UNIVERSITY OF OKLAHOMA

GRADUATE COLLEGE

DISCOVERING INTERESTING PATTERNS AND ASSOCIATIONS IN DATA STREAMS

A DISSERTATION

SUBMITTED TO THE GRADUATE FACULTY

in partial fulfillment of the requirements for the

Degree of

DOCTOR OF PHILOSOPHY

By

NAN JIANG
Norman, Oklahoma
2009

DISCOVERING INTERESTING PATTERNS AND ASSOCIATIONS
IN DATA STREAMS

A DISSERTATION APPROVED FOR THE
SCHOOL OF COMPUTER SCIENCE

BY

_____
Dr. T.H. Lee Williams, Chair

_____
Dr. John K. Antonio

_____
Dr. Mohammed Atiquzzaman

_____
Dr. Yifei Dong

_____
Dr. Changwook Kim

_____
Dr. Albert B. Schwarzkopf

To my family

Acknowledgements

First, I would like to thank our University for providing a nourishing environment for study and research. I would like to express my sincere thanks to the officers and staffs in the Graduate College who provide a warm and lively environment to encourage and help graduate students to grow in their graduate study. My research enthusiasm was encouraged and enriched by different research seminars, travel and research grants and workshop presentations offered through Graduate College over the years. When there is any question during my graduate study, their help and advice have always been timely, specific and professional.

Next, I would like to thank all my committee members. Dr. T.H. Lee Williams, Dr. John K. Antonio, Dr. Mohammed Atiquzzaman, Dr. Yifei Dong, Dr. Changwook Kim, and Dr. Albert B. Schwarzkopf. I indebted to all of them for their valuable time spent in discussing with me on the dissertation over the past years while serving on my supervisory committee. Their expert advice and suggestions have been very helpful in improving the quality of this dissertation. I appreciate and remember all of the time we spend together in discussing the dissertation. Your patience, kindly help and advice, supports and encouragements are in my lifetime memory. My dissertation and graduate career would not have been completed without your help and support.

Dr. Yun Chi, Dr. Mohammed Zaki and Mihale Halatchev provided the comparing programs to perform the experimental study. Thank you for giving me

access to this data. This dissertation work has been done while I was working on different projects as a research or teaching assistant, which provide me financial supports on my graduate study and a nice study environment. I would like to thank Dr. John K. Antonio for providing me the opportunity to work as a teaching assistant for our department during the first couple of years in my graduate study; thank Dr. Deborah Trytten for providing me the opportunity to work as a research assistant in the OSIIS project in the Oklahoma Health Department; thank Dr. Le Gruenwald for providing me the opportunity to work as a research assistant in the SENSE project in the conjunction team with the Oklahoma State, Tulsa, and Langston University; and thank Dr. Robert Dauffenbach and John G. McCraw for providing me the opportunity to work as a research assistant in the Center for Economic and Management Research on multiple projects. Many thanks to my supervisors and colleagues while I was working in these projects, working with you is a great pleasure and the work experience developed from these projects is a fortunate in my life.

I would like to thank my family, for encouraging me to pursue graduate studies and for always being there to share and to help. Their love, patience and support to me during my graduate study never ceased, and always be there whenever I need help. I thank you for your love and support, and dedicate this work to you. Last but not least, thanks for God who leads me through the way and makes me to have the courage, patience and knowledge to complete this dissertation.

Contents

List Of Tables

List Of Figures

Abstract

A data stream is a sequence of items that arrive in a timely order. Different from data in traditional static databases, data streams are continuous, unbounded, usually come with high speed, and have a data value distribution that often changes with time (Guha, 2001). As more applications such as web transactions, telephone records, and network flows generate a large number of data streams every day, efficient knowledge discovery of data streams is an active and growing research area in data mining with broad applications. Traditional data mining algorithms are developed to work on a complete static dataset and, thus, cannot be applied directly in data stream applications.

One area of data mining research is to mine association relationship in a data set. Most of association mining techniques for data streams can be categorized into two types: those developed based on frequent patterns and those developed based on closed patterns. Due to the number of frequent patterns are often huge and redundant, non-informative patterns are contained in frequent patterns. An alternative way is to develop the association mining approaches for data streaming applications based on closed patterns, which generally represent a small subset of all frequent patterns, but provide complete and condensed information. In these researches, the closed pattern mining is the prerequisite condition for non-redundant and informative association mining.

In this dissertation, a sliding window technique for dynamic mining of closed patterns in data streams is proposed, and an approach of mining non-redundant and informative associations based on the discovered closed patterns is developed. The closed pattern and relevant association mining techniques are selected research area in this dissertation. First, the closed patterns for a given collection of data are currently the most compact data knowledge that can provide complete support information for all data patterns. Compared with other techniques, the proposed closed pattern mining technique has potential to largely decrease the number of subsequent combinatorial calculations performed on the data patterns. Second, the memory requirement to store the closed patterns and relevant associations is generally lower than the corresponding frequent patterns and associations. In some data streaming applications, memory usage is an important measurement, because in these applications memory usage is the bottleneck for knowledge discovery. Third, the associations generated for data streams are the knowledge used to identify the relations within the data. The discovered relations can find their wide applications in many data streaming environments.

