generated with the corresponding support values of X and Y for the given setting. In the green region, constraints (ii) and (iii) are already satisfied and only constraint (i) requires to be verified. In the yellow region, constraint (iii) is already satisfied and both constraints (i) and (ii) need to be verified. Finally, the region marked in blue requires to evaluate all three constraints.

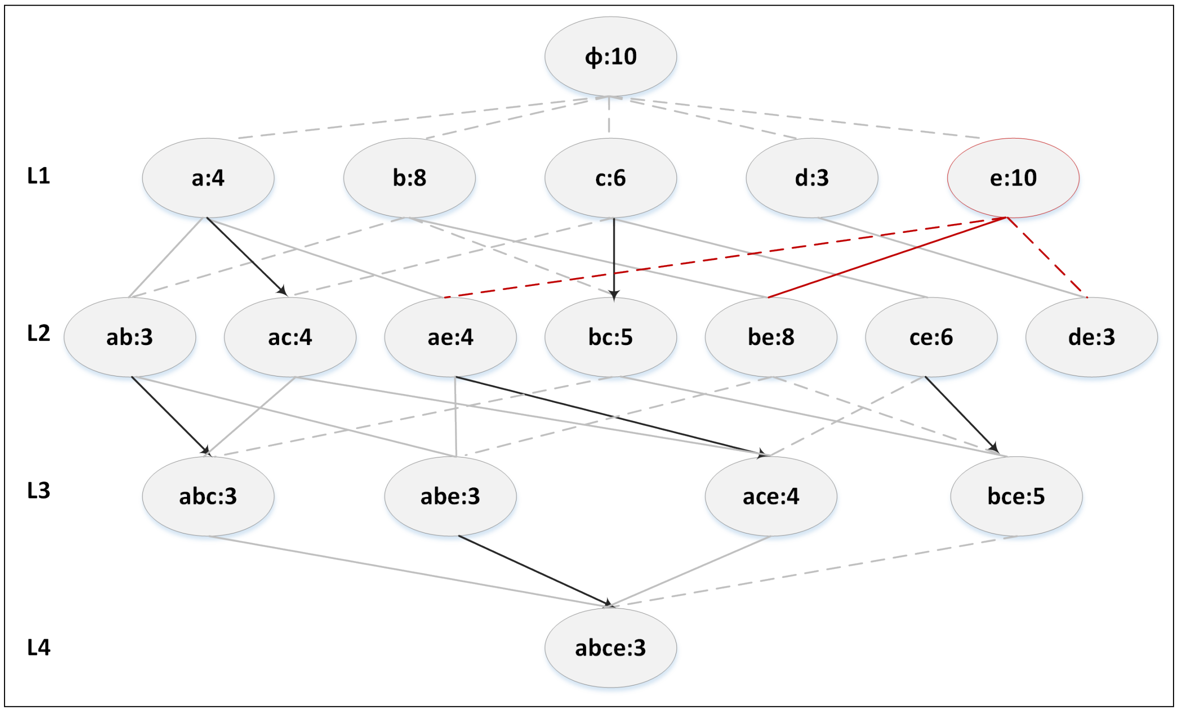


Figure 5.2: Partial lattice of frequent itemsets with $[n-1]$ rules

The relationships between various items in an itemset are often presented using a lattice [38]. A lattice is a fundamental and general algebraic structure used to represent a partially-ordered set that is often drawn using Hasse diagram[1]. A lattice is denoted by $\langle L, \vee, \wedge \rangle$ where L is a non-empty set that supports binary OR and binary AND operations over L. From an alphabet $\mathcal{A}$, a lattice can be

[1]https://en.wikipedia.org/wiki/Hasse_diagram

derived using a partially-ordered set $(L, \preceq)$ by considering $X \wedge Y = X \cap Y$ and $X \vee Y = X \cup Y$ for any $X, Y \in L$. In lattice theory, $X \wedge Y$ is called infimum, meet or greatest lower bound. The $X \vee Y$ is called supremum, join or least upper bound. Hence, the lattice for $\mathcal{A}$ contains all possible subsets of $\mathcal{A}$. For our use, a full lattice representation with all feasible itemsets is very large, memory consuming and unnecessary. Thus, we tent to prune the lattice by removing itemsets that do not meet the support threshold minimum requirement. The resulting structure is called a partial lattice or meet-semilattice [38]. A partial lattice only respects the greatest lower bound constraint of a full lattice.

The Figure 5.2 depicts a partial lattice of itemsets and their support in the first sliding window from the example presented in Figure 2.1. The partial lattice represents itemsets with $s_{min} = 3$ and reflects association with $c_{min} = 0.7$. The root is represented using $\emptyset$ (as a common practice). The first level, level 1 $(L1)$, represents the alphabet single items in a strict prefix order. Each of the subsequent levels maintains itemsets of the same size of each level using the same prefix order along with the itemsets' support in the current sliding window. For instance, $(L4)$ keeps itemsets with four items, which is $\{abce\}$ in Figure 5.2. Black continuous arrows show the $[1-1]$ and $[n-1]$ interesting association rules between two successive levels. The solid gray lines show association rules that did not satisfy the lift constraint (iii). The dotted lines express failure to form any association rule because constraint (ii) was not satisfied. In the Figure 5.3, we evaluate feasible $[1-n]$ and $[n-n]$ interesting rules similarly between every other levels among frequent itemsets that satisfy constraints (i), (ii) and (iii) using green continuous arrows. Finally, we present a $[1-n]$ interesting rule $(a \xrightarrow{l} bce)$ between $(L1)$ and $(L4)$. Figure 5.2 and Figure 5.3 depict two $[1-1]$ interesting rules $a \xrightarrow{l} c$ and $c \xrightarrow{l} b$; four $[n-1]$ interesting rules: $ab \xrightarrow{l} c$, $ae \xrightarrow{l} c$, $ce \xrightarrow{l} b$ and $abe \xrightarrow{l} c$, two $[n-n]$ interesting rules: $ab \xrightarrow{l} ce$ and $ae \xrightarrow{l} bc$, and four $[1-n]$ interesting rules: $a \xrightarrow{l} bc$, $a \xrightarrow{l} ce$, $c \xrightarrow{l} be$ and $a \xrightarrow{l} bce$.

It is indicated that all non-frequent itemsets and their supersets cannot form association rules (Apriori property). Therefore, if the support of an itemset is found to be lower than the minimum support, no superset of that itemset is investigated or maintained. In addition, omnipresent itemsets, such as item e marked with a red border, cannot directly help in generating association rules with lift greater than 1, constraint (iii) (Property 2). However, supersets of e should be evaluated as

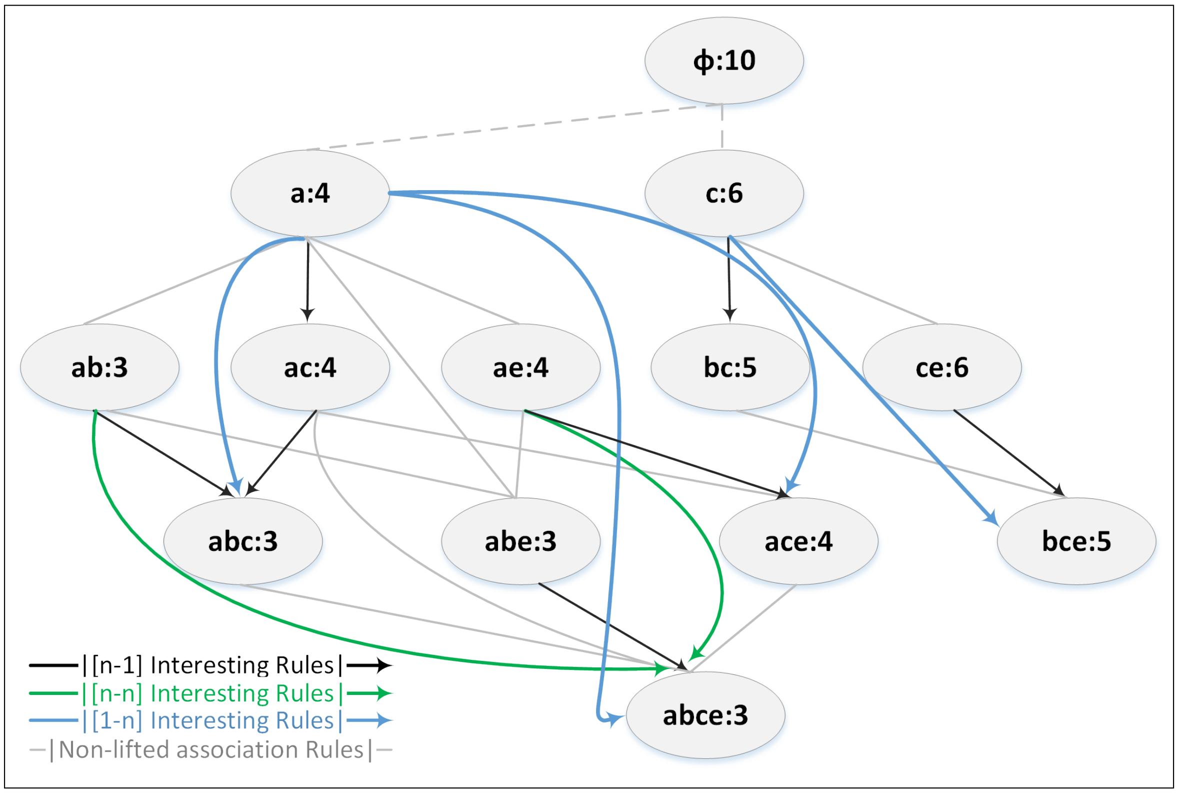

ϕ:10
a:4
c:6
ab:3
ac:4
ae:4
bc:5
ce:6
abc:3
abe:3
ace:4
bce:5
abce:3
|[n-1] Interesting Rules|
|[n-n] Interesting Rules|
|[1-n] Interesting Rules|
|Non-lifted association Rules|

Figure 5.3: Interesting $[n-1]$, $[1-n]$ and $[n-n]$ association rules with $(lift > 1)$

antecedent or consequent of rules if they are not omnipresent. Furthermore, it can be noted that the support values of a, c and d are less than 7 ($\tau \times c_{min}$). Association rules having consequent a, c and d or any of their supersets will have *lift* > 1 (Property 7). Over a partial lattice, itemsets and rules can be accessed and searched using breadth-first, depth-first or a hybrid search. In MAREDS, we use a hybrid depth-first search of rules since it can take advantage of the properties during the traversal to minimize the search time as much as possible.

5.1.2 Incremental Association Rule Update

Incremental update of the mining outcome could be the most import phase in any stream data mining. The general requirements of incremental update of association rule mining over data streams are stated in Table 5.1. The incremental update process examines two mutually exclusive itemsets namely X and Y along with their joint appearance, denoted by XY, in the current sliding window. The table states twelve update setting, $U1 - U12$. The settings are extracted based on the increase and/or decrease of support values of X, Y and XY. In addition, it considers whether the itemsets already formed any interesting association rule ($X \xrightarrow{l} Y$) or not ($X \overset{l}{\nrightarrow} Y$). As the sliding window progresses, support of (X), (Y) and (XY) will either increase by 1, decrease by 1 or remain the same. Assuming X, Y and XY are frequent in the current and the next sliding windows, each column reflects the evaluation requirements to maintain the current condition. If the given setting is true, then the rule becomes invalid in that condition. Support increment and decrement are denoted by ($\uparrow$) and ($\downarrow$), respectively. The symbol c denotes the association rule confidence.

Example 5.1. Assume there exists an interesting rule ($X \xrightarrow{l} Y$) in the current sliding window τ_j. If the support of X decreases in the next sliding window τ_{j+1} while the support of Y and XY remain the same, the update setting $U1$ in Table 5.1 states that this existing interesting rule maintains its validity in the new window and does not require any further evaluation. However, if the support of X increases in the next sliding window while the support of Y and XY remain the same, the existing interesting rule requires two further evaluations as per $U2$. If the new support of X is less than ($\frac{c_{min}}{c - c_{min}}$) or if the support of Y is less or equal to ($\tau c - \frac{\tau c}{sup_j(X)+1}$), then in the next sliding window τ_{j+1} the rule ($X \xrightarrow{l} Y$) is no longer valid.

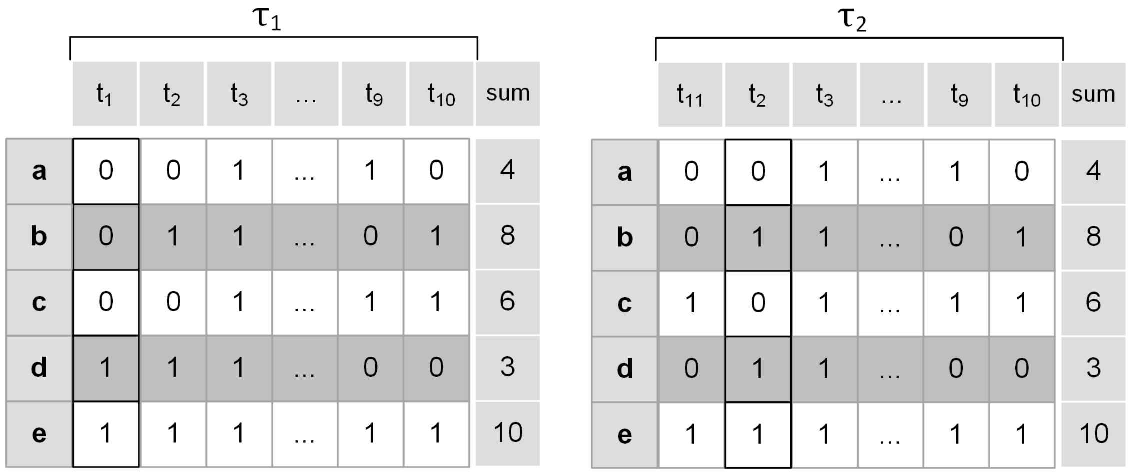
Table 5.1: Updating selected rules with the change of support