Different from the closed pattern mining techniques on traditional databases, which require multiple scans of the entire database, the proposed technique determines the closed patterns with a single scan. It is an incremental mining process; as the sliding window advances, new data transactions enter and old data transactions exit the window. But instead of regenerating closed patterns from the entire window, the proposed technique updates the old set of closed patterns

whenever a new transaction arrives and/or an old transaction leaves the sliding window to obtain the current set of closed patterns. This incremental feature allows the user to get the most recent updated closed patterns without rescanning the entire updated database, which saves not only the computation time, but more importantly, the I/O operating time to load and write data from database to memory. Third, the proposed sliding window technique can handle both the insertion and deletion operations independently, which allows the user to adjust the sliding window size in different application environments. Furthermore, the proposed interesting patterns and association mining framework can handle different users' requests at the same time at their specified support and confidence thresholds, and interested input and output patterns.

The research includes both theoretical proofs of correctness for the proposed algorithms and simulation experiments to compare the proposed techniques with those existing in the literature using synthetic and real datasets. The utility of the proposed technique is applied to sensor network databases of a traffic management and an environmental monitoring site for missing data estimation purpose.

1 Introduction

1.1 Problem Definition

1.1.1 Data Mining and Knowledge Discovery

The term ‘data mining’ refers to a process of nontrivial extraction of implicit, previously unknown, and potentially useful information (such as knowledge rules, regularities and outliers) from data in databases (Tan, 2005). The term ‘knowledge discovery’ is more general than the term ‘data mining’. Data mining is usually viewed as a step in the process of knowledge discovery (Han, 2001). The entire life cycle of knowledge discovery includes steps such as data cleaning, data integration, data selection, data transformation, data mining, pattern evaluation, and knowledge presentation.

Briefly stated, Knowledge Discovery in Database (KDD) is the rapidly growing inter-disciplinary field that merges together database management, statistics, and machine leaning and aims to extract useful and understandable knowledge from large volumes of data. Data mining is a critical step of the KDD process that performs the extraction of unknown knowledge in data. Data mining can be performed on a variety of data stores, including relational databases, transactional databases, data warehouses, and data streams. A comprehensive data mining system usually provides multiple mining functions. Association mining is one of the key features that can be found in such systems.

1.1.2 Data Streaming Application

A data stream is a sequence of items that arrive in a timed order. Different from data in traditional static databases, data streams are continuous, unbounded, usually arrive with high speed, and have a data value distribution that often changes with time (Guha, 2001). A data stream is represented mathematically as an ordered pair (r, Δ) where: r is a sequence of tuples, Δ is the sequence of time intervals (i.e. rational or real numbers) and each $\Delta_i > 0$.

Applications that reply on data streams can be classified into offline and online streaming. Offline streaming applications are characterized by regular bulk arrivals (Manku, 2002). Generating reports based on accumulated web log streams is an example of mining offline data streams because most of reports are made based on log data that is collected over a relatively large period of time. Online streaming applications are characterized by real-time updated data that needs to be quickly processed as the data is arrived. Predicting frequency of Internet packet streams is an application of mining online data streams because the prediction needs to be made in real time. Other potential online data streaming applications include stock tickers, network measurements, and evaluation of sensor data. In online data streaming applications, data is often discarded soon after it arrives and has been processed, because of the high update rate and huge resulting amount of data. Therefore, unlike offline data streaming applications, bulk processing a large portion of received data is not appropriate for online data streaming applications.

1.1.3 Association Rule and Association Mining

An association rule is an implication of the form $X \Rightarrow Y$ (s, c), where X and Y are frequent sets of items (also called itemsets) in a database, and $X \cap Y = \phi$. The parameter s, support of the rule, represents the percentage of records that contain both X and Y in the database. The parameter c, confidence of the rule, is the percentage of records containing X that also contain Y. An association rule is said to hold if both s and c are above or equal to a user-specified minimum support and confidence (Agrawal, 1993).

Association mining, also called association rule mining, searches for interesting relationships among items in a given database and displays them in rule form, for example $X \Rightarrow Y$. In practice, association mining involves finding association rules, the support and confidence of which are above or equal to a user-specified minimum support and confidence, respectively (Agrawal, 1993).

With the massive amounts of data continuously being collected and stored in databases, many industries are becoming interested in mining associations. Below is a typical market basket analysis example of association mining.

Example 1.1 Suppose, as a manager of a supermarket, you would like to learn more about the buying habits of your customers. Specifically, you may wonder "Which groups or sets of items are customers likely to purchase on a given trip to the supermarket?" To answer your question, association mining can be performed on the retail data of customer transactions at your supermarket. The knowledge that

customers who purchase bread also tend to buy milk at the same time is represented in the association rule below.

bread $\Rightarrow$ milk (s = 2%, c = 60%)

Support and confidence are two measures of rule interestingness. In the above association rule, the support of 2% means that 2% of all the transactions under analysis show that bread and milk are purchased together. The confidence of 60% means that 60% of the customers who purchase bread also buy milk. For this example, it should be noticed that the association rule: milk $\Rightarrow$ bread, has the same support, but not necessarily the same confidence as the association rule: bread $\Rightarrow$ milk. In short, support represents the percentage of data samples that the given rule satisfies and confidence assesses the degree of certainty of the detected association. Support and confidence thresholds are usually set by users or domain experts.

Association rule mining is typically considered to be a two-step process (Agrawal, 1993).

Step 1: Find all *frequent patterns*. By definition, each of these patterns will occur at least as frequently as a user-specified minimum support count.

Step 2: Generate strong association rules above user-specified support and confidence thresholds from the *frequent patterns*.

Frequent pattern mining (Step 1) is a crucial step of the process, and its computational efficiency strongly impacts the overall performance of mining association rules (Agrawal, 1994).

1.1.4 Frequent Itemsets and Closed Itemsets

An itemset is frequent if its support is above or equal to a user-specified support threshold. An itemset is closed if none of its proper supersets has the same support as it has (a formal mathematical definition of a closed itemset is given in Chapter 3). A closed frequent itemset is an itemset that is both frequent and closed.

1.1.5 Frequent Pattern Mining and Closed Pattern Mining

As discussed in Section 1.1.3, frequent pattern mining is a crucial step of mining associations. A number of methods have been proposed and developed for frequent pattern mining in various kinds of databases, including transaction databases and time series databases. These methods can be roughly classified into two groups: frequent pattern mining and closed pattern mining.

The process of discovering the entire collection of frequent itemsets is called frequent pattern mining. Mining all frequent patterns often generates a large number of frequent itemsets due to the following combinatorial reality: for any collection of frequent itemsets, their subsets are also frequent. For example, assume the itemset $\{a, b\}$ has a frequency of three. Therefore, the subsets of this itemset, which are $\{a\}$ and $\{b\}$, also are frequent patterns with a support of at least three.

Closed pattern mining is a process of discovering the entire collection of closed frequent itemsets, which is generally a small subset of the complete set of frequent itemsets (Pasquier, 1999). Referring back to the example in the previous paragraph, because items $\{a\}$ and $\{b\}$ both have a support of three, and the itemset $\{a, b\}$ also has a support of three, then we conclude that the items $\{a\}$ and $\{b\}$ are not closed relative to a support value of three.

1.1.6 Association Mining in Data Streams based on Closed Pattern Mining

From the above discussions, we can see that the purpose of association mining in data streams based on closed pattern mining is to discover interesting associations among closed patterns in a given data stream. Similar with the process of discovering associations based on frequent pattern mining, it is a two-step process.

Step 1: Find all *closed patterns*. By definition, each of these closed patterns will occur at least as frequently as a user-specified minimum support count.

Step 2: Generate strong association rules above user-specified support and confidence thresholds from the *closed patterns*.

Closed pattern mining (Step 1) is a crucial step of the process, and its computational efficiency strongly impacts the overall performance of the mining process.

Many researchers have been discussing the theoretical foundations and complexity of closed pattern and association mining including (Zaki 1998, Wijsen 1998, Angiulli 2004, Yang 2004). In the following study, we focus is not on asymptotic complexity analysis, but rather we focus on discovering and applying the closed pattern and association mining in practical data streaming applications.

1.2 Motivation

As the number of data streaming applications grows, there is an increasing need to perform association mining in data streams. One example application is to estimate missing data in sensor networks (Halatchev, 2005). Another example application is to predict frequency of Internet packet streams (Demaine, 2002). In the MAIDS project (Cai, 2004), an association mining technique is used to find alarming incidents from data streams. Association mining can also be applied to monitor manufacturing flows (Kargupta, 2004) to predict failures or generate reports based on accumulated web log streams.

Traditional association mining algorithms are developed to work on a complete static dataset and, thus, cannot be applied directly to mine associations in data streams. A number of association mining techniques for data streams have been developed recently, and most of them are based on mining frequent patterns, the number of which might be huge due to the number of redundant and non-informative patterns that they contain. Thus, these types of approaches are not always efficient for data streaming applications. An alternative approach is to mine closed patterns,

which generally represent a small subset of all corresponding frequent patterns, but provide complete and condensed information. Once the closed patterns are determined, then non-redundant and informative associations can be found based on these closed patterns.

Our motivation for developing the proposed closed pattern and association mining technique are as follows. First, the number of closed patterns for a given collection of data items is generally much smaller than the corresponding set of frequent patterns for the same data items. Thus the approach has potential to largely decrease the number of subsequent combinatorial calculations performed on the patterns. Second, because the number of closed patterns is generally smaller than the corresponding number of frequent patterns, memory usage is reduced. Third, associations generated from closed patterns contain non-redundant information, which is more easily understandable. Therefore, the objective of this study is to develop an efficient closed pattern mining technique for data streams, and to derive non-redundant and informative association rules based on the discovered closed patterns.