	Condition:	$X \overset{l}{\not\to} Y$	$X \xrightarrow{l} Y$
	Setting	Update Requirement	Update Requirement
$U1$	$(X) \downarrow$	$\left(c \geq c_{min} \textit{ or } sup_j(X) \leq \frac{c_{min}}{c_{min}-c}\right)$ *and* $\left(sup_j(Y) < \tau c + \frac{\tau c}{sup_j(X)-1}\right)$	Rule unchanged
$U2$	$(X) \uparrow$	Rule unfeasible	$\left(sup_j(X) < \frac{c_{min}}{c-c_{min}}\right)$ *or* $\left(\tau c - \frac{\tau c}{sup_j(X)+1} \leq sup_j(Y)\right)$
$U3$	$(Y) \downarrow$	$\left(c \geq c_{min}\right)$ *and* $\left(sup_j(Y) < \tau c + 1\right)$	Rule unchanged
$U4$	$(Y) \uparrow$	Rule unfeasible	$\left(\tau c - 1 \leq sup_j(Y)\right)$
$U5$	$(X, XY) \downarrow$	Rule unfeasible	$\left(sup_j(X) < \frac{1-c_{min}}{c-c_{min}}\right)$ *or* $\left(\tau c - \frac{\tau(1-c)}{sup_j(X)-1} \leq sup_j(Y)\right)$
$U6$	$(X, XY) \uparrow$	$\left(c \geq c_{min} \textit{ or } sup_j(X) \leq \frac{1-c_{min}}{c_{min}-c}\right)$ *and* $\left(sup_j(Y) < \tau c + \frac{\tau(1-c)}{sup_j(X)+1}\right)$	Rule unchanged
$U7$	$(Y, XY) \downarrow$	Rule unfeasible	$\left(sup_j(X) < \frac{1}{c-c_{min}}\right)$ *or* $\left(\tau c + 1 - \frac{\tau}{sup_j(X)} \leq sup_j(Y)\right)$
$U8$	$(Y, XY) \uparrow$	$\left(c \geq c_{min} \textit{ or } sup_j(X) \leq \frac{1}{c_{min}-c}\right)$ *and* $\left(sup_j(Y) < \tau c - 1 + \frac{\tau}{sup_j(X)}\right)$	Rule unchanged
$U9$	$(X) \downarrow$ & $(Y) \uparrow$	$\left(c \geq c_{min} \textit{ or } sup_j(X) \leq \frac{c_{min}}{c_{min}-c}\right)$ *and* $\left(sup_j(Y) < \tau c - 1 + \frac{\tau c}{sup_j(X)-1}\right)$	$\left((sup_j(X) > sup_j(Y) + 1) \textit{ and } (\tau c - 1 + \frac{\tau c}{sup_j(X)-1} \leq sup_j(Y))\right)$
$U10$	$(X) \uparrow$ & $(Y) \downarrow$	$\left(sup_j(X) \geq \frac{c_{min}}{c-c_{min}}\right)$ *and* $\left(sup_j(Y) < \tau c + 1 - \frac{\tau c}{sup_j(X)+1}\right)$	$\left(sup_j(X) < \frac{c_{min}}{c-c_{min}}\right)$ *or* $\left((sup_j(X) < sup_j(Y) - 1) \textit{ and } (\tau c + 1 - \frac{\tau c}{sup_j(X)+1} \leq sup_j(Y))\right)$
$U11$	$(X, Y, XY) \downarrow$	$\left(c \geq c_{min} \textit{ and } sup_j(X) \geq \frac{1-c_{min}}{c-c_{min}}\right)$ *and* $\left(sup_j(Y) < \tau c + 1 - \frac{\tau(1-c)}{sup_j(X)-1}\right)$	$\left(sup_j(X) < \frac{1-c_{min}}{c-c_{min}}\right)$ *or* $\left(\tau c + 1 - \frac{\tau(1-c)}{sup_j(X)-1} \leq sup_j(Y)\right)$
$U12$	$(X, Y, XY) \uparrow$	$\left(c \geq c_{min} \textit{ or } sup_j(X) \leq \frac{1-c_{min}}{c_{min}-c}\right)$ *and* $\left(sup_j(Y) < \tau c - 1 + \frac{\tau(1-c)}{sup_j(X)+1}\right)$	$\left(\tau c - 1 + \frac{\tau(1-c)}{sup_j(X)+1} \leq sup_j(Y)\right)$

It is noticed that τ and c_{min} are predefined fixed values while the support for X, support of Y and *conf* are changing variables from one sliding window to another. Table 5.1 provides clear instructions of whether these variables require further evaluation with every progression of the sliding window. In the upcoming sections, we use the twelve update setting to devise a single pass traversal process over a stored partial lattice in order to reveal interesting association rules.

5.2 Transaction Representation and Itemset Scanning

Dataset transactions can be stored in horizontal or vertical layout. In a horizontal layout, each row represents a transaction of items. Such layout is often incorporated by Apriori-like algorithms [59]. In the vertical layout, each row represents a single item. It encloses the item's occurrences in every transaction over the current sliding window. The occurrences may be maintained as either transaction IDs or as a bit string. Algorithms using vertical layout generally perform faster than horizontal ones when the sliding window size is relatively large [59].



τ_1

	t_1	t_2	t_3	...	t_9	t_{10}	sum
a	0	0	1	...	1	0	4
b	0	1	1	...	0	1	8
c	0	0	1	...	1	1	6
d	1	1	1	...	0	0	3
e	1	1	1	...	1	1	10

τ_2

	t_{11}	t_2	t_3	...	t_9	t_{10}	sum
a	0	0	1	...	1	0	4
b	0	1	1	...	0	1	8
c	1	0	1	...	1	1	6
d	0	1	1	...	0	0	3
e	1	1	1	...	1	1	10

Figure 5.4: Transactions in a bit matrix over sliding window

In MAREDS, we use a *bit matrix* to store the current sliding window transactions in a vertical layout. Figure 5.4 depicts an incremental insertion of transactions from Figure 2.1 using the bit matrix. Occurrences of each item in the sliding window are stored as a separate bit array. Items are placed in a strict predefined order as per the alphabet (e.g. a, b, $\cdots$, e). Each column in the bit matrix represents one transaction. The rows or cells of a column represent items in the corresponding transaction. A cell value of 1 denotes the presence of the corresponding item in that transaction. On the contrary, a cell value of 0 denotes that the item is absent in that transaction. A new transaction is stored in the column indicated by a *sliding pointer*. In Figure 5.4, the sliding pointer is marked in dark black squares borders. Once all columns in a sliding window are filled, a new transaction replaces the oldest one. In this case, the sliding pointer is placed at the oldest transaction. The corresponding column's cells are reset according to the new transaction items.

The bit matrix is used to compute momentary support of an itemset. The computation procedure is called *Scanning*. To find support of an input itemset, scanning generates a new bit array of the same size of the sliding window. Then, it uses bit-wise "AND" operation among the rows corresponding to every item from the queried itemset. The support value will be the total number of $1s$ in the output bit array. Scanning can either be performed sequentially for every row or hierarchically. When the sliding window size is large, multiple bit arrays store the occurrences of an item. Thus, serially counting support can be slower. Hierarchical counting saves execution time and reaches time-complexity of approximately $O(log(\tau))$ using parallel processing. The support values of all single items are kept in a separate integer array, namely "sum". The array is updated with the sliding window progression.

The aforementioned scanning process is efficient but it does not keep track of any momentary support of any itemset beyond the scanning procedure. Thus, with every window update, it requires to apply Apriori or similar technique(s) to extract frequent itemsets and thereafter computing association rules. Such traditional mining techniques is extremely time consuming with respect to the stream speed. For an alphabet $\mathcal{A}$ of size n, the total number of possible itemsets and association rules is ($2^n - 1$) and ($3^n - 2^{n+1} + 1$), respectively. In contrast, storing momentary support of itemsets offers faster computation of association rules. In this regard, we propose maintaining selective itemsets and their support values in a prefix tree variant which is discussed in the next Section 5.3.

5.3 Partial Association Enumeration Tree

In this section, we introduce the *Partial Association Enumeration Tree (PAET)*. PAET is as a prefix tree variant that mines and maintains relevant itemsets. It offers prompt search for momentary interesting association rules.

Definition 5.1. (Partial Association Enumeration Tree). Over a sliding window of size τ, given a strictly ordered set of alphabet $(\mathcal{A}, \prec)$ and a root node $\langle \emptyset, \tau, - \rangle$, PAET can be defined recursively as a collection of nodes starting from the root. Each PAET node $n_X \in \Psi$ is a triplet, denoted by $\langle X, sup_j(X), n_{X \setminus \{e\}} \rangle$. It consists of a itemset X, the node label, where $X \in \mathcal{P}(\mathcal{A}) \setminus \{\emptyset\}$, its support $sup_j(X)$ in the current sliding window τ_j and a pointer to a parent node $n_{X \setminus \{e\}}$. A tree node n_X satisfies one of the following:

- $|X| = 1$: Node holds a single item.
- $|X| > 1$: Node holds an itemset with two or more items and :
    - $sup_j(X) \geq s_{min} - 1$: node support is greater or equal to the (minimum support-1).
    - has a pointer $n_{X \setminus \{e\}}$ such that $\forall e, e' \in \mathcal{A}$, $sup_j(X \setminus \{e\}) \leq \min\limits_{e' \in \mathcal{A}} sup_j(X \setminus \{e'\})$

$\mathcal{P}(\mathcal{A})$ denotes the powerset of the alphabet $\mathcal{A}$, The symbols $\prec$ and $\preceq$ indicate strict and partial ordering respectively. PAET keeps all single items in the first level immediately below the root and thus reflects the "sum" array stated in Section 5.2). Additionally, PAET keeps itemset nodes where current itemset support is greater or equal to ($s_{min} - 1$). The node n_X associated pointer links the node to one of its parent nodes which has the lowest support value. Figure 5.9 depicts PAET generation for the first sliding window of Figure 2.1.

5.3.1 PAET Construction

Stream association rule mining over a sliding window requires fast access to previous mining outcome to avoid any possible delay. Hence, PAET is constructed in memory to provide the MAREDS algorithm with rapid access to its nodes (itemsets) [12]. The tree design and node selection require special attention to have a smooth update process where nodes are inserted and deleted incrementally. The core challenge lies in constructing the tree to efficiently fetch selected association rules for every new window. To analyze a selected association rule ($X \xrightarrow{l} Y$), momentary support of nodes X, Y or XY should be accessed from PAET. Otherwise, additional scanning is required to compute the support values in order to validate the rule for every new window. This will make the overall rule generation and update process slower. Hence, we propose inserting additional tree nodes representing necessary itemsets at least one transaction before they can possibly meet the first constraint (i) of rule generation. Chi et al. term these nodes as gateway nodes [12]. Below we describe node insertion and deletion processes into and from PAET:

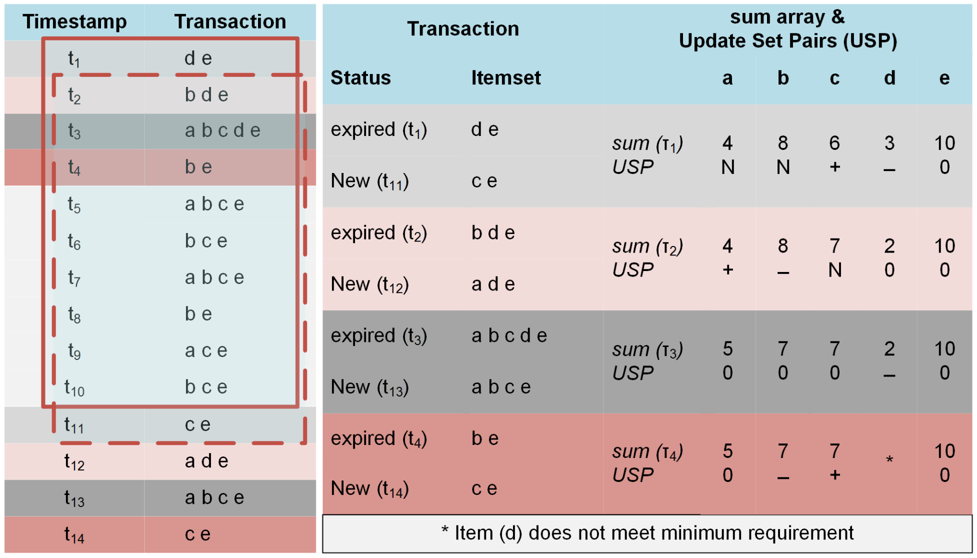
- *Node Insertion*: Node n_X corresponding to itemset X is newly created and inserted into PAET if X is a subset of the new incoming transaction. It should have momentary support greater or equal to $s_{min} - 1$, i.e. $(sup_j(X) \geq s_{min} - 1)$. This process is referred to as *leaf insertion*. If a new itemset with two or more items, $|X| > 1$, is inserted into PAET, it generates a series of consecutive leaf insertions. This is referred to as *branch insertion*. The support of each new node is required to be evaluated. Property 6-a. is used to reduce the total number of evaluations using the scanning process stated in the previous Section 5.2.

- *Node Deletion*: Node n_X corresponding to itemset X is deleted from PAET if its momentary support is reduced in the current sliding window and becomes lower than the threshold of $s_{min} - 1$, i.e. $(sup_j(X) < s_{min} - 1)$. If the node n_X is a leaf, the process is referred to as *leaf deletion*. If the node is not a leaf, the whole branch of superset nodes below n_X are deleted as all these nodes do not meet the requirement threshold as per Apriori rule. Such deletion is referred to as *branch deletion*.

The proposed node insertion and deletion differ from other common approaches of stream frequent itemset mining [12, 65, 10, 39, 49]. An itemset is stored in PAET if and only if it can possibly be an antecedent or consequent of an association rule when the next transaction is available. Leaf insertion or deletion deals with one gateway node while branch insertion or deletion handles multiple gateway nodes at every transaction.

5.3.2 Maximum Confidence Analysis

The confidence of an association rule is impacted by the change of support of its antecedent itemset and the support of the joint appearances of the antecedent and the consequent itemsets. Therefore, we propose that for each node n_X in PAET, in addition to tracking its support, we track the itemset's parent subset nodes. To be able to track these nodes, we use the pointer in each PAET



Timestamp	Transaction
t_1	d e
t_2	b d e
t_3	a b c d e
t_4	b e
t_5	a b c e
t_6	b c e
t_7	a b c e
t_8	b e
t_9	a c e
t_{10}	b c e
t_{11}	c e
t_{12}	a d e
t_{13}	a b c e
t_{14}	c e

Transaction		sum array & Update Set Pairs (USP)					
Status	Itemset		a	b	c	d	e
expired (t_1)	d e	*sum* (τ_1)	4	8	6	3	10
New (t_{11})	c e	*USP*	N	N	+	–	0
expired (t_2)	b d e	*sum* (τ_2)	4	8	7	2	10
New (t_{12})	a d e	*USP*	+	–	N	0	0
expired (t_3)	a b c d e	*sum* (τ_3)	5	7	7	2	10
New (t_{13})	a b c e	*USP*	0	0	0	–	0
expired (t_4)	b e	*sum* (τ_4)	5	7	7	*	10
New (t_{14})	c e	*USP*	0	–	+		0

* Item (d) does not meet minimum requirement

Figure 5.5: Single-pass PAET update: (sliding widow, sum array, USP)

node to point to the parent node with the least minimum support value among all its parent nodes. Thus, each of these pointers identify a $[n-1]$ association rule with maximum confidence, if existing. We define a *Maximum Confidence Rule (MCR)* as follows.

Definition 5.2. ($[n-1]$ Maximum Confidence Rule). An association rule $(X \rightarrow Y)$ is called $[n-1]$ maximum confidence rule of XY if and only if $X \subset XY$, $|XY| - |X| = 1$ and $conf_j(X \rightarrow Y)$ has maximum value of confidence among any subset of XY.

Example 5.2. Let us consider itemset ace of cardinality 3 in Figure 5.9. In PAET, the node's parents of cardinality 2 are ac, ae and ce with support values of 4, 4 and 6 respectively. Therefore, either $(ac \rightarrow e)$ or $(ae \rightarrow c)$ can be considered as a $[n-1]$ MCR for itemset ace. Similarly, $(c \rightarrow b)$ is an $[n-1]$ MCR for itemset bc. The concept can be extended such that $(ab \rightarrow ce)$ is a $[n-2]$ MCR for $abce$.

The $[n-1]$ MCR use during the sliding window update is very important. In the following, we highlight the conditions that reflect that importance.

(C1) *If* $[n-1]$ *MCR for a PAET node* n_X *is not a valid association rule, because it does not meet the minimum confidence threshold, then no association rule can be formed with any of* n_X *other parent nodes.*

(C2) *If* $sup_j(X) = sup_j(XY)$ *then* ($X \to Y$) *is* n_{XY} *node's* $[n-1]$ *MCR unless there already exists a* $[n-1]$ *MCR,* ($X \to Y'$)*, where* $sup_j(X) = sup_j(XY')$.

(C3) *If* ($X \to Y$) *is a* $[n-1]$ *MCR, an increase in support value of any subset of* XY *(except for* X*),* ($X \to Y$) *holds as* $[n-1]$ *MCR.*

(C4) *If* ($X \to Y$) *is a* $[n-1]$ *MCR, a decrease in support value of* X *will hold* ($X \to Y$) *as* $[n-1]$ *MCR.*

(C5) *If* ($X \to Y$) *is a* $[n-1]$ *MCR for a PAET node* n_{XY}*, then if the parent nodes either all increase, all decrease or all have no change in the support value, then* ($X \to Y$) *will still hold as* $[n-1]$ *MCR.*

5.3.3 PAET Update

Upon arrival of a new transaction, it is inserted into the sliding window whereas the oldest transaction is deleted from the window. PAET is updated accordingly as well. We propose the *single-pass* update process where all itemsets and interesting association rules are updated all at once. In single-pass, the itemsets' updates are reflected in PAET with only one single tree traversal having minimal node visits as possible. The update operation in other surveyed algorithms in the literature review (Chapter 3) usually consists of two operations: addition and deletion. The two-operations model requires at least two data access passes over the used data structure to perform the update task. Moreover, an itemset addition may take place, even though the same itemset may be removed by the later delete operation. This creates unnecessary data structure access, not to mention unnecessary operations which are extremely costly in a stream environment.