Due to the characteristics of streaming data, there are some inherent challenges and issues need to be considered for association mining in data streams. First, due to the continuous, unbounded, and high speed characteristics of data streams (Guha, 2001), they contain a huge amount of data, and thus, there is usually not enough time to rescan the whole database or perform multiple scans whenever an

update occurs, as in traditional data mining algorithms. This is especially true in online data streaming applications, which require real-time updated results. Furthermore, there is often not enough space to store all the streaming data for processing over the entire dataset. Therefore, the single scan of data and compact memory usage of the association mining technique are preferable. Second, the mining method of data streams needs to adapt to the changing data value distribution; otherwise, it may result in what is known as the "concept drifting problem" (Wang, 2003) – as new streaming data arrives, the patterns which are previously frequent or closed may become infrequent and unclosed, and vice versa – and not perform well when the mining concepts changes dramatically. Third, due to the high speed characteristics of online data streams, they need to be processed as fast as possible; the speed of the mining algorithm should be faster than the data input rate. Otherwise, data approximation techniques, such as sampling and load shedding, must be applied and these generally decrease the accuracy of the mining results. Fourth, due to the high update rate characteristics of streaming data, mining of data streams is better performed as an incremental process. In other words, the new iterations of mining results are incrementally built based on old mining results combined with newly received items so that the results will not have to completely be recalculated each time a user's querying request is received.

The proposed technique is applied to sensor network databases of a traffic management and an environmental monitoring site for missing data estimation

purpose, in which data missing by a sensor is estimated using the data generated by its related sensors.

1.3 Research Contributions

In this research we developed an incremental closed pattern mining technique for data streams. By mining closed patterns, which are generally much smaller subsets of the corresponding frequent patterns, this technique has potential to largely decrease the size of the subsequent combinatorial calculation performed on the patterns, which could be more serious in the streaming environment because of the huge amount of streaming data. Also, by storing complete and compact information, the technique reduces memory usage while still providing complete information to fulfill different users' requests. Different from the closed pattern mining techniques on traditional databases, which require multiple scans of the entire database, the proposed technique determines the closed patterns with a single scan. It is an incremental mining process; as the sliding window advances, new data items enter and old data items exit the window. But instead of regenerating closed patterns from the entire window, it updates the old set of closed patterns whenever a new transaction arrives and/or an old transaction leaves the sliding window to obtain the current closed patterns. This incremental feature allows the user to get the most recent updated closed patterns without rescanning the entire updated database, which saves not only the computation time, but more importantly, the I/O operating time to load and write data from database to memory. Furthermore, the proposed sliding

window technique can handle both the insertion and deletion operations independently, which allows the user to adjust the sliding window size in different application environments.

We then developed an association mining framework in data streams to derive interesting associations based on the discovered closed patterns. The associations generated from closed patterns contain non-redundant and complete information, which are more useful and concise for data analysis than the associations generated based on frequent patterns (Zaki, 2000). Based on the users' querying requests, different sets of non-redundant and correlated association rules which contains user interested input and output patterns can be generated at the same time with users' specified support and confidence thresholds.

Furthermore, a data estimation algorithm based on our proposed association rule mining technique is developed for sensor network database applications of a traffic management and an environmental monitoring site to first identify the related sensors, and then compute the estimated values of missing data from a sensor by using the data generated by its related sensors. This technique enables us to find out the relationships between two or more sensors when they have the same or different values, therefore it can improve the estimation accuracy compared to the existing technique in the literature which tracks relationships between two sensors when they report the same value, while still achieving both time and space efficiency.

1.4 Dissertation Structure

The rest of this dissertation is arranged as follows. Chapter 2 reviews the related work. This chapter is divided into three major sections that correspond to the background materials relevant to the work presented in Chapters 4, 5, and 6, respectively. Chapter 3 presents preliminary concepts and definitions that are used throughout the remainder of the dissertation. Chapter 4 introduces the main contribution of the dissertation, which is the development of the sliding window algorithm for closed pattern mining in data streams. Chapter 5 describes the association mining framework based on closed pattern mining developed in Chapter 4. Chapter 6 illustrates how the association mining based on closed patterns can be applied to sensor network database applications for missing data estimation purpose. Chapter 7 describes the simulation experiment results of the proposed work and comparing it with the existing literatures. Chapter 8 summarizes the work that has been done, outlines directions of future work, and concludes the dissertation.

2 Related Work

In this chapter, the existing literatures are reviewed for three main areas: data pattern mining, association mining, and missing data estimation. These are covered in three sections, and provide the relevant background for the discussions in Chapters 4, 5 and 6, respectively.

2.1 Data Pattern Mining

2.1.1 Frequent Pattern Mining on Static Data

Traditional frequent pattern mining algorithms are developed to work on static data and, thus, are not suitable to be used for frequent pattern mining in online data streaming applications. The first recognized frequent pattern mining algorithm for traditional databases is Apriori (Agrawal, 1993). The Apriori Algorithm finds the frequent patterns by repeating the following steps through multiple scans of the database. At step k, it finds the frequent k-itemsets. The set of all frequent k-itemsets is denoted by L_k. Then the candidate k+1 frequent itemsets, denoted by C_{k+1}, are generated by combining all combinations of itemsets in L_k. Finally, in the prune phase, any k-itemset that is not frequent and cannot be included in L_{k+1} is removed from C_{k+1}.

Before describing the Apriori Algorithm further, we introduce standard notation for itemsets and frequent itemsets. For convenience, an itemset $\{a, b\}$ is

denoted simply as ab. Furthermore, if the itemset $\{a, b\}$ has a frequency of 3, then this is conveyed using the notation ab^3.

To illustrate the Apriori Algorithm, let us examine the following example. Assume that we have a transaction database ST1 as in Table 2-1, and the user-specified support threshold is 2, which corresponds to 40% in this case because there are five transactions. During the first scan of the database, we find the set of all the frequent 1-itemsets, which is denoted by L_1. L_1 contains all the frequent 1-itemsets whose frequency are equal or above the user-specified threshold 2, in this case $L_1 = \{a^3, c^4, d^2, e^4, f^4\}$. Then the candidates of frequent 2-itemsets are generated by combining all combinations of itemsets in L_1. The candidate set is denoted as C_2, in this case $C_2 = \{ac, ad, ae, af, cd, ce, cf, de, df, ef\}$. Next, in the prune phase, we find the counts of itemsets in C_2: $\{ac^2, ad^2, ae^2, af^2, cd^2, ce^3, cf^3, de^1, df^2, ef^3\}$. Any 2-itemset that is not frequent and cannot be included in L_2 is removed from C_2. The resulting set of L_2 is as follows: $\{ac^2, ad^2, ae^2, af^2, cd^2, ce^3, cf^4, df^2, ef^3\}$. Repeating the same operations, we get the result set for L_3 as $\{acd^2, acf^2, adf^2, cdf^2, cef^3\}$, L_4 as $\{acdf^2\}$, and L_5 as ϕ. The Apriori Algorithm terminates when the resulting set reaches empty. Combining all the frequent patterns derived, we get the set of frequent patterns for database ST1: $\{acdf^2, acd^2, acf^2, adf^2, cdf^2, cef^3, ac^2, ad^2, ae^2, af^2, cd^2, ce^3, cf^4, df^2, ef^3, a^3, c^4, d^2, e^4, f^4\}$.

Transaction ID	Items in Transaction
1	*a, c, d, e, f*
2	*a, b, e*
3	*c, e, f*
4	*a, c, d, f*
5	*c, e, f*

Table 2-1: Sample transaction database ST1

After Apriori Algorithm was introduced in 1993 (Agrawal, 1993), many other algorithms based on the ideas of Apriori were developed for performance improvement (Agrawal 1994, Inokuchi 2000, Yoshio 2002). Apriori-based algorithms require multiple scans of the entire database, which lead to high CPU and I/O costs. Therefore, they are not usually suitable for online data streaming applications, in which data is generally scanned and/or processed only once.

Another category of frequent pattern mining algorithms for traditional databases proposed by Han and Pei (Han, 2000) are those using a frequent pattern tree (FP-tree) data structure and an FP-growth Algorithm, which allows mining of frequent itemsets without generating candidate itemsets. In the FP-growth Algorithm, the FP-tree is used to store the compressed and important information about frequent patterns. FP-growth is an FP-tree-based mining method for mining the entire collection of frequent patterns by pattern growth.

To illustrate the FP-tree data structure and the FP-growth Algorithm, let us consider the application of the FP-growth Algorithm on the same transaction

database ST1 as defined in Table 2-1. Also, as was previously the case, we assume the user-specified support threshold is 2. During the first scan of the database, the algorithm collects the count for each item and eliminates those items whose supports do not pass the user-specified support threshold. The resulting set after the first step is as follows: $\{a^3, c^4, d^2, e^4, f^4\}$. Then the database ST1 is scanned a second time. For each transaction, the algorithm filters out the infrequent items and sorts the remaining ones in frequency descending order, and the revised patterns are inserted into the FP-tree as a branch. In this case the patterns stored in the FP-tree are shown in Figure 2-1.

Before describing the FP-growth Algorithm further, we introduce the standard notation for patterns stored in an FP-tree. For convenience, an item a with support 1 is denoted simply as a^1. Furthermore, when the items in a branch of FP-tree have the same or different support as shown in Figure 2-1, we denote the patterns stored in the FP-tree as: $\{f^4e^3c^3a^1d^1, f^4c^1a^1d^1, e^1a^1\}$.

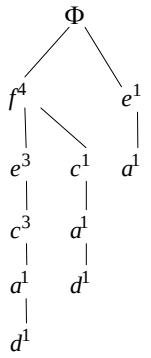

Figure 2-1: The FP-tree structure

The constructed FP-tree is then mined from bottom to top. Starting from *d*, for each frequent 1-itemset, its conditional pattern base is constructed. A conditional pattern base for an itemset contains the transactions that end with that itemset. Then the conditional pattern base is regarded as a transaction database and based on that, the conditional FP-tree is built.

Take item *d* as an example. Item *d*'s conditional pattern base is: {$f^1e^1c^1a^1$, $f^1c^1a^1$}. In this conditional pattern base, *e* occurs only once and thus is eliminated. The conditional FP-tree is constructed as {$f^2c^2a^2$}. There is only one branch in *d*'s conditional FP-tree. The possible combinations are {$fcad^2$, cad^2, fad^2, ad^2, fcd^2, cd^2, fd^2}. In the same way, we can get item *a*'s conditional FP-tree and generate the frequent patterns as {fca^2, ca^2, ea^2, fa^2}. The frequent patterns generated based on item *c*'s conditional FP-tree are {fec^3, ec^3, fc^4}, and the frequent patterns generated based on item *e*'s conditional FP-tree are {fe^3}. Combining the frequent 1-itemsets generated during the first database scan, we get the same set of frequent patterns for transactional database ST1: {$fcad^2$, cad^2, fad^2, ad^2, fcd^2, cd^2, fd^2, d^2, fca^2, ca^2, ea^2, fa^2, a^3, fec^3, ec^3, fc^4, c^4, fe^3, e^4, f^4}.