Let us denote the oldest transaction in the sliding window as ξ_- and the incoming new transaction as ξ_+. The powerset $\mathcal{P}(\xi_-)$ indicates the set of all sets affected by the expiring transaction. For example, in Figure 5.5, at sliding window τ_2, ξ_- represents the expiring transaction $\{de\}$. Therefore, $\mathcal{P}(\xi_-) = \{\emptyset, \{d\}, \{e\}, \{de\}\}$ and similarly $\mathcal{P}(\xi_+) = \{\emptyset, \{c\}, \{e\}, \{ce\}\}$. The momentary support values for ξ_- itemsets should decrease, while the momentary support values for ξ_+ should increase. With a simple analysis, we note that there is no change in the support values for all itemsets represented by $\mathcal{P}(\xi_- \cap \xi_+)$. In addition, only itemsets denoted by $\mathcal{P}(\xi_+) \setminus \mathcal{P}(\xi_- \cap \xi_+)$ have

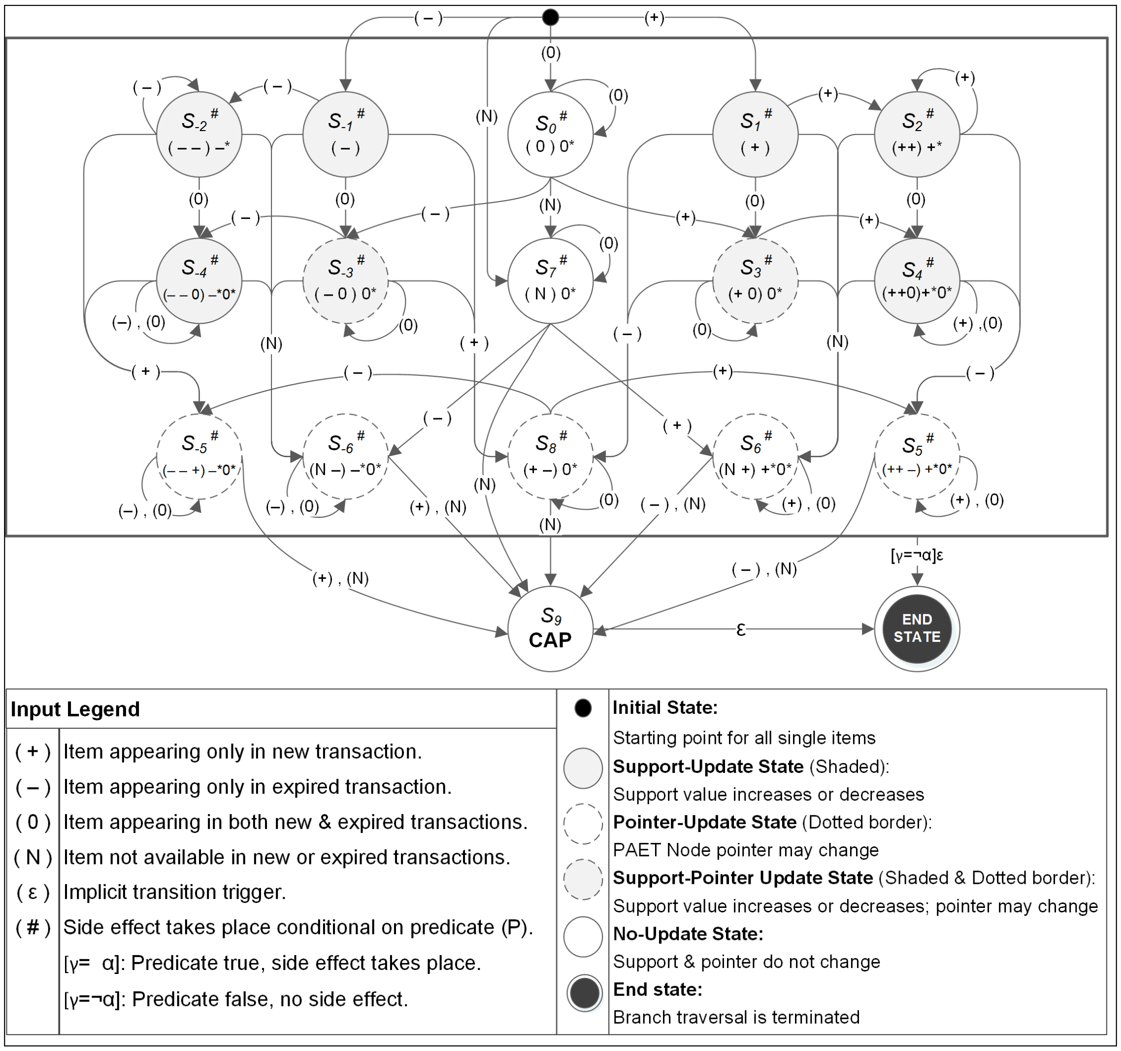

S₋₂# (− −) −*
S₋₁# (−)
S₀# (0) 0*
S₁# (+)
S₂# (++) +*
S₋₄# (− − 0) −*0*
S₋₃# (− 0) 0*
S₇# (N) 0*
S₃# (+ 0) 0*
S₄# (++0)+*0*
S₋₅# (− − +) −*0*
S₋₆# (N −) −*0*
S₈# (+ −) 0*
S₆# (N +) +*0*
S₅# (++ −) +*0*
S₉ CAP
END STATE
[γ=¬α]ε
ε
(+) , (N)
(−) , (N)
(−) , (0)
(+) , (0)
Input Legend
(+) Item appearing only in new transaction.
(−) Item appearing only in expired transaction.
(0) Item appearing in both new & expired transactions.
(N) Item not available in new or expired transactions.
(ε) Implicit transition trigger.
(#) Side effect takes place conditional on predicate (P).
[γ= α]: Predicate true, side effect takes place.
[γ=¬α]: Predicate false, no side effect.
Initial State: Starting point for all single items
Support-Update State (Shaded): Support value increases or decreases
Pointer-Update State (Dotted border): PAET Node pointer may change
Support-Pointer Update State (Shaded & Dotted border): Support value increases or decreases; pointer may change
No-Update State: Support & pointer do not change
End state: Branch traversal is terminated

Figure 5.6: States and transitions for incremental update using *Hybrid Automaton*

momentary support increases and itemsets denoted by $\mathcal{P}(\xi_-) \setminus \mathcal{P}(\xi_- \cap \xi_+)$ have momentary support decreases. Moreover, there exists a big number of itemsets in the alphabet $\mathcal{A}$ powerset, denoted by $\mathcal{P}(\mathcal{A}) \setminus \mathcal{P}(\xi_- \cup \xi_+)$. These itemsets are not impacted by the sliding window update. Yet, some of them are important to note because they might be part of the existing association rules which may be impacted by some of the updated itemsets. PAET single pass takes place through a tree traversal that can be captured with transitions among a set of different states triggered by the above four cases. Let IP be a set of transition triggers which represent the update input for each item. IP consists of the following values:

- Increase $(+)$: update value for items appearing only in the new incoming transaction.
- Decrease $(-)$: update value for items appearing only in the expired transaction.
- No Change (0): update value for items appearing in both new and expired transactions.
- No impact (N): update value for items not available in new or expired transactions.

In Figure 5.6, we introduce a hybrid automaton that consists of fifteen different update states in order to govern PAET update process. These states reflect any possible set of trigger combinations for a given itemset. To retrieve the update state of an itemset, we start at the initial state and transit through the states based on each of the itemset IP values. Below is an example that describes state transition over a given input. Table 5.2 provides a description of the accepted input for each state in the automaton.

Example 5.3. Let us consider the example in Figure 5.5 for the sliding window τ_2. The oldest transaction $\xi_- = \{d, e\}$ and the incoming transaction $\xi_+ = \{c, e\}$. The IP values for c,d and e are $(+), (-)$ and (0), respectively. The update state for itemset $\{c, d, e\}$ is S_8 ($InitialState \xrightarrow{(+)} S_1 \xrightarrow{(-)} S_8 \xrightarrow{(0)} S_8$). Similarly, the parent itemset $\{c, e\}$ update state is S_3.

Each of the states executes a series of update actions to maintain itemsets' momentary support and track $[n-1]$ MCR. The states are categorized into four groups depending on their update actions:

- *Support-Update States*: (Member states are S_1, S_2, S_4, S_{-1}, S_{-2} and S_{-4})

  In these states, the support of itemset X of PAET node n_X changes but a pointer update is not required because all the parent nodes' support either increase or decrease (Condition C5).
- *Pointer-Update States*: (Member states are S_5, S_6, S_8, S_{-5} and S_{-6})

  In these states, the support value of itemset X of PAET node n_X does not change but the support of one or multiple parent nodes changes. This may require a re-evaluation of node n_X pointer value. Since the evaluation is computationally and memory expensive during the search, we further examine the requirements for incremental update to eliminate unnecessary evaluations:

    - In states (S_5) and (S_6) only one of the parent nodes has an increase in the support value. If the pointer of n_X points to that node, it is required to investigate all other n_X parent nodes to find the parent node with the lowest support value. Otherwise no pointer update is required.
    - In states (S_{-5}) and (S_{-6}) only one of the parent nodes has a decrease in the value. If the pointer of (n_X) does not point to that specific node, it is required to compare the current parent node support against it to find the least support value of both parent nodes. Otherwise no pointer update is required.
    - In state (S_8), the pointer of node n_X may point to a parent node whose support value is either increasing, decreasing or having no change. In the first case, it is required to evaluate all parent nodes of n_X to find the lowest support value. In the second case, no evaluation is necessary and no pointer update is required. Finally, in the third case, it is required to compare the support of currently pointer parent node with the support of that specific parent node whose support is decreasing to find out the if a pointer update is needed or not.

- *No-Update States*: (Member states are S_0, S_7 and S_9)

  In these states, the support value of itemset X of the PAET node n_X does not change and the node's pointer does not require any update. All items in X have update values of (0) and/or (N).

- *Support-Pointer Update States*: (Member states are S_3 and S_{-3})

  In these states, the support value of itemset X in the PAET node n_X changes. In addition, its node's pointer may also require further evaluation as follows.

    - In state (S_3) only one of the parent nodes has an increase in the support value. If the pointer of n_x does not point to that specific node, it is required to compare the current parent node support against it to find the least support value of both parent nodes. Otherwise no pointer update is required.
    - In state (S_{-3}) only one of the parent nodes does not have a decrease in the support value. If the pointer of n_X points to that node, it is required to investigate all other n_X parent

nodes to find the parent node with the lowest support value. Otherwise no pointer update is required.

Table 5.2: Hybrid automaton: state accepted input specification

State	Accepted Input	Description
S_0	$(0)0^*$	At least one ‘0’ or more.
S_1	$(+)$	A single ‘+’ only.
S_2	$(++)+^*$	At least two ‘+’s or more.
S_3	$(+0)0^*$	At least one ‘+’, one ‘0’ and any additional ‘0’s.
S_4	$(++0)+^*0^*$	At least two ‘+’s, a single ‘0’ and any additional ‘+’s and ‘0’s.
S_5	$(++-)+^*0^*$	At least two ‘+’s, a single ‘$-$’ and any additional ‘+’s and ‘0’s.
S_6	$(+N)+^*0^*$	At least one ‘+’, one ‘N’ and any additional ‘+’s and ‘0’s.
S_{-1}	$(-)$	A single ‘$-$’ only.
S_{-2}	$(--)-^*$	At least two ‘$-$’s or more.
S_{-3}	$(-0)0^*$	At least one ‘$-$’, one ‘0’ and any additional ‘0’s.
S_{-4}	$(--0)^*0^*$	At least two ‘$-$’s, a single ‘0’ and any additional ‘$-$’s and ‘0’s.
S_{-5}	$(--+)^*0^*$	At least two ‘$-$’s, a single ‘+’ and any additional ‘$-$’s and ‘0’s.
S_{-6}	$(-N)^*0^*$	At least one ‘$-$’, one ‘N’ and any additional ‘$-$’s and ‘0’s.
S_7	$(N)0^*$	At least one ‘N’ and any additional ‘0’s.
S_8	$(+)0^*$	At least one ‘+’, one ‘$-$’ and any additional ‘0’s.

The changes of the support values and update of $[n-1]$ MCR pointers inside PAET nodes can be formally considered as a *side-effect* of the hybrid automaton described in Figure 5.6. Once the required actions are applied, the state has to transit to end state to terminate. Hence, we add an additional transition trigger (ε) to IP in order to describe this implicit transition from any state to the end state. Thus, the IP set is denoted as $\{+,-,0,N,\varepsilon\}$. Every state transition is a tuple $\langle s_i,\gamma,\alpha,e,s_j\rangle$ corresponding to a move from state (S_i) to state (S_j) based on an transition trigger $e \in IP$. The destination state (S_j) checks if the underlying predicate ($\gamma = \alpha$) is true. If so, the state side effect takes place if required as mentioned above. On the contrary, if ($\gamma = \neg\alpha$) is true, no side effect takes place at (S_j). In Figure 5.6, the automaton includes 15 states (described in Table 5.2) which process these transition tuples. In addition, the automaton includes one state (S_9), where no action is required. This state is denoted as *CAP State*. If an itemset X update request reaches the CAP state, non of X supersets have any update nor they can form any new association rules. Then, the state implicitly transits to end state without any predicate checking. In the following, we provide an example that illustrates the traversal and update through PAET using this automaton.

Example 5.4. Let us consider the example in Figure 5.5 for the sliding window τ_2. We construct the *update set pairs (USP)* as: $\{(a, N), (b, N), (c, +), (d, -), (e, 0)\}$. Then, in a depth-first search manner, we start traversing the first PAET branch as described in Figure 5.9 starting at node $a \Rightarrow ab \Rightarrow abc \Rightarrow abce$. The first sequence of inputs is $\langle([\gamma = \alpha], N), ([\gamma = \alpha], N), ([\gamma = \alpha], +), ([\gamma = \alpha], 0), ([\gamma = \neg\alpha], \varepsilon)\rangle$. For simplicity, we abbreviate the inputs as $\langle[\alpha]N[\alpha]N[\alpha]{+}[\alpha]0[\neg\alpha]\varepsilon\rangle$. Each of the transitions is guarded by a predicate and holds an input value from IP. The input sequence is tailed by $[\neg\alpha]\varepsilon$ at the end of every input sequence. However, in this example, the corresponding state transitions finish after three steps. The fist step from initial state to (S_7) accepting $([\gamma = \alpha], N)$. The second step from (S_7) to (S_9) accepting $([\gamma = \alpha], N)$. Finally, transit implicitly from (S_9) to end state irrespective of the rest of the input sequence. In all visited states, required side effects take place if available. The second sequence of inputs: $\langle[\neg\alpha]N[\alpha] + [\alpha]0[\neg\alpha]\varepsilon\rangle$ is derived from the first input sequence. It corresponds to searching $a \Rightarrow ac \Rightarrow ace$. We note that PAET node n_A is preceded by a predicate [$\gamma = \neg\alpha$], and hence will not have a side effect since it was already processed by the first input sequence. The depth-first search continues until node n_E is reached.

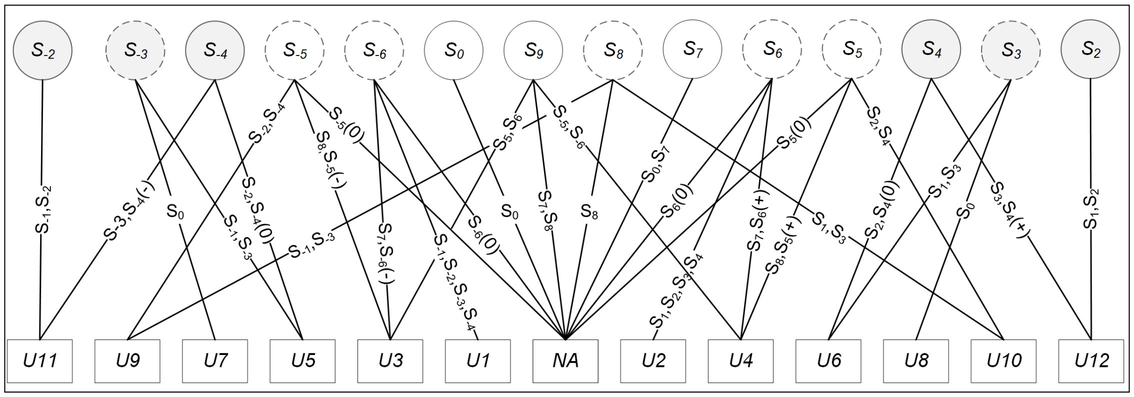


Figure 5.7: Selection of update settings for state transitions

5.3.4 $[n - 1]$ Association Rule Tracking

The preceding hybrid automaton simplifies the actions required for PAET update and rule generation during the traversal process. Figure 5.7 presents a bipartite graph that describes the links between the automaton states and the update settings in Table 5.1. The set of states, placed at the top, represent the recipient states. The set of update settings at the bottom presents interesting rule required evaluations. The edges present a collection of transitions among the automaton states and the corresponding update requirements. The update setting *“NA”* denotes that no action is required. The exact choice of update settings for some of the transitions depends on both previous state and the received input.

Example 5.5. State (S_3) can be reached from states (S_0), (S_1) and (S_3). If (S_3) was reached from (S_0), it requires evaluation using update setting $U8$. While, if (S_3) was reached from (S_1) or (S_3), it requires evaluation using update setting $U6$.

Example 5.6. State transition from (S_4) to same state (S_4) depends on provided input. An input of $[\alpha](0)$ or $[\alpha](-)$ leads to update setting $U6$ or $U12$, respectively.

In $U12$, the support values of antecedent X, consequent Y and their joint appearance XY are all increased by 1, as identified in Table 5.1. A further analysis of $U12$ reveals that, if any of the two conditions, $\Big(c \geq c_{min}$ or $sup_j(X) \leq \frac{1-c_{min}}{c_{min}-c}\Big)$, is $true$ then new association rules can be created. However, in order to identify an interesting association rule with (*lift* > 1), another condition, $\Big(sup_j(Y) < \tau c - 1 + \frac{\tau(1-c)}{sup_j(X)+1}\Big)$, has to be satisfied also. If the first two conditions are not satisfied, no association rule exists between the specific itemsets. If only the last condition does not hold, a strong association rule exists but it is no longer interesting. As for the existing rules, they remain valid between itemsets corresponding to the originating state (S_3 or $S_4(+)$) and the recipient state S_4. But they require further evaluation, $\Big(\tau c - 1 + \frac{\tau(1-c)}{sup_j(X)+1} \leq sup_j(Y)\Big)$, to check if the rules are interesting or not.

In Table 5.1, the big brackets $\Big(\dots\Big)$ split the evaluation requirements for a given update setting U_i between both *conf*$_j(\dots)$ and *lift*$_j(\dots)$ functions. In Table 5.3, we show at each state what is the exact needed evaluation actions for association rule generation or maintenance. The table is divided into two sections. The *existing rules* section, where a $[n-1]$ association rule (interesting or only strong) exists between corresponding itemsets. Second, the *non-existing rules* section, where a $[n-1]$ association rule does not meet the minimum confidence threshold (constraint (ii)). The column “Current State” denotes the present state of the single pass update process. The column “Consequent Support Update” denotes the update input, such as increase $(+)$, decrease $(-)$, no

Table 5.3: Requirements for incremental evaluation of *confidence* and *lift*

Line No.	Current State	Consequent Support Update	Evaluation Requirements $conf_j(\dots)$	$lift_j(\dots)$
	Existing Rules			
1.	S_2	Any	No	Yes
2.	S_{-2}	Any	Yes	Yes
3.	S_3	Any	No	if ($lift_{j-1}(\dots) \leq 1$)
4.	S_{-3}	Any	Yes	if ($lift_{j-1}(\dots) > 1$)
5.	S_4	(0)	No	if ($lift_{j-1}(\dots) \leq 1$)
6.	S_{-4}	(-)	Yes	Yes
7.	S_{-4}	(0)	Yes	if ($lift_{j-1}(\dots) > 1$)
8.	S_4, S_8, S_{-5}	(+)	No	Yes
9.	S_5, S_6, S_9	(+)	No	if ($lift_{j-1}(\dots) > 1$)
10.	S_5, S_8	(-)	Yes	Yes
11.	S_6	(N)	Yes	if ($lift_{j-1}(\dots) > 1$)
12.	S_{-6}	(N)	No	if ($lift_{j-1}(\dots) \leq 1$)
13.	S_9, S_{-5}	(-)	No	if ($lift_{j-1}(\dots) \leq 1$)
	Non-Existing Rules			
14.	S_{-5}, S_8	(+)	Yes	Yes
15.	S_{-6}	(N)	Yes	Yes
16.	S_2, S_3, S_4	Any	Yes	Yes

change (0) or no impact (N) for the rule consequent itemset. It should be noted that, for all $[n-1]$ association rules, consequent is an itemset consisting of a single item only. Subsequently, two columns under "Evaluation Requirements" state the required confidence and list evaluations for a given state. We note that all actions take place only if ($[\gamma = \alpha]$). The following example elaborates the necessary actions.

Example 5.7. We take the same $USP = \{(a, N), (b, N), (c, +), (d, -), (e, 0)\}$ in Example 5.4. In the Figure 5.2, let us consider traversing the partial lattice starting at $b \Rightarrow bc \Rightarrow bce$. The hybrid automaton input is formed as $\langle[\alpha]N[\alpha] + [\alpha]0[\neg\alpha]\varepsilon\rangle$. Once single pass reaches node n_{bc} in PAET, the corresponding transition in the automaton is state (S_7) to state (S_6). The state (S_6) in the current example is reached with a consequent update input of $(+)$ from state (S_7) denoting the relation from b to bc. In addition, (S_6) is reached with the same input from (S_1) denoting relation between from c to bc. It is clear from the partial lattice in Figure 5.2 that ($b \nrightarrow c$) is not an association rule while ($c \xrightarrow{l} b$) is a valid interesting association rule. As stated in Table 5.3 in the non-existing rules' section, no reevaluation of confidence or lift is required for ($b \nrightarrow c$). On the other hand, the transition from state (S_1) to (S_6) invokes update setting U2. A reevaluation of both confidence and lift is required for the existing rule ($c \xrightarrow{l} b$) as stated at line no.11 in the same table. As shown in

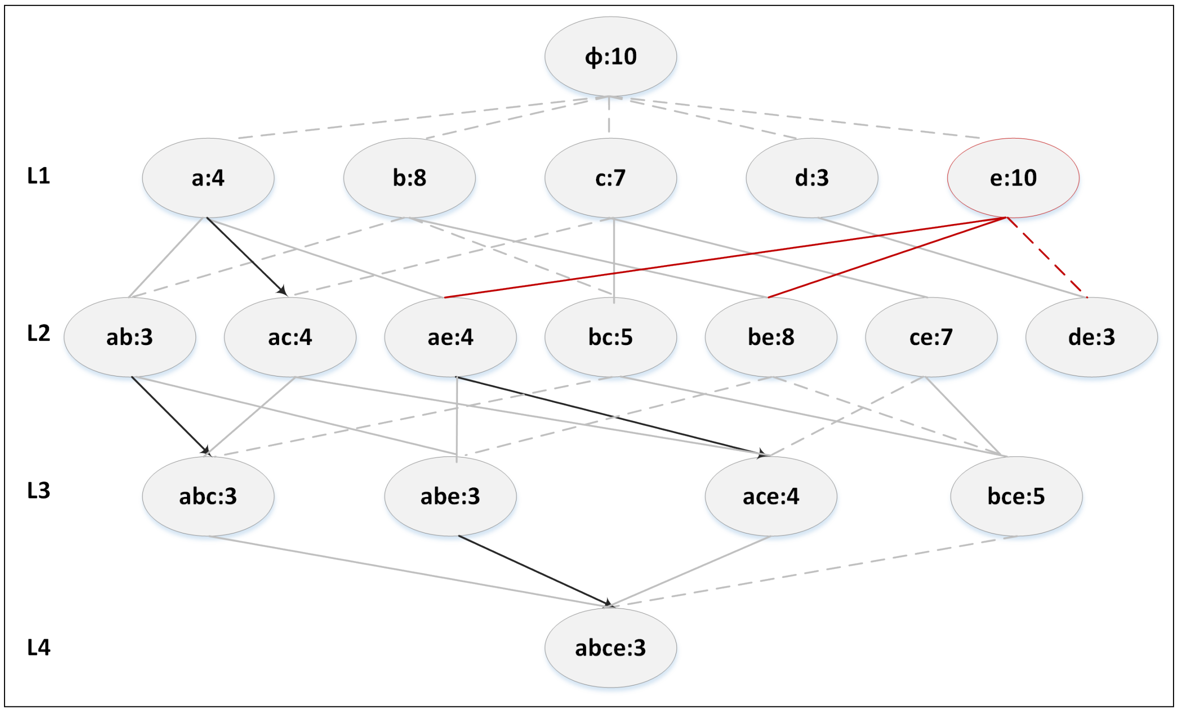


Figure 5.8: Partial lattice of frequent itemsets with $[n-1]$ rules for sliding window (τ_2)

Figure 5.8, at sliding window τ_2, the rule ($c \rightarrow b$) is a valid strong association rule but its lift is less than 1 which makes it no longer an interesting rule. Similarly, when the search reaches node n_{bce} in PAET, the corresponding automaton state is still (S_6). The reevaluation of both confidence and lift is required for ($ce \xrightarrow{l} b$). As found in Figure 5.8, at sliding window τ_2, the rule ($ce \rightarrow b$) is a valid strong association rule but its lift is less than 1.

5.4 MAREDS Design

MAREDS uses two algorithms to capture all the $[n-n]$ association rules. The first algorithm captures incremental generation of all the $[n-1]$ association rules in the current sliding window. The second algorithm derives $[n-n]$ association rules from the first one output using a modified Apriori technique. Additional filters and acceleration techniques are deployed to eliminate unnecessary operations and quickly capture interesting rules.

5.4.1 $[n-1]$ Association Rules Generation

Algorithm 1 PAET update algorithm for incremental mining of $[n-1]$ association rules

```
Require: $\mathcal{PAET}_{t-1}$, $Stack\langle Map(itemset, parent, s_i)\rangle$
 1: Constant: $IP : \{+, -, 0, N\}$, $S : \{s_{-6}, s_{-5}, \ldots, s_0, \ldots, s_9\}$, $MIN\_SUP$, $MIN\_CONF$
 2: Known: $\xi_-$, $\xi_+$;                {//Outgoing and incoming transactions respectively}
 3: Input: $USP\langle(item, ip)\rangle$ ←genUSP($\xi_-, \xi_+, IP$)      {//$item \in \mathcal{A}$ (alphabet), $\emptyset$: root of $\mathcal{PAET}_{t-1}$}
 4: Initialize: Push each $(item, ip)$ of $USP$ at $Stack$ in reverse alphabetical order along with parent $\emptyset$ and
        state $s_{item}$
 5: Function singlePass($USP$, $Stack$) {                {//Recursive update of PAET tree}
 6: Pop top $(itemset, parent, s_i)$ from $Stack$
 7: Search corresponding node $n_{itemset}$ in $\mathcal{PAET}_{t-1}$
 8: bool ←exists($n_{itemset}$)
 9: if bool and $s_i \in \{s_1, s_2, s_3, s_4\}$ then
10:     Increase support of $itemset$ at $n_{itemset}$ by 1
11: else if bool and $s_i \in \{s_{-1}, s_{-2}, s_{-3}, s_{-4}\}$ and $sup_{t-1}(itemset) >= MIN\_SUP$ then
12:     Decrease support of $itemset$ at $n_{itemset}$ by 1
13: else if bool and $s_i \in \{s_{-1}, s_{-2}, s_{-3}, s_{-4}\}$ and $sup_{t-1}(itemset) == MIN\_SUP - 1$ then
14:     Remove node $n_{itemset}$ and all nodes in $\mathcal{PAET}_{t-1}$ representing superset of $itemset$; bool ← $false$
15: else if ¬bool and $s_i \in \{s_1, s_2, s_3, s_4\}$ and $sup_t(itemset) == MIN\_SUP - 1$ then
16:     Add new node $n_{itemset}$ in $\mathcal{PAET}_{t-1}$; bool ← $true$
17: end if
18: Update pointer for $n_{itemset}$ using Hybrid Automaton as described in Section 5.3.2
19: if confidence(getMCR($n_{itemset}$)) ≥ MIN_CONF then
20:     Update $[n-1]$ association rules incrementally using Table 5.3
21: end if
22: if bool then
23:     $childs$ ← getChildNodes($(itemset, ip), USP$)
24:     Push each $child$ of $childs$ at $Stack$ in reverse alphabetical order with parent $itemset$ corresponding state $s_{child}$
25: end if
26: if ¬ empty($Stack$) then
27:     singlePass($USP$, $Stack$)
28: end if
29: }
```

Algorithm 1 for $[n-1]$ association rules generation is initiated when the sliding window has an update. The sliding window is updated upon the arrival of a new transaction ξ_+. The oldest transaction ξ_- is removed from the window if it is full. This process forms a sequence of update pair sets (USP) using *genUSP* function in Step 3. The *genUSP* function maps every potential or frequent item in the alphabet with a value from IP based on the incoming and the outgoing transactions. A pair of $(item, ip)$ in USP denotes whether a particular item's support value is increasing, decreasing, having no change or under no effect. Afterwards, each of the USP items are pushed into a $stack$

in a reverse lexicographical order for further evaluation (Step 4). The stack structure maintains the parent of each pushed item along with its corresponding update state in the hybrid automaton. Each of the automaton states has a predefined set of update instructions which includes (i) node support value updates, (ii) $[n-1]$ MCR pointer handling, (iii) existing association rule evaluation, (iv) investigation of non-existing association rules and (v) further tree traversal directions. The function *singlePass*, Steps $(6-27)$, updates PAET through a tail-recursion to perform a selective depth-first search using the stack structure. Inside *singlePass*, the traversal starts by *popping* the topmost element out of the stack. The algorithm incrementally updates all feasible $[n-1]$ association rules. In Step 10 and Step 12, the itemset support update takes place. In Step 14 and Step 16, PAET nodes are added or deleted if required. Step 18 performs the node's pointer update while Step 20 includes the reevaluation of all $[n-1]$ association rules. The association rule evaluation is only performed when the confidence of the $[n-1]$ MRC rule of the corresponding itemset is greater than the minimum confidence threshold (Step 19). Finally, if further traversal is required for an itemset branch, the children nodes are generated using function *getChildNodes*. A subset of the children nodes is selected based on the need for further traversal using USP. Their update states are acquired using the hybrid automaton. Similar to Step 4, the selected children nodes are re-pushed into the stack in a reverse lexicographical order to ensure the depth-first search. It is noticed that the algorithm relates between the search procedure and the automaton traversal. As the depth-first search progresses, the new input performs a state transition in the automaton as well. If the automaton reaches the end state as described in Section 5.3.3, the search on the current branch is terminated. The relation between tree traversal and state transition is elaborated later in Section 5.5.