There are two advantages of the FP-growth Algorithm compared to the Apriori Algorithm. First, the FP-tree is usually smaller than the original database and thus, saves the costly database scans in the mining process. Second, it applies a pattern growth method that avoids candidate generation. Compared with Apriori-based algorithms, the FP-growth Algorithm achieves higher performance by

avoiding iterative candidate generations. However, it still is not practical to mine associations in data streaming applications because the construction of FP-tree requires two scans of the entire dataset.

2.1.2 Frequent Items Mining on Streaming Data

One of the most basic problems associated with mining streaming data is to find the most frequently occurring items in a data stream. It is a challenge to find and maintain frequent items over a data stream because the stream of data can be huge and comes continuously, so memory intensive solutions associated with traditional approaches, such as keeping a counter for each distinct element (like in the Apriori Algorithm) or sorting the stream (required by the FP-growth Algorithm), are infeasible. Furthermore, the stream of data often comes with rapid speed, and thus, it is desirable that the analysis can be done online in one pass as the data arrives.

The Frequent Algorithm (Karp 2003) is a two pass, exact algorithm for finding frequent items above a user-specified threshold *s*. It is noted that the Frequent Algorithm does *not* find frequent itemsets, but only finds frequent (individual) items, i.e., 1-itemsets. The Frequent Algorithm requires that the total number of items to be processed, denoted by *N*, is known. The first pass can be processed as an online processing algorithm; after the first pass over the *N* data items, the set of candidate items, denoted as *K*, is found, which contains items with frequency over the user-specified threshold *s*, possibly among other items. Once the

set K is determined in the first pass, with a second pass, the items in K that have frequency less than sN are deleted.

Take the sample transaction database ST2 of Table 2-2 as an example. In this context, there are 6 transactions in ST2, and assume that the user-specified threshold s is 25%. That means we want to find all those transactions that appear more than 25% of the time. The sampling-based Frequent Algorithm identifies a set K of $1/s$ symbols, in this case 1/0.25 = 4 memory cells. During the first step, it sets up a counter for each transaction $\{a^1, f^1, c^1, d^1\}$. When the 5^{th} transaction arrives, the count of f increases, and the set K contains: $\{a^1, f^2, c^1, d^1\}$. When the 6^{th} transaction arrives, the set of K is updated as: $\{a^1, f^2, c^1, d^1, g^1\}$. As the memory cells exceed 4 and go to 5, the algorithm decreases each counter by 1, and eliminates the cells whose counts are zeros. Therefore the resulting set K is $\{f^1\}$. During a second scan of the database, we can find f's exact support, which is $s = 2/6 = 33.3\%$.

Transaction ID	Items in Transaction
1	*a*
2	*f*
3	*c*
4	*d*
5	*f*
6	*g*

Table 2-2: Sample transaction database ST2

From the above discussion, we see that the Frequent Algorithm requires two passes of the data. It can maintain the possibly frequent items dynamically online, but cannot provide the exact frequent items and their counts dynamically online. The Frequent Algorithm cannot handle deletion operation in data streams, because the counters are incremented whenever their corresponding items are observed and decremented when the size of K is greater than $1/s$, and it preserves only a part of sample data. Furthermore, the length of the data stream couldn't be too long for the second offline algorithm to run, due to the corresponding memory or hard disk space it needs to store the data stream offline. Therefore, it is undesirable for the time sensitive applications, especially in the online data streaming applications.

Count Sketch Algorithm, proposed in (Charikar, 2004), is a single pass algorithm for estimating the most frequent items in a data stream using limited storage space. It can estimate the frequencies of all the items in a data stream using a data structure called Count Sketch. It returns the items whose frequencies satisfy a user-specified threshold with high probability. For each element, the algorithm uses the Count Sketch data structure to estimate its count, and keeps a heap of the top k elements seen so far.

Count Sketch Algorithm is a hash-based algorithm. It needs one pass over the data. The output of the Count Sketch Algorithm is approximate; however, a user-specified output error is guaranteed. The user needs to define pre-specified

parameters before running the algorithm, which are the maximum allowable error ε, and the heap parameter *k*.

Count Sketch Algorithm requires the user to know the data range of the input data stream, which is not applicable in some cases where the received data range is not known. Also, the Count Sketch Algorithm does not handle deletion operation because it preserves only a part of the sample data which is the top *k* frequent items. Suppose that an item that is currently frequent is subject to a number of deletions so that it is no longer among the most frequent items. In this case, it is not possible, using this algorithm, to retrieve items from the past that have consequently become frequent.

2.1.3 Frequent Itemsets Mining on Streaming Data

In (Manku 2002, Chang 2003, Jin 2003, Yang 2004, Dang 2007), the authors proposed algorithms to find frequent patterns over the entire history of data streams. In (Giannella 2003, Chang 2004, Lin 2005, Mozafari 2008), the authors use different sliding window models to find recently frequent patterns in data streams. These algorithms focus on mining frequent patterns, instead of closed patterns, with one scan over the entire data stream.