5.4.2 $[n-n]$ Association Rules Generation

The $[n-n]$ association rules generation, Algorithm 2, reflects a modified version of the Apriori rule generation technique with two modifications. First, interesting association rules are generated using the $[n-1]$ association rules from Algorithm 1 instead of frequent itemsets. The original Apriori algorithm [4] uses frequent itemsets to generate association rules. Second, the algorithm uses properties of association rules stated in Section 5.1 to determine the set of possible rules. A tail-recursion technique is used to subsequently generate all feasible $[n-n]$ rules. The algorithm starts by generating $[n-2]$ association rules from $[n-1]$ association rules. Next, $[n-3]$ association rules are generated from the $[n-2]$ rules and so on. An additional filter is applied to extract only interesting association rules. In order to accelerate the $[n-n]$ rule search, we develop an additional data structure namely *boundary*. It tracks useful information from PAET for efficient $[n-n]$ rule generation. It enables the search to perform faster, however, it requires some extra memory and periodic update for every new sliding window. Thus, its effectiveness is highly important. We maintain two different boundaries as follows:

- *Omnipresence* (B_1): B_1 helps in eliminating itemsets present in all transactions during the rule search. Property 2 states that no interesting rule can be formed considering an omnipresent itemset as an antecedent or a consequent.

- *Consequence* (B_2): B_2 determines if the consequent itemset Y has any impact over a selected association rule generation. Property 7 states that if $sup_j(Y) < \tau \times c_{min}$, then the lift of the corresponding association rule is always greater than 1 (constraint (iii)). In this case, we just need to confirm the first two rule generation constraints.

Algorithm 2 starts by storing all $[n-1]$ association rules in an array of maps namely $Rulelist$. Each map in the $Rulelist$ maintains only association rules with the same antecedent itemset length. The map at first position in the list maintains rules with antecedent of single items. The map at the second position maintains rules with antecedent itemset of two items and so on. Rules within one map are lexicographically ordered.

Example 5.8. The association rule ($ab \rightarrow c$) is maintained within the second map of the $RuleList$ since its antecedent belongs to Level 2 ($|ab| = 2$). In addition, this rule is the first element of the lexicographically ordered map given the ordering of the association rule.

After placing all the $[n-1]$ association rules into the $Rulelist$, a temporary new list is initialized (Step 4). The new list structure is similar to $RuleList$. For each $[n-1]$ rule in $RuleList$, a new candidate $[\grave{n}-2]$ rules are produced by moving a single item from the antecedent and placing it into the consequent ($\grave{n} = n-1$, Step 11).

Example 5.9. Using the same association rule ($ab \rightarrow c$), we may generate two candidate $[1-2]$ rules: ($a \rightarrow bc$) and ($b \rightarrow ac$).

The same concept applies for later generation of $[n - i]$ rules. New candidates are of form $[\grave{n} - \grave{i}]$ rules where $\grave{n} = n - 1$ and $\grave{i} = i + 1$. Subsequently, properties of the required rules including the confidence and/or lift are evaluated for every candidate rule using the *verify* function in Step 14. Rules that pass the verification process are considered as new $[n-n]$ rules. The new rules are stored and then sorted in a lexicographical order (Step 25) as they are subjected for the next level of rule evaluation. Step 26 recursively calls the $[n - n]$ rule generation function over the newly discovered rules.

It is important to mention that Algorithm 2 is not incremental. It is called with every update of the sliding window after incrementally generating the $[n - 1]$ association rules. The procedure is performed from scratch. It is possible to fully generate $[n - n]$ rules in an incremental manner by storing new association rules and removing invalid association rules. However, the process would require great memory and computing resources to track the rules. Moreover, the number of valid $[n - n]$ rules is much less than that number of $[n - 1]$ association rules. Therefore, we consider a design decision to maintain the $[n - 1]$ association rules incrementally followed by generating the small proportion of the $[n - n]$ association rules upon the sliding window update.

5.5 Case Study

MAREDS consists of several modules that work together to achieve the objective of mining association rules over data streams. In order to show how each module integrates with the other modules, we provide a case study. In this study, the first four sliding windows are analyzed with emphasis on the incremental association rule generation.

The stream alphabet used in the case study is $\mathcal{A} = \{a, b, c, d, e\}$. The sliding window size is $\tau = 10$. The application specified thresholds are $s_{min} = 3$ and $c_{min} = 0.7$. Figure 5.9 depicts PAET at the end of the first sliding window (τ_1). Table 5.4 provides a step by step explanation of PAET traversal over the sliding window from τ_1 to τ_2 as shown in Figure 5.5. The traversal stack is considered to already hold the USP items in a reverse lexicographical order. In every step, the top entry is popped out of the stack and examined further for the proper update actions. The columns *Prev. Ptr.* and *Curr. Ptr.* present the update for the $[n - 1]$ MCR pointers for corresponding PAET

Algorithm 2 $[n-n]$ association rule mining from $[n-1]$ association rules

```
Require: N21Map($\langle id, rule \rangle$)                    {//All [n-1] association rules}
 1: Constant: $WINDOW\_SZ$, $MIN\_CONF$, $MIN\_LIFT$;
 2: Initially: $RuleList\langle Map_{ord}(\langle id, rule \rangle)\rangle \leftarrow$ sort($N21Map$), $N2NMap(\langle id, rule \rangle) \leftarrow \emptyset$;
 3: Function N2NRuleGen($RuleList$) {          {//Modified Apriori-based rule generation}
 4: $newRuleList\langle Map(\langle id, rule \rangle)\rangle \leftarrow \emptyset$ ,
 5: if sizeof($RuleList$) ¿1 then
 6:     for level=1 to sizeof($RuleList$) do
 7:         $entry_{map}(\langle id, rule \rangle) \leftarrow$ elementof($RuleList, level$)
 8:         if $\neg$ empty($entry_{map}(\langle id, rule \rangle)$) then
 9:             $visitList\langle id \rangle \leftarrow \{\}$
10:             for all $antecedent \in$ parentsof(antecedentof($rule$)) do
11:                 $proposedRule \leftarrow \langle antecedent$, consequentof($rule$)$\rangle$
12:                 if $\neg$ contains($visitList, proposedRule$) then
13:                     Add getId($proposedRule$) to $visitList$
14:                     $bool \leftarrow$verify($proposedRule$)     {//Use Properties 4a., 5a., 6b. and 1 to check rule}
15:                     if $bool$ then
16:                         Add to $N2NMap$ using add($proposedRule, bool$) {//new [n-n] assoc. rule found}
17:                         $lvl \leftarrow$ getLevel($proposedRule$)
18:                         Add $proposedRule$ at level $lvl$ of $newRuleList\langle Map_{ord}(\langle id, rule \rangle)\rangle$
19:                     end if
20:                 end if
21:             end for
22:         end if
23:     end for
24: end if
25: sort ($newRuleList$)
26: N2NRuleGen($newRuleList$)                    {//Perform a tail-recursion}
27: }
```

nodes. The star sign (*) denotes that a pointer evaluation is required as discussed in Section 5.3.2. In the next column, *Rule Update Tasks*, evaluation of association rules is indicated. These evaluations include creating new rules or deleting or updating the existing rules. Using the $B1$ boundary, the rule evaluation task in Steps 13 and 15 is skipped as item e is omnipresent. Similarly, using the $B2$ boundary, all the lift evaluations are avoided except for *lift*$(ce \xrightarrow{l} b)$ as $sup_1(c)$ is less than ($\tau \times c_{min}$ = 7). Finally, the column *Push (Stack)* presents the itemsets that will be pushed into the stack for further traversal. In the first PAET traversal, no new nodes are created nor any existing node is removed from the tree.

Table 5.5 provides analytical details of PAET structure over the first four sliding windows in

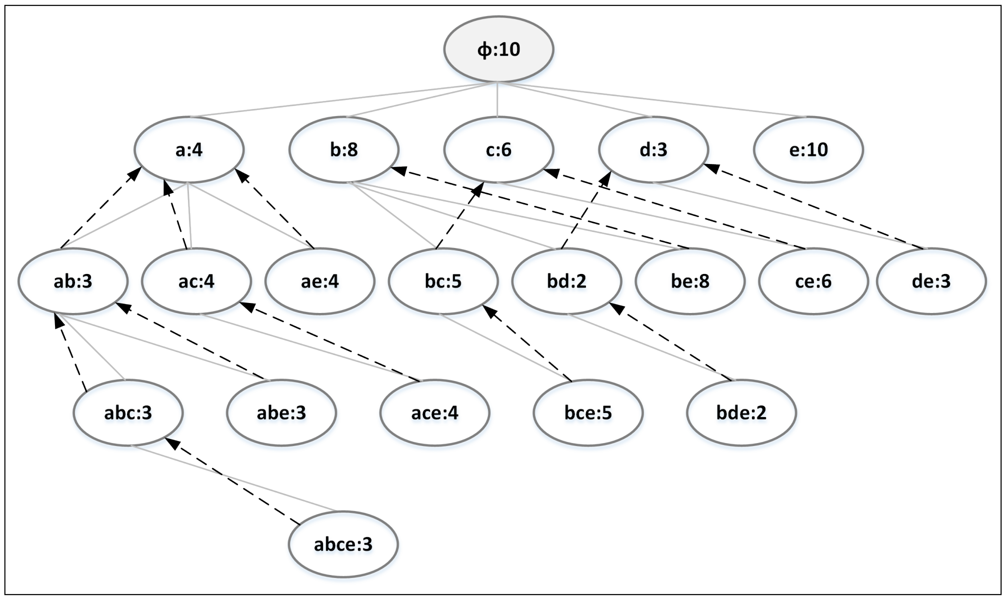


Figure 5.9: PAET generated from 1st sliding window (τ_1) of Figure 2.1

Table 5.4: $[n-1]$ Association rule generation for USP $\{(a,N),(b,N),(c,+),(d,-),(e,0)\}$

Step	Pop ($Stack$)	Prev. Ptr.	Curr. Ptr.	Rule Update Tasks	Push ($Stack$)
1	$(a,\emptyset,S_7)$	$\emptyset$	$\emptyset$	-	$(ae,a,S_7),(ac,a,S_6),(ab,a,S_9)$
2	(ab,a,S_9)	n_a	n_a	eval. lift($ab \xrightarrow{l} c$)	-
3	(ac,a,S_6)	n_a	n_a	eval. lift($a \xrightarrow{l} c$)	(ace,ac,S_6)
4	(ace,ac,S_6)	n_{ac}	n_{ac}	eval. lift($ae \xrightarrow{l} c$)	-
5	(ae,a,S_7)	n_a	n_a	-	-
6	$(b,\emptyset,S_7)$	$\emptyset$	$\emptyset$	-	$(be,b,S_7),(bd,b,S_{-6}),(bc,b,S_6)$
7	(bc,b,S_6)	n_c	n_c*	eval. lift($b \xrightarrow{l} c$)	(bce,bc,S_6)
8	(bce,bc,S_6)	n_{bc}	n_{bc}	eval. lift($be \xrightarrow{l} c$), eval. lift($ce \xrightarrow{l} b$)	-
9	(bd,b,S_{-6})	n_d	n_d	-	(bde,bd,S_{-6})
10	(bde,bd,S_{-6})	n_{bd}	n_{bd}	-	-
11	(be,b,S_7)	n_b	n_b	-	-
12	$(c,\emptyset,S_1)$	$\emptyset$	$\emptyset$	-	(ce,c,S_3)
13	(ce,c,S_3)	n_c	n_c*	- (e Omnipresent)	-
14	$(d,\emptyset,S_{-1})$	$\emptyset$	$\emptyset$	-	(de,d,S_{-3})
15	(de,d,S_{-3})	n_d	n_d	- (e Omnipresent)	-
16	$(e,\emptyset,S_0)$	$\emptyset$	$\emptyset$	-	-

Figure 5.5. It presents the total number of PAET nodes, the number of $[n-n]$ strong association rules and the number of $[n-n]$ interesting association rules. The analysis is done over different minimum support and minimum confidence threshold values.

Figure 5.10 shows a comparison between MAREDS proposed data structure, PAET, and other

Table 5.5: PAET analysis for various minimum support and minimum confidence values

Supp.	Conf.	τ_1			τ_2			τ_3			τ_4		
		Nodes	Rules	Lifted	Nodes	Rules	Lifted	Nodes	Rules	Lifted	Nodes	Rules	Lifted
	60%	19	37	21	19	33	12	19	33	19	15	33	19
30%	70%	19	27	12	19	26	9	19	23	13	15	23	13
	80%	13	18	9	19	14	6	19	14	6	15	14	6
	60%	17	19	12	15	16	3	15	16	7	15	16	7
40%	70%	17	12	6	15	12	3	15	15	7	15	15	7
	80%	17	12	6	15	9	3	15	9	3	15	9	3
	60%	11	11	6	11	11	0	11	11	4	11	11	4
50%	70%	11	7	3	11	7	0	11	10	4	11	10	4
	80%	11	7	3	11	4	0	11	4	0	11	4	0

similar data structures from existing research solutions. The *Closed Enumeration Tree* (CET) from Chi et al. [12] and *Frequent Pattern Tree* (FP-tree) from Han et al. [24] are constructed for the 1^{st} sliding window τ_1. The FP-tree is constructed with only 9 nodes whereas PAET has 19 nodes and CET has 20 nodes. Even though FP-tree hosts a small number of nodes, the nodes' structure in the tree is dependent on the itemsets' support within the sliding window. Thus, when the window is updated, the whole tree structure may change. Hence, it would not be possible to incrementally generate or track association rules. As for PAET, it contains comparatively less number of nodes than CET since CET starts accumulating infrequent gateway nodes as the sliding window progresses.

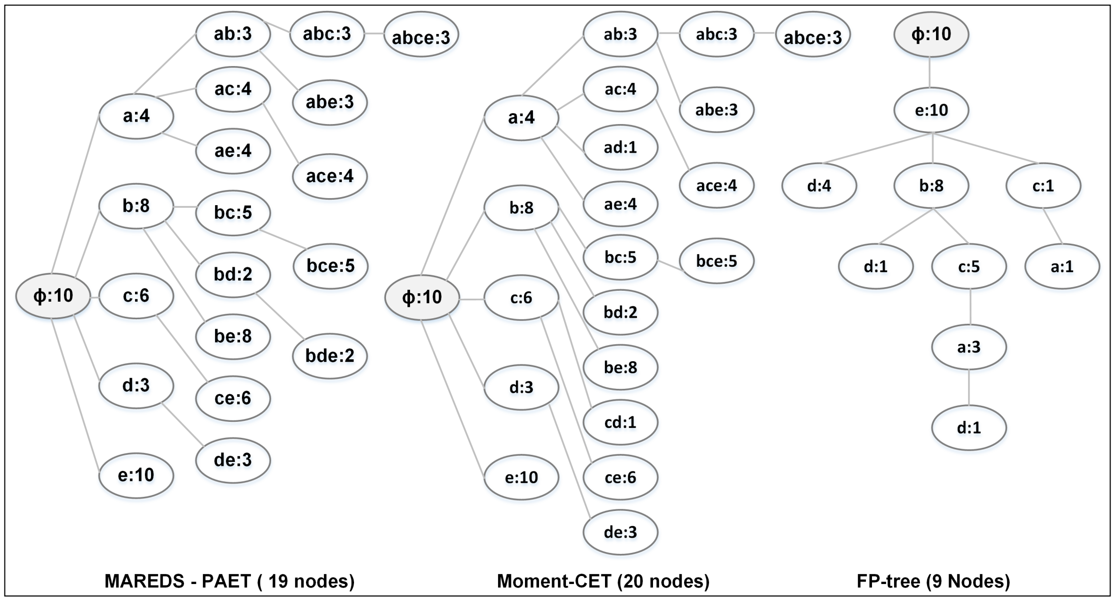


Figure 5.10: Comparison of three tree structures: PAET, CET (Moment) and FP-Tree

Chapter 6

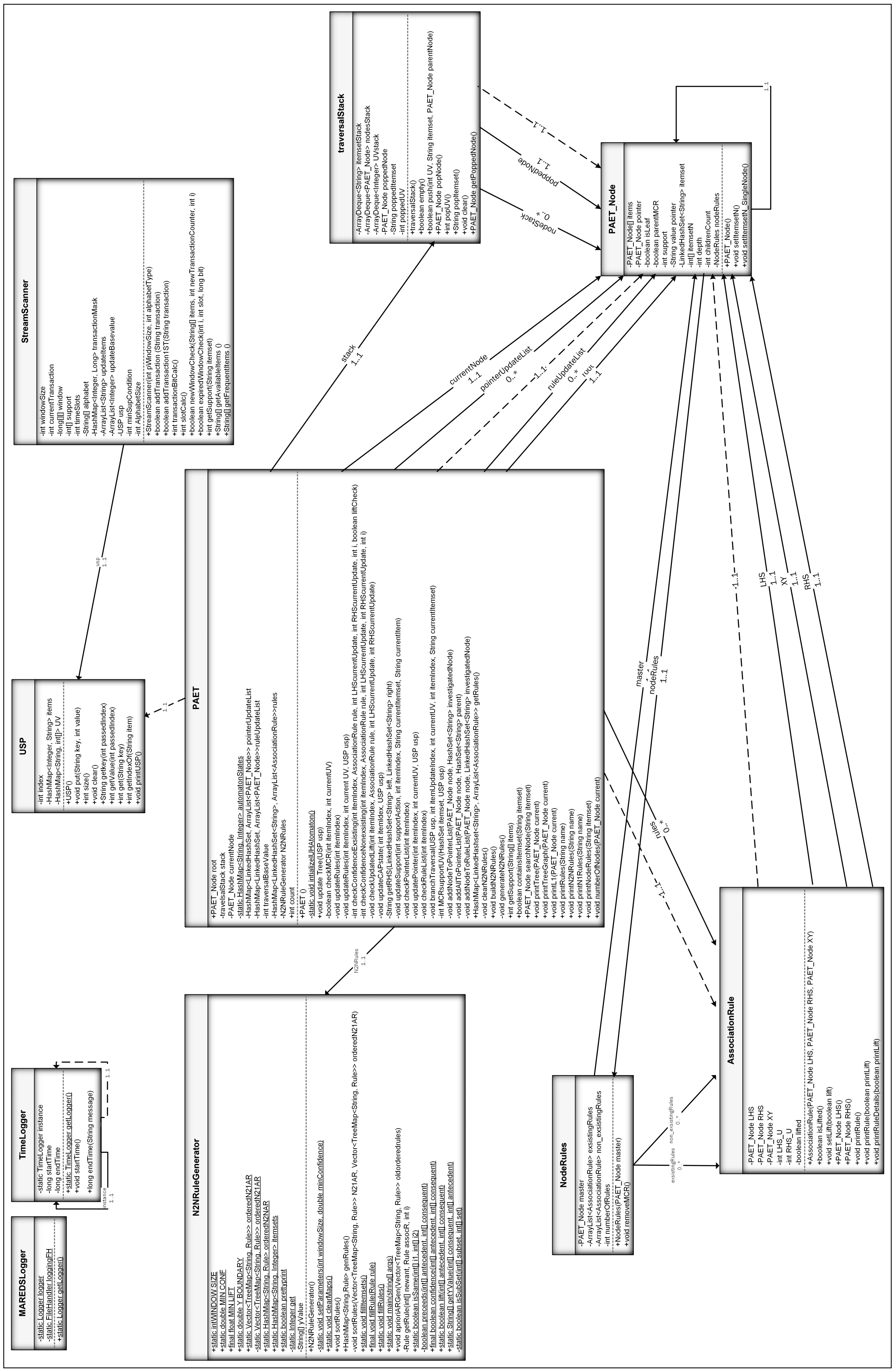
Results and Analysis

The objective of the experiments presented in this chapter is to evaluate the performance and scalability of our proposed stream association rule mining algorithm, MAREDS. The performance is evaluated by the average runtime of the tests and number of nodes required to maintain the association rules. The scalability is evaluated by the average runtime over different large size sliding windows. The performed tests aim to explore small to medium number of association rules which can be transformed into meaningful information in different applications. All experiments are conducted using a 3.40 GHz Intel Core i7-2600 PC with 8 GB main memory and running a 64-bit Windows 7 operating system.

6.1 Implementation and Design

MAREDS is implemented using Java programming language. Figure 6.1 shows the *UML* class diagram. There are three main classes that cover the major functionalities in the proposed algorithm: *StreamScanner*, *PAET* and *N2NRuleGenerator*. The other stated classes: *USP, PAET_Node, traversalStack, AssociationRule, MAREDSLogger* and *TimeLogger* have supporting functionalities only. The main classes functionalities are described below:

- *StreamScanner*: This class receives the data steam transactions and stores them in the sliding window. It keeps the sliding window updated as stated in Section 5.2. StreamScanner uses the USP class to pass updates to PAET.



MAREDSLogger
-static Logger logger
-static FileHandler loggingFH
+static Logger getLogger()
TimeLogger
-static TimeLogger instance
-long startTime
-long endTime
+static TimeLogger getLogger()
+void startTime()
+long endTime(String message)
instance
1..1
N2NRuleGenerator
USP
-int index
-HashMap<Integer, String> items
-HashMap<String, int[]> UV
+USP()
+void put(String key, int value)
+int size()
+void clear()
+String getkey(int passedIndex)
+int getValue(int passedIndex)
+int get(String key)
+int getIndexOf(String item)
+void printUSP()
usp
1..1
StreamScanner
-int windowSize
-int currentTransaction
-long[][] window
-int[] support
-int timeSlots
-String[] alphabet
+HashMap<Integer, Long> transactionMask
-ArrayList<String> updateItems
-ArrayList<Integer> updateBasevalue
-USP usp
-int minSupCondition
-int AlphabetSize
+StreamScanner(int pWindowSize, int alphabetType)
+boolean addTransaction (String transaction)
+boolean addTransaction1ST(String transaction)
+int transactionBitCalc()
+int slotCalc()
+boolean newWindowCheck(String[] items, int newTransactionCounter, int i)
+boolean expiredWindowCheck(int i, int slot, long bit)
+int getSupport(String itemset)
+String[] getAvailableItems()
+String[] getFrequentItems()
PAET
N2NRules
1..1
stack
1..1
currentNode
1..1
pointerUpdateList
0..*
1..1
ruleUpdateList
0..*
1..1
traversalStack
-ArrayDeque<String> itemsetStack
-ArrayDeque<PAET_Node> nodesStack
-ArrayDeque<Integer> UVstack
-PAET_Node poppedNode
-String poppedItemset
-int poppedUV
+traversalStack()
+boolean empty()
+boolean push(int UV, String itemset, PAET_Node parentNode)
+PAET_Node popNode()
+int popUV()
+String popItemset()
+void clear()
+PAET_Node getPoppedNode()
1..1
poppedNode
1..1
nodeStack
0..*
PAET_Node
-PAET_Node[] items
-PAET_Node pointer
-boolean isLeaf
-boolean parentMCR
-int support
-String value pointer
-LinkedHashSet<String> itemset
-int[] itemsetN
-int depth
-int childrenCount
-NodeRules nodeRules
+PAET_Node()
+void setItemsetN()
+void setItemsetN_SingleNode()
1..1
NodeRules
-PAET_Node master
-ArrayList<AssociationRule> existingRules
-ArrayList<AssociationRule> non_existingRules
-int numberOfRules
+NodeRules(PAET_Node master)
+void removeMCR()
master
nodeRules
1..1
existingRules
0..*
non_existingRules
0..*
rules
0..*
1..1
LHS
1..1
XY
1..1
RHS
1..1
AssociationRule
-PAET_Node LHS
-PAET_Node RHS
-PAET_Node XY
-int LHS_U
-int RHS_U
-boolean lifted
+AssociationRule(PAET_Node LHS, PAET_Node RHS, PAET_Node XY)
+boolean isLifted()
+void setLift(boolean lift)
+PAET_Node LHS()
+PAET_Node RHS()
+void printRule()
+void printRule(boolean printLift)
+void printRuleDetails(boolean printLift)

Figure 6.1: MAREDS algorithm class diagram

- *PAET*: This class represents the partial association enumeration tree stated in Section 5.3. It maintains the selected itemsets and relations among them. It uses the traversalStack class to traverse and update PAET. In addition, the class tracks all $[n-1]$ association rules as per Section 5.4.1 and stores them using the AssociationRule class.
- *N2NRuleGenerator*: This class is responsible for generating all $[n-n]$ rules as described in Section 5.4.2.

Since there is no suitable algorithm to incrementally mine association rules over a sliding window data stream, we choose to compare our proposed algorithm against two closely related existing algorithms. First, we consider *Moment*, an incremental stream frequent itemset mining algorithm [12]. Moment uses CET, a prefix tree structure, to maintain the mining outcome. Second, we consider *FP-Growth*, a non-incremental frequent itemset mining algorithm [24]. FP-Growth mines frequent itemsets from scratch every time the sliding window is updated. It uses FP-tree, a tree structure consisting of a set of item-prefix subtrees [24]. Apriori association rule learning [4] is used afterwards on both algorithm outcomes to generate desired association rules. To have a fare base of comparison, Moment, FP-Growth and Apriori implementations are all acquired in Java programming language.

6.2 Datasets

MAREDS is tested extensively over seven datasets, four of which are real-life datasets and the other three are synthetic. The datasets are carefully chosen in similarity to previous research efforts. Table 6.1 presents the characteristics of all the datasets used in our experiments. The first four datasets, namely BMS-WebView-1, BMS-WebView-2, Kosarak and Accident were generated by capturing actual events from a real-life environment. BMS-WebView-1 and BMS-WebView-2 represent two click streams of size 59,601 and 77,512 transactions, respectively. These two real-world datasets were used for KDDCUP 2000 [80]. Kosarak[1] is a large dataset that consists of 990,000 anonymized click stream transactions from a large on-line news portal. The Accidents dataset, published by Geurts et al. [19], contains information of traffic accidents from 1991 to 2000

[1] `http://fimi.ua.ac.be/data/`

in the region of Flanders (Belgium) as obtained from the National Institute of Statistics, Belgium. This dataset consists of highly correlated itemsets. The three synthetic datasets namely T5I4D100K, T10I4D100K and T20I5D100K were generated using the IBM Quest Synthetic Data Generator [2]. The symbols T, I and D in the datasets' naming denote the average number of items per transaction, the average size of itemsets in potential frequent sequences and the number of transactions in the dataset, respectively.

6.3 Experimental Results

In this section, we evaluate the performance of our stream association rule mining algorithm in terms of efficiency and scalability. The data stream environment was simulated in all experiments using the above mentions datasets. In all experimental results, for all the algorithms, we report the average running time over 100 consecutive updates of the sliding window. Figure 6.2 evaluates the performance of MAREDS approach over a sliding window of size 50,000 (50K). All the datasets are tested over the same range of minimum confidence values. However, each dataset uses a different minimum support value that would generate a reasonable number of association rules. The used support values are marked in the brackets next to each dataset within the legend. Each data point in this chart represents the minimum and maximum number of rules as found during the sliding window updates (presented in the square brackets). In all these experiments, MAREDS finds association rules in less than 10 milliseconds. It should be noted that, while the minimum confidence values decrease, the number of association rules increase, yet the performance of MAREDS

[2]`https://sourceforge.net/projects/ibmquestdatagen/`

Table 6.1: Experimental datasets characteristics

Dataset	Data Type	Number of items	Transactions			Window Size
			Count	Avg. length	Max. length	
BMS-WebView-1	Real	497	59602	2.51	267	2K, 50K
BMS-WebView-2	Real	3340	77512	4.62	161	2K, 50K
Kosarak	Real	41270	990002	8.10	2498	5K - 120K
Accidents	Real	468	340183	33.81	51	10K
T5I4D100K	Synthetic	500	100K	4.87	17	10K - 80K
T10I4D100K	Synthetic	500	100K	9.80	29	10K - 80K
T20I5D100K	Synthetic	500	100K	19.85	47	10K - 80K

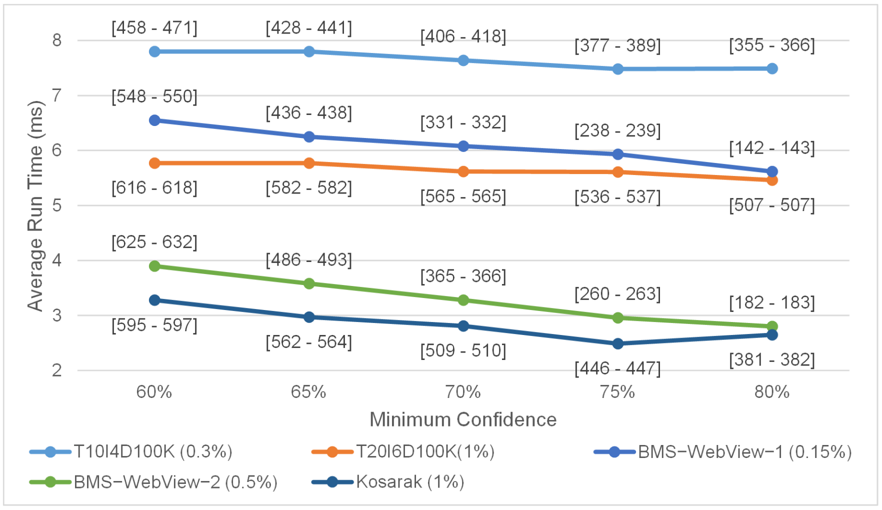


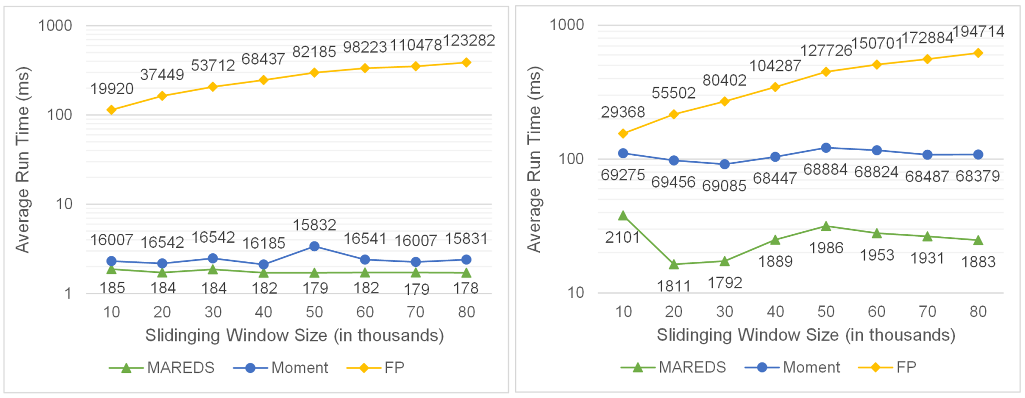
Figure 6.2: Association rules and performance evaluation for different datasets

remains stable for a fixed minimum support and a fixed sliding window size.