Lossy Counting Algorithm is proposed in (Manku, 2002). It is a one pass, landmark model[1], incremental algorithm using an in-memory data structure. The

[1] The landmark model mines all frequent itemsets over the entire history of streaming data from a specific time point called landmark to the present [Zhu, 2002].

mining result is approximate, and the error is guaranteed through a user-specified error parameter. The algorithm proceeds as follows.

The data structure D is a set of entries of the form (x, f, e), where x is an element in the stream, f is an integer representing its estimated frequency, and e is the maximum possible error in f. Initially, D is empty. The user-specified parameters are a support threshold $s \in (0, 1)$, and an error parameter $\varepsilon \in (0, 1)$, such that $\varepsilon << s$. N denotes the current length of the stream. The Lossy Counting Algorithm divides the incoming transaction stream into buckets, where each bucket consists of $w = \left\lceil \frac{1}{\varepsilon} \right\rceil$ transactions. Buckets are labeled with bucket identifiers, starting from 1. The current bucket identifier is denoted by $b_{current}$. Whenever a new element x arrives, the algorithm first determines whether an entry for x already exists or not. If the look up succeeds, it updates the entry by incrementing its frequency f by one. Otherwise, it creates a new entry of the form (x, 1, $b_{current} - 1$). It also prunes D, by deleting some of its entries at bucket boundaries, i.e., whenever $N = 0 \bmod w$. The rule for deletion is: an entry (x, f, e) is deleted if $f + e \leq b_{current}$. When a user requests a list of items with threshold s, it outputs those entries in D where $f \geq (s\text{-}\varepsilon) N$.

The Lossy Counting Algorithm computes frequency counts in a single pass with the output error guaranteed not to exceed a user-specified parameter ε. It is an incremental algorithm. The disadvantage of the Lossy Counting Algorithm is that its output is approximate, and the users need to define the pre-specified parameters

before running this algorithm, which are the minimum support *s*, the maximum allowable error ε, and the probability parameter *e*.

In the estDec Algorithm (Chang, 2003), a method of finding recent frequent itemsets adaptively over an online data stream is proposed. It uses a one pass algorithm to maintain the occurrence count of a significant itemset that appears in each transaction using a prefix-tree lattice structure in main memory. The effect of old transactions on the current mining result is diminished by decaying the old occurrence count of each itemset as time goes by.

In the estDec data structure, every node in a monitoring lattice maintains a triple (*cnt*, *err*, *MRtid*) for its corresponding itemset *X*. The count of the itemset *X* is denoted by *cnt*. The maximum error count of the itemset *X* is denoted by *err*. Finally, the transaction identifier of the most recent transaction that contains the itemset *X* is denoted by *MRtid*. The estDec method is composed of four phases: parameter updating phase, count updating phase, delayed-insertion phase and frequent itemset selection phase. When a new transaction is generated in a data stream, the total number of transactions in the current data stream is updated in the parameter updating phase. In the count updating phase, the counts of those itemsets in a monitoring lattice that appear in the new transaction are updated. After all of these itemsets are updated, the delayed-insertion phase is started in order to find any new itemset that has a high possibility to become a frequent itemset in the near future. The frequent itemset selection phase is performed only when the mining result of the

current dataset is required. A force-pruning operation is performed periodically or when the current size of a monitoring lattice reaches a user-specified threshold to prune all insignificant itemsets.

With the estDec Algorithm, the recent change of information in a data stream can be adaptively reflected to the current mining results of the data streams. The weight of information in a transaction of a data stream is gradually reduced as time goes by while its reduction rate can be flexibly controlled. Due to this reason, no transaction needs to be maintained physically. The disadvantage of the estDec Algorithm is that it is an approximate algorithm; its processing time is flexibly controlled while sacrificing its accuracy. Also its output error is not guaranteed.

The hCount Algorithm is proposed in (Jin, 2003). It maintains a hash table and uses h hash functions to map a digit from (0..M-1) to (0..m-1) uniformly and independently. The algorithm checks and outputs the itemsets with frequency above a user-specified threshold s along with their estimated frequencies.

The hCount Algorithm can output a list of most frequent itemsets with a relatively small usage of memory space. It can handle both insertion and deletion of itemsets, and does not request the pre-knowledge on the value range of a data stream. The disadvantage of the hCount Algorithm is that its output is approximate, and users need to define pre-specified parameters before running the algorithm, which are the frequency threshold and the maximum allowable error.

In (Yang, 2004), the authors proposed an algorithm that uses limited computer memory to keep frequency counts of all short itemsets. Its objective is to find those frequent itemsets and association rules, the lengths of which are not longer than a pre-defined length *k*. It introduces a method to keep frequency counts of all short itemsets and to discover association rules from the short frequent itemsets. This method uses an array to keep frequency counts of all short itemsets. A bijection between itemsets and array elements is set up. Itemsets are arranged in the array so that new items can be inserted at any time during the mining process.