6.3.1 Experiments on Synthetic Datasets

Figures 6.3, 6.4 and 6.5 provide a performance comparison of the MAREDS, Moment and FP-Growth for the three synthetic datasets T5I4D100K, T10I4D100K and T20I5D100K, respectively. In these test cases, we examine the average run time while having a fixed minimum support and fixed minimum confidence values over a range of different sliding window sizes. For each dataset, we perform the test with two fixed minimum support values, one low and one high. The shown average run time is evaluated over a logarithmic scale. In addition, we compare the number of created nodes by the tree data structures used by each of the approaches. As the sliding window size increases, the absolute value of the minimum support linearly increases. However, we can notice that the affect on MAREDS's performance is minor whereas the run time of FP-Growth continuously increases. This stems from the fact that FP-Growth needs to reload all the sliding window transactions and build the FP-tree from scratch with each window update. In addition, every time the sliding window size becomes larger, FP-Growth needs more time to build the FP-Tree and generate the association rules. On the other hand, Moment and MAREDS algorithms incrementally update their tree data structures. Hence, their performance is relatively stable as the changes in the data stream are minimal after the first window is being loaded. Moreover, it can be observed that the

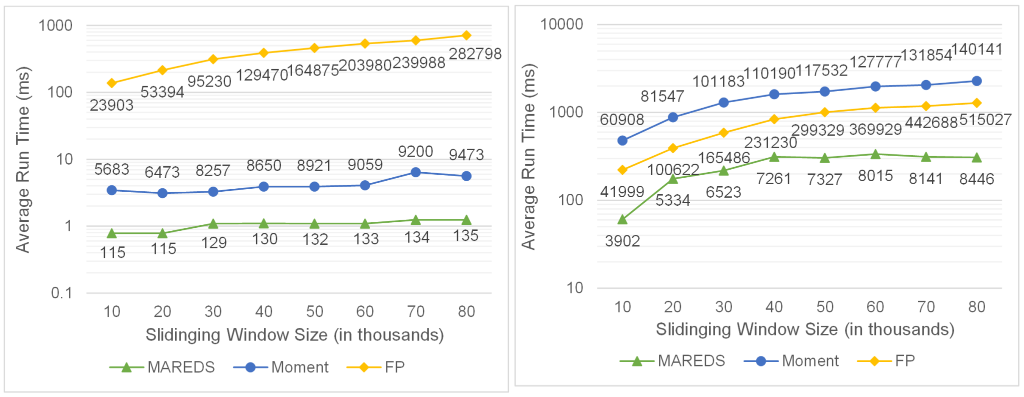
Moment algorithm performance degrades with lower minimum support values since it starts to have a large number of infrequent CET nodes. This impacts negatively on its performance and memory use. Finally, we can notice that among all three approaches, MAREDS stores the least number of tree nodes as marked on the labels in the figure.



(a) Support 1.0%

(b) Support 0.2%

Figure 6.3: Performance comparison for T5I4D100K dataset



(a) Support 2.5%

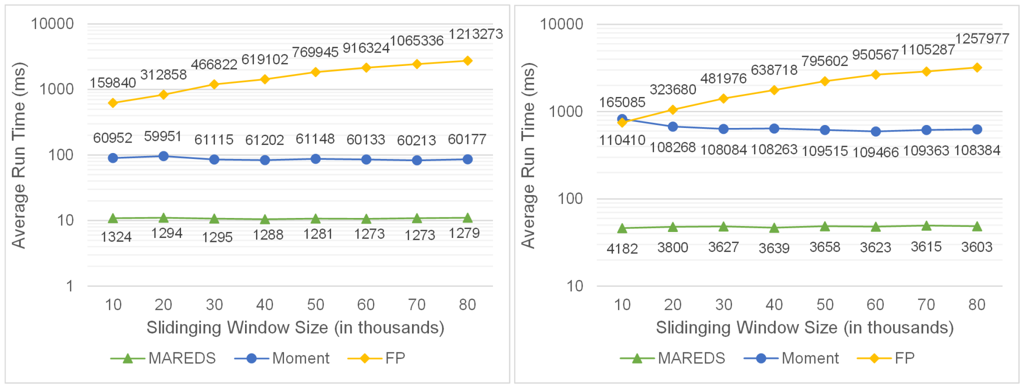
(b) Support 0.25%

Figure 6.4: Performance comparison for T10I4D100K dataset

6.3.2 Experiments on Real-Life Datasets

BMS-WebView-1 and BMS-WebView-2 Datasets

The experimental tests over BMS-WebView-1 and BMS-WebView-2 datasets are performed using two sliding window sizes, 2K and 50K and over a range of different minimum support values



(a) Support 1.6% (b) Support 1.1%

Figure 6.5: Performance comparison for T20I5D100K dataset

while having the minimum confidence fixed at 70%. The tests were chosen similar to previous research efforts by Moment [12]. Figures 6.6 and 6.7 provide a comparison among the three approaches in terms of run time and memory usage. The performance is evaluated over a logarithmic scale. Each data point label represents the number of created nodes in the corresponding trees. It is clear that MAREDS performs faster than the other two approaches in this set of test cases. In addition, it creates less number of PAET nodes in comparison with Moment CET nodes and FP-Growth FP-tree nodes. Hence, it handles memory efficiently. The column graph in the background of each sub-figure projects the average number of association rules over the same logarithmic scale. The average number of rules is calculated from 100 consecutive sliding window updates as well. We can see that the average number of association rules exceeds 20,000 at the end of each sub-figure. Therefore, for the purpose of having useful and beneficial information, we do not intend to stretch the experiments further on lower minimum support values.

Kosarak Dataset

Figure 6.8 compares all three approaches over the real dataset of click stream Kosarak. The number of distinct items in Kosarak is 41270 . The maximum transaction length is 2498 items. Both values are the highest among all used datasets. Such characteristics generate a large number of infrequent itemsets. This makes CET handling difficult for Moment algorithm even though the

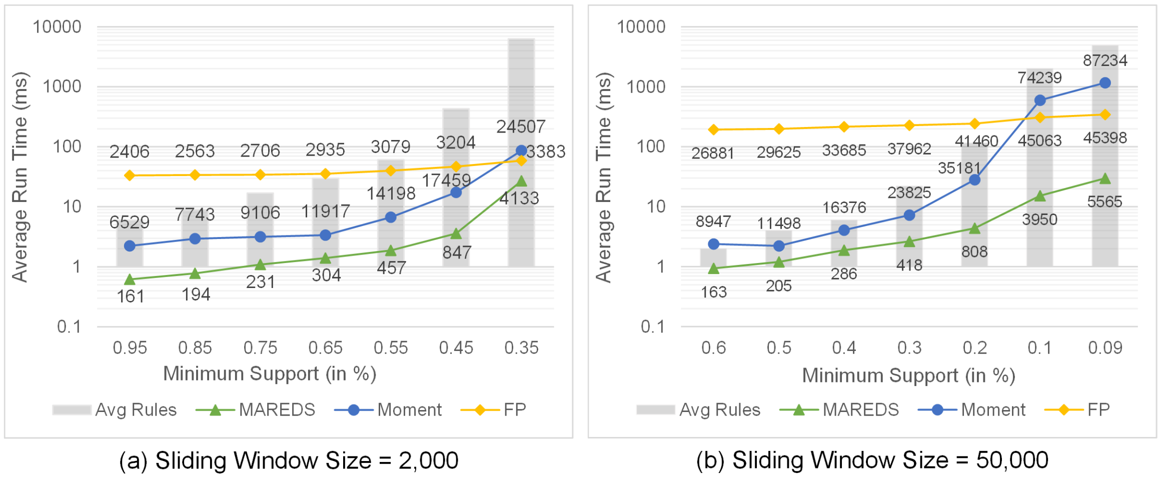


Figure 6.6: Run time and memory usage comparison for BMS-WebView-1 dataset

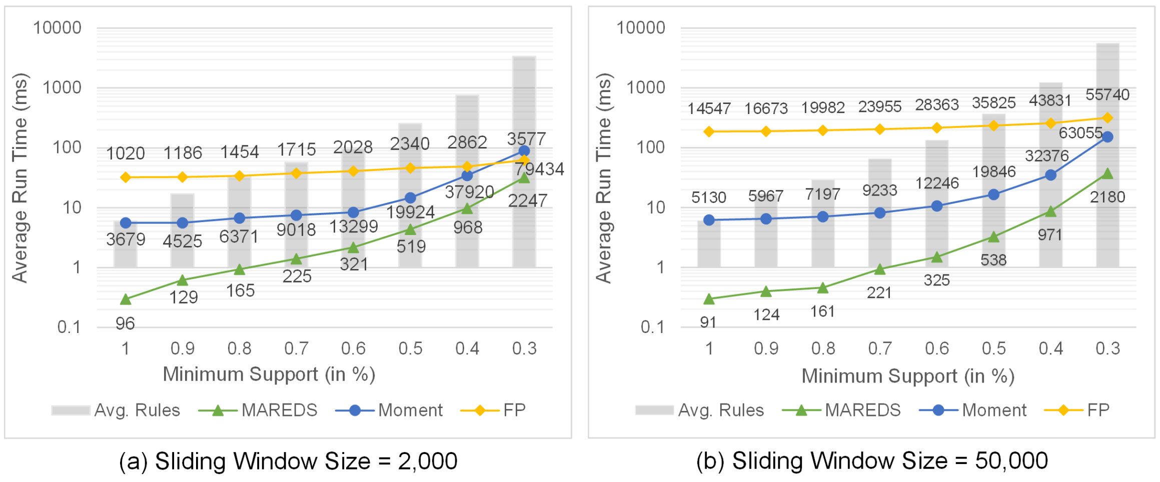
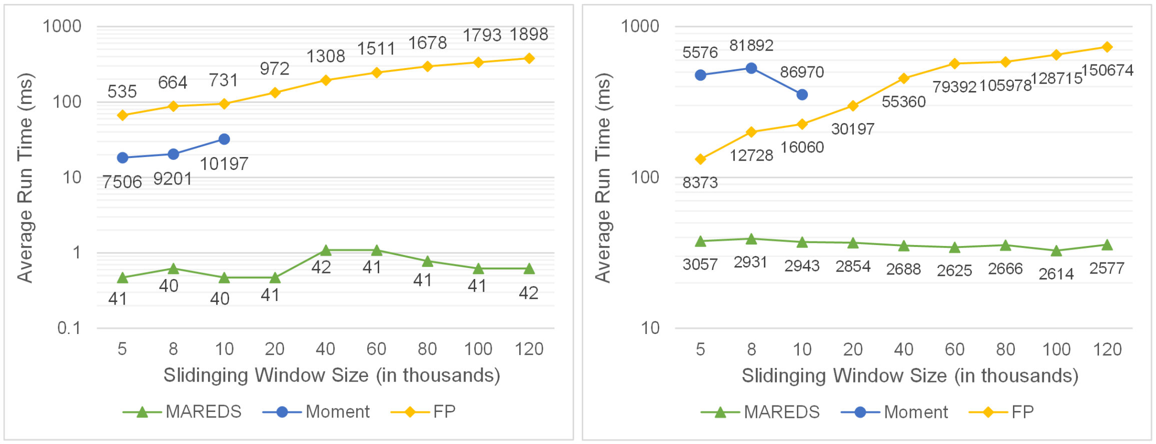


Figure 6.7: Run time and memory usage comparison for BMS-WebView-2 dataset

infrequent itemsets cannot be the part of any association rule. We can see in Figure 6.8 that Moment algorithm fails to produce results when the sliding window size grows larger than 10k. As for FP-Growth, association rule generation takes a longer time and creates far more tree nodes in comparison to MAREDS.



(a) Support 4.0% (b) Support 0.4%

Figure 6.8: Performance comparison over Kosarak dataset

Accidents Dataset

Figure 6.9 shows the performance evaluation for the Accidents dataset. In this dataset, the average transaction length is 33.81 with an alphabet size of 468 items. Having such characteristics generates frequent itemsets with high support values. Over a fixed sliding window of size 10k and a minimum confidence value of 70%, as the minimum support decreases from 90% to 65%, the number of generated association rules increase from 218 to 27020 rules. Similarly, the number of lifted association rules increase from 172 to 24144 rules. The column graphs in the background present the average number of strong association rules and interesting association rules on the secondary axis. Even with such a large number of association rules, MAREDS manages to generate rules faster than the other two approaches. It creates less number of tree nodes as well.

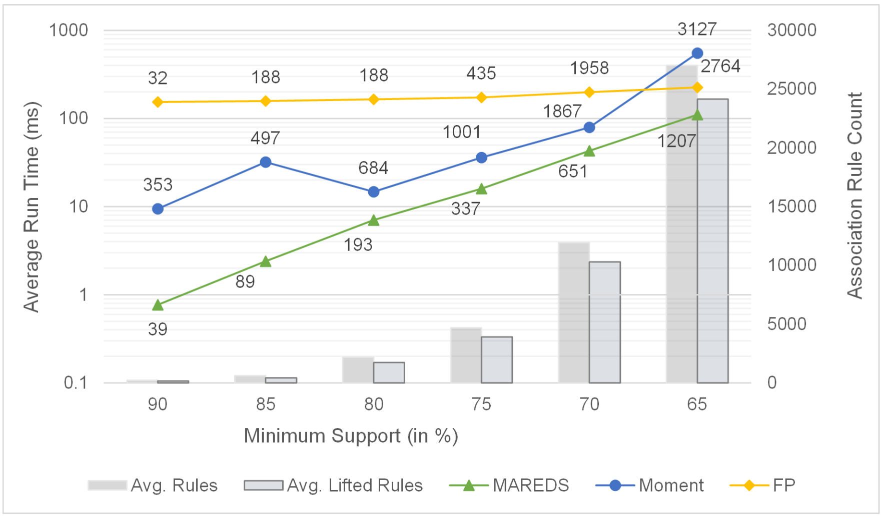


Figure 6.9: Finding lifted association rules from Accidents dataset

6.4 Summary

The aforementioned results and figures clearly demonstrate the efficiency, scalability and suitability of MAREDS algorithm. They manifest the benefits of incremental association rule mining over other existing approaches. The loading of the first sliding window for MAREDS is a bit time consuming in comparison to FP-Growth. However, once the first window is fully loaded, the node creation and deletion within PAET is found to be minimal as the sliding window progresses. Hence, MAREDS updates the support of tree nodes rapidly and maintains association rules efficiently.

Chapter 7

Conclusion and Future Work

In this thesis, we implemented an association rule mining algorithm that can incrementally generate association rules from data streams. Our proposed solution differs from other stream rule mining algorithms due to fact that association rules are always updated and available upon any user query. Other solutions require further calculations to generate the association rules when requested. Prompt information availability is of high value for critical and decisive applications.

In this chapter, we conclude the thesis by firstly providing a summary of the contribution and secondly describing the research directions that can be conducted as a future work.

7.1 Summary of Contributions

First, we conducted a survey over the major and most recent work on stream frequent itemset mining and association rule mining algorithms. The assessment of the algorithms showed that majority of the solutions focus on frequent itemset mining. Association rule generation was not discussed thoroughly in data streams. Moreover, algorithms which claim to generate association rules from data streams actually extracted all frequent itemsets and then applied traditional techniques to generate the rules.

Second, we proposed a novel algorithm, MAREDS, to incrementally mine association rules from evolving data streams over a sliding window model in a centralized setting. MAREDS used PAET, an in-memory efficient enumeration tree structure, to maintain frequent itemsets, potential itemsets and the relations among the frequent itemsets. We provided a generic, yet scalable, framework to maintain the itemsets in PAET and incrementally maintains $[n-1]$ interesting association rules with the least number of possible operations. The framework used the $[n-1]$ rules and a set of rule analysis properties to generate all valid $[n-n]$ interesting association rules.

Finally, we conducted an extensive experimental study over four real-life datasets and three synthetic datasets, where the effectiveness of the algorithm in terms of run time and memory efficiency was demonstrated. We also established that our approach is highly scalable over large sizes of sliding windows with different minimum support and minimum confidence threshold values.

7.2 Future Work

For future work, we identify the following potential research directions:

- The proposed algorithm in this thesis mines association rules from data streams in a centralized setting. This implies that different stream data has to be collected in a central location. In the future, we can consider a distributed solution where association rules are mined incrementally at different locations.

- In this thesis, we assume that all participating streams share their data without having any privacy concerns. In the future, a privacy-preserving model can be introduced where the stream owners do not have to expose all their data, yet the associations among the data can be generated incrementally.

- The generated association rules within a sliding window are stored as long as they meet the three rule generation constraints. If any of the constraints becomes invalid, the association rule is discarded. It would be interesting in the future to examine which association rules are always frequent and which association rules keep on changing statuses as the sliding window progresses.

Bibliography

[1] Charu C Aggarwal. *Managing and mining sensor data*. Springer Science & Business Media, 2013.

[2] Charu C Aggarwal and Philip S Yu. Online generation of association rules. In *Data Engineering, 1998. Proceedings., 14th International Conference on*, pages 402–411. IEEE, 1998.

[3] Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. Mining association rules between sets of items in large databases. In *Acm sigmod record*, volume 22, pages 207–216. ACM, 1993.

[4] Rakesh Agrawal, Ramakrishnan Srikant, et al. Fast algorithms for mining association rules. In *Proc. 20th int. conf. very large data bases, VLDB*, volume 1215, pages 487–499, 1994.

[5] Apache. Apache hadoop. [online], `http://hadoop.apache.org/`, September 2016.

[6] Christian Borgelt. Efficient implementations of apriori and eclat. In *FIMI'03: Proceedings of the IEEE ICDM workshop on frequent itemset mining implementations*, 2003.

[7] Dariusz Brzeziński. *Mining data streams with concept drift*. PhD thesis, Masters thesis, Poznan University of Technology, 2010.

[8] Eugenio Cesario, Carlo Mastroianni, and Domenico Talia. A multi-domain architecture for mining frequent items and itemsets from distributed data streams. *Journal of grid computing*, 12(1):153–168, 2014.

[9] Joong Hyuk Chang and Won Suk Lee. Finding recently frequent itemsets adaptively over online transactional data streams. *Information Systems*, 31(8):849–869, 2006.

[10] Peng Chen, Hongye Su, Lichao Guo, and Yu Qu. Mining fuzzy association rules in data streams. In *Computer Engineering and Technology (ICCET), 2010 2nd International Conference on*, volume 4, pages V4–153. IEEE, 2010.

[11] Herman Chernoff. A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *The Annals of Mathematical Statistics*, pages 493–507, 1952.

[12] Yun Chi, Haixun Wang, S Yu Philip, and Richard R Muntz. Catch the moment: maintaining closed frequent itemsets over a data stream sliding window. *Knowledge and Information Systems*, 10(3):265–294, October 2006.

[13] Frans Coenen. Data mining: past, present and future. *The Knowledge Engineering Review*, 26(01):25–29, 2011.

[14] Samet Çokpınar and Taflan İmre G undem. Positive and negative association rule mining on xml data streams in database as a service concept. *Expert Systems with Applications*, 39(8):7503–7511, 2012.

[15] Xuan Hong Dang, Vincent CS Lee, Wee Keong Ng, and Kok Leong Ong. Incremental and adaptive clustering stream data over sliding window. In *Database and Expert Systems Applications*, pages 660–674. Springer, 2009.

[16] Daniel Price. Surprising facts and stats about the big data industry. [online], `http://cloudtweaks.com/2015/03/surprising-facts-and-stats-about-the-big-data-industry/`, March 2015.

[17] Mahmood Deypir and Mohammad Hadi Sadreddini. A dynamic layout of sliding window for frequent itemset mining over data streams. *Journal of Systems and Software*, 85(3):746–759, 2012.

[18] Steve Donoho. Early detection of insider trading in option markets. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 420–429. ACM, 2004.

[19] Karolien Geurts, Geert Wets, Tom Brijs, and Koen Vanhoof. Profiling high frequency accident locations using association rules. In *Proceedings of the 82nd Annual Transportation Research Board*, page 18pp, Washington DC. (USA), Jan. 2003.

[20] Lukasz Golab and M Tamer Özsu. Issues in data stream management. *ACM Sigmod Record*, 32(2):5–14, 2003.

[21] Pritam Gundecha and Huan Liu. Mining social media: a brief introduction. *Tutorials in Operations Research*, 1(4):1–17, 2012.

[22] Jiawei Han, Jian Pei, and Micheline Kamber. *Data mining: concepts and techniques*. Elsevier, 2011.

[23] Jiawei Han, Jian Pei, and Yiwen Yin. Mining frequent patterns without candidate generation. In *ACM Sigmod Record*, volume 29, pages 1–12. ACM, 2000.

[24] Jiawei Han, Jian Pei, Yiwen Yin, and Runying Mao. Mining frequent patterns without candidate generation: A frequent-pattern tree approach. *Data mining and knowledge discovery*, 8(1):53–87, 2004.

[25] Jing He. Advances in data mining: History and future. In *Intelligent Information Technology Application, 2009. IITA 2009. Third International Symposium on*, volume 1, pages 634–636. IEEE, 2009.

[26] Jen-Wei Huang, Su-Chen Lin, and Ming-Syan Chen. Dpsp: distributed progressive sequential pattern mining on the cloud. In *Advances in Knowledge Discovery and Data Mining*, pages 27–34. Springer, 2010.

[27] Nan Jiang and Le Gruenwald. Cfi-stream: mining closed frequent itemsets in data streams. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 592–597. ACM, 2006.

[28] Nan Jiang and Le Gruenwald. Research issues in data stream association rule mining. *ACM Sigmod Record*, 35(1):14–19, 2006.

[29] Mehmed Kantardzic. *Data mining: concepts, models, methods, and algorithms*. John Wiley & Sons, 2011.

[30] Mahnoosh Kholghi and Mohammadreza Keyvanpour. An analytical framework for data stream mining techniques based on challenges and requirements. *arXiv preprint arXiv:1105.1950*, 2011.

[31] Sotiris Kotsiantis and Dimitris Kanellopoulos. Association rules mining: A recent overview. *GESTS International Transactions on Computer Science and Engineering*, 32(1):71–82, 2006.

[32] Sotiris Kotsiantis and Dimitris Kanellopoulos. Association rules mining: A recent overview. *GESTS International Transactions on Computer Science and Engineering*, 32(1):71–82, 2006.

[33] Peter J Landin. The next 700 programming languages. *Communications of the ACM*, 9(3):157–166, 1966.

[34] Pat Langley, Wayne Iba, and Kevin Thompson. An analysis of bayesian classifiers. In *Aaai*, volume 90, pages 223–228, 1992.

[35] Carson Kai-Sang Leung and Boyu Hao. Mining of frequent itemsets from streams of uncertain data. In *Data Engineering, 2009. ICDE'09. IEEE 25th International Conference on*, pages 1663–1670. IEEE, 2009.

[36] Carson Kai-Sang Leung and Fan Jiang. Frequent itemset mining of uncertain data streams using the damped window model. In *Proceedings of the 2011 ACM Symposium on Applied Computing*, pages 950–955. ACM, 2011.

[37] Carson Kai-Sang Leung and Quamrul I Khan. Dstree: a tree structure for the mining of frequent sets from data streams. In *Data Mining, 2006. ICDM'06. Sixth International Conference on*, pages 928–932. IEEE, 2006.

[38] Deren Li, Shuliang Wang, and Deyi Li. *Spatial Data Mining: Theory and Application*. Springer, 2016.

[39] Hua-Fu Li, Chin-Chuan Ho, and Suh-Yin Lee. Incremental updates of closed frequent itemsets over continuous data streams. *Expert Systems with Applications*, 36(2):2451–2458, 2009.

[40] Hua-Fu Li and Suh-Yin Lee. Mining frequent itemsets over data streams using efficient window sliding techniques. *Expert Systems with Applications*, 36(2):1466–1477, 2009.

[41] Hua-Fu Li, Suh-Yin Lee, and Man-Kwan Shan. An efficient algorithm for mining frequent itemsets over the entire history of data streams. In *Proc. of First International Workshop on Knowledge Discovery in Data Streams*, volume 39, 2004.

[42] Xuejun Liu, Jihong Guan, and Ping Hu. Mining frequent closed itemsets from a landmark window over online data streams. *Computers & Mathematics with Applications*, 57(6):927–936, 2009.

[43] Amit Manjhi, Vladislav Shkapenyuk, Kedar Dhamdhere, and Christopher Olston. Finding (recently) frequent items in distributed data streams. In *Data Engineering, 2005. ICDE 2005. Proceedings. 21st International Conference on*, pages 767–778. IEEE, 2005.

[44] Gurmeet Singh Manku and Rajeev Motwani. Approximate frequency counts over data streams. In *Proceedings of the 28th International Conference on Very Large Data Bases*, VLDB '02, pages 346–357. VLDB Endowment, 2002.

[45] Ahmed Metwally, Divyakant Agrawal, and Amr El Abbadi. An integrated efficient solution for computing frequent and top-k elements in data streams. *ACM Transactions on Database Systems (TODS)*, 31(3):1095–1133, 2006.

[46] João M Moreira, Carlos Soares, Alípio M Jorge, and Jorge Freire de Sousa. The effect of varying parameters and focusing on bus travel time prediction. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 689–696. Springer, 2009.

[47] Barzan Mozafari, Hetal Thakkar, and Carlo Zaniolo. Verifying and mining frequent patterns from large windows over data streams. In *2008 IEEE 24th International Conference on Data Engineering*, pages 179–188. IEEE, 2008.

[48] Shankar B Naik and Jyoti D Pawar. A quick algorithm for incremental mining closed frequent itemsets over data streams. In *Proceedings of the Second ACM IKDD Conference on Data Sciences*, pages 126–127. ACM, 2015.

[49] Fatemeh Nori, Mahmood Deypir, and Mohamad Hadi Sadreddini. A sliding window based algorithm for frequent closed itemset mining over data streams. *Journal of Systems and Software*, 86(3):615–623, 2013.

[50] Juryon Paik, Junghyun Nam, Ung Mo Kim, and Dongho Won. Association rule extraction from xml stream data for wireless sensor networks. *Sensors*, 14(7):12937–12957, 2014.

[51] Byung-Hoon Park and Hillol Kargupta. Distributed data mining: Algorithms, systems and applications. pages 341–358. Citeseer, 2002.

[52] Animesh Patcha and Jung-Min Park. An overview of anomaly detection techniques: Existing solutions and latest technological trends. *Computer networks*, 51(12):3448–3470, 2007.

[53] S Pramod and OP Vyas. Data stream mining: A review on windowing approach. *Global Journal of Computer Science and Technology Software & Data Engineering*, 12(11):26–30, 2012.

[54] S Pramod and OP Vyas. Data stream mining: A review. In *Proceedings of the Third International Conference on Trends in Information, Telecommunication and Computing*, pages 621–627. Springer, 2013.

[55] Anand Rajaraman and Jeffrey David Ullman. *Mining of Massive Datasets*. Cambridge University Press, New York, NY, USA, 2011.

[56] Vinaya Sawant and Ketan Shah. A survey of distributed association rule mining algorithms. *Journal of Emerging Trends in Computing and Information Sciences*, 5(5), 2014.

[57] Se Jung Shin and Won Suk Lee. On-line generation association rules over data streams. *Information and Software Technology*, 50(6):569–578, 2008.

[58] Paria Shirani, Mohammad Abdollahi Azgomi, and Saed Alrabaee. A method for intrusion detection in web services based on time series. In *Electrical and Computer Engineering (CCECE), 2015 IEEE 28th Canadian Conference on*, pages 836–841. IEEE, 2015.

[59] Mingjun Song and Sanguthevar Rajasekaran. A transaction mapping algorithm for frequent itemsets mining. *IEEE transactions on Knowledge and Data Engineering*, 18(4):472–481, 2006.

[60] Jimeng Sun, Spiros Papadimitriou, and Christos Faloutsos. Distributed pattern discovery in multiple streams. In *Advances in Knowledge Discovery and Data Mining*, pages 713–718. Springer, 2006.

[61] Pang-Ning Tan, Steinbach Michael, and Vipin Kumar. Chapter 6. association analysis: Basic concepts and algorithms. *Introduction to Data Mining. Addison-Wesley. ISBN*, 321321367, 2005.

[62] Syed Khairuzzaman Tanbeer, Chowdhury Farhan Ahmed, Byeong-Soo Jeong, and Young-Koo Lee. Sliding window-based frequent pattern mining over data streams. *Information sciences*, 179(22):3843–3865, 2009.

[63] Keming Tang, Caiyan Dai, and Ling Chen. A novel strategy for mining frequent closed itemsets in data streams. *Journal of Computers*, 7(7):1564–1573, 2012.

[64] Yingying Tao. Mining time-changing data streams. 2011.

[65] Hetal Thakkar, Barzan Mozafari, and Carlo Zaniolo. Continuous post-mining of association rules in a data stream management system. *Post-Mining of Association Rules: Techniques for Effective Knowledge Extraction*, pages 116–132, 2009.

[66] Manisha Thool and Preeti Voditel. Association rule generation in streams. *International Journal of Advanced Research in Computer and Communication Engineering*, 2(5), May 2013.

[67] Tzoumas, Ewen and Metzger. High-throughput, low-latency, and exactly-once stream processing with apache flink. [online], `http://data-artisans.com/`, August 2015.

[68] S Vijayarani and R Prasannalakshmi. Comparative analysis of association rule generation algorithms in data streams. *International Journal on Cybernetics & Informatics (IJCI)*, 4(1), Feb. 2015.

[69] En Tzu Wang and Arbee L. P. Chen. A novel hash-based approach for mining frequent itemsets over data streams requiring less memory space. *Data Mining and Knowledge Discovery*, 19(1):132–172, 2009.

[70] En Tzu Wang and Arbee LP Chen. Mining frequent itemsets over distributed data streams by continuously maintaining a global synopsis. *Data Mining and Knowledge Discovery*, 23(2):252–299, 2011.

[71] Ho Jin Woo and Won Suk Lee. estmax: Tracing maximal frequent item sets instantly over online transactional data streams. *Knowledge and Data Engineering, IEEE Transactions on*, 21(10):1418–1431, 2009.

[72] Gang Wu, Huxing Zhang, Meikang Qiu, Zhong Ming, Jiayin Li, and Xiao Qin. A decentralized approach for mining event correlations in distributed system monitoring. *Journal of parallel and Distributed Computing*, 73(3):330–340, 2013.

[73] Gang Wu, Huxing Zhang, Meikang Qiu, Zhong Ming, Jiayin Li, and Xiao Qin. A decentralized approach for mining event correlations in distributed system monitoring. *Journal of Parallel Distributed Computing*, 73(3):330–340, 2013.

[74] Show-Jane Yen, Cheng-Wei Wu, Yue-Shi Lee, Vincent S Tseng, and Chaur-Heh Hsieh. A fast algorithm for mining frequent closed itemsets over stream sliding window. In *Fuzzy Systems (FUZZ), 2011 IEEE International Conference on*, pages 996–1002. IEEE, 2011.

[75] Jeffrey Xu Yu, Zhihong Chong, Hongjun Lu, Zhenjie Zhang, and Aoying Zhou. A false negative approach to mining frequent itemsets from high speed transactional data streams. *Information Sciences*, 176(14):1986–2015, 2006.

[76] Mohammed J Zaki. Scalable algorithms for association mining. *Knowledge and Data Engineering, IEEE Transactions on*, 12(3):372–390, 2000.

[77] Mohammed J Zaki and Karam Gouda. Fast vertical mining using diffsets. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 326–335. ACM, 2003.

[78] Li Zeng, Ling Li, Lian Duan, Kevin Lu, Zhongzhi Shi, Maoguang Wang, Wenjuan Wu, and Ping Luo. Distributed data mining: A survey. *Information Technology and Management*, 13(4):403–409, 2012.

[79] Meng Zhang and Guojun Mao. An approach for distributed streams mining using combination of nave bayes and decision trees. In *Proceedings of The Third International Conference on Advances in Databases, Knowledge, and Data Applications*, pages 29–33, St. Maarten, The Netherlands Antilles, jan 2011. IARIA.

[80] Zijian Zheng, Ron Kohavi, and Llew Mason. Real world performance of association rule algorithms. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 401–406. ACM, 2001.

[81] Xie Zhi-Jun, Chen Hong, and Cuiping Li. An efficient algorithm for frequent itemset mining on data streams. In *Advances in Data Mining. Applications in Medicine, Web Mining, Marketing, Image and Signal Mining*, pages 474–491. Springer, 2006.