Given n items, there are $C(n, k)$ k-itemsets and $\sum_{i=0}^{k} C(n,i)$ up-to-k-itemsets, which denote any i-itemset where $i \leq k$. With a 32-bit modern computer which addresses 4GB memory space, k can be chosen as up to 3 when the value n is less than 1,800. The frequency counts of all up-to-k-itemsets are stored in memory. They are arranged in a pre-defined order and then an array is used to keep these frequency counts. With this pre-defined order, when inserting a new item, it only needs to extend the list of itemsets at its end to include the up-to-k-itemsets containing the new itemset. The ranks of all existing itemsets in the order do not need any change.

This method is simple, fast, and capable of online and data stream mining. It takes one pass over the data, and keeps all the short itemsets (itemsets with $k \leq 3$, where k is the maximum size of frequent itemsets) and their frequency counts in memory. The drawback of this algorithm is that it is only suitable for mining small database which n is less than 1,800 and $k \leq 3$.

In (Lin, 2005), the authors propose an approach for frequent pattern mining in data streams based on a time-sensitive sliding window model. It consists of a storage structure that captures all possible frequent itemsets and a table providing approximate counts of the expired data itemsets, the size of which can be adjusted by the available storage space.

A data structure called Discounting Table (DT) is devised to retain the frequent itemsets with their support counts in the individual basic blocks of the current time-sensitive sliding window. Another data structure named Potentially Frequent-itemset Pool (PFP) is used to keep the potential frequent itemsets, which are not frequent in the last time-sensitive sliding window, but are frequent in the current transaction block. The time-sensitive sliding window model divides the data stream into blocks by time. The support count threshold for each basic block is computed and stored into an entry in the Threshold Array (TA). Only sliding window size entries are maintained in the TA. An algorithm to mine frequent itemsets is applied to the transactions in the buffer. Each frequent itemset is inserted into PFP in the form of (*ID*, *Items*, *Acount*, *Pcount*), recording a unique identifier, the items in it, the accumulated count, and the potential count, respectively. Each itemset in PFP is inserted into Discounting Table (DT) in the form of (*B_ID*, *ID*, *B_count*), recording the serial number of the current basic block, the identifier in PFP, and its support count in the current basic block. Basically there are five steps to run this algorithm: In Step 1, the incoming data are stored in the buffer; in Step 2, the itemsets are discounting by DT, the min or max function is used to maintain the DT

through self adjustment-merge. In Step 3, new itemsets are inserted and old itemsets are updated; in Step 4, the potential counts are estimated by TA and TA is updated; and finally in Step 5, the frequent itemsets are output.

The time-sensitive sliding window approach takes one pass over the raw data. It uses a time-sensitive sliding window model, which can answer time-sensitive queries asked by the user within the time sliding window, and guarantees no false dismissal or false alarm of the mining result. A mechanism to self-adjust the DT under the memory limitation is presented. It can handle both insertion and deletion of the data transactions, and the output error is guaranteed. The disadvantage of the time-sensitive sliding window approach is that it stores duplicate information in different data structures (DT and PFP) for each itemset, which will take more space to store the redundant information. Although the authors developed a mechanism to adjust the DT when memory is limited, it sacrifices the accuracy of this algorithm.

In (Dang, 2007), the authors propose an algorithm called EStream that allows online processing of streaming data and guarantees the support error of frequent patterns within a user-specified threshold. In (Mozafari, 2008), the authors propose frequent itemset mining method for sliding windows by using a verification technique, called verifier. Two verifiers and a hybrid verifier are used to mine frequent itemsets.

All the above algorithms focus on mining frequent itemsets, instead of closed frequent itemsets over streaming data, which could result in redundancy on both the data patterns and the derived associations based on these data patterns.

2.1.4 Closed Pattern Mining on Static Data

The concept of closed frequent itemsets was first introduced by Pasquier et al in 1999 (Pasquier, 1999). It is well known that mining the entire collection of frequent patterns often generates a large number of frequent itemsets, among which users have to search through to find useful ones. For example, the set of frequent patterns $\{a^3, b^3, ab^3\}$ can be more simply represented by $\{ab^3\}$, from which we can observe that the total number of closed frequent itemsets is a smaller subset of their corresponding frequent itemsets. Furthermore, all frequent itemsets can be derived from closed frequent itemsets. Because a frequent itemset must be a subset of one (or more) closed frequent itemset(s), and its support is equal to the maximal support of those closed itemsets that contain the frequent itemset, mining frequent closed itemsets provides complete and condensed information for frequent pattern analysis. More importantly, associations extracted from closed sets have been shown to be more meaningful for analysis because all redundancies are discarded (Zaki, 2000).

A-close (Pasquier, 1999) is a variation of Apriori. It adopts the Apriori framework, but looks for frequent closed itemsets and prunes the frequent itemsets that are not closed. The mining process of A-close is as follows. First, A-close scans the database and finds all frequent itemsets. Then, the Apriori heuristic is applied to

generate all candidate 2-itemsets. In the second scan of the database, A-close counts the supports of candidate 2-itemsets and derives the frequent 2-itemstes. Itemsets that are not frequent are pruned during this scan. In the third scan of the database, A-close collects the supports for the candidate 3-itemsets and finds that they are frequent or not. The iterative candidate generation-and-testing process terminates until no frequent itemsets are found. In order to generate the frequent closed itemsets, A-close applies one more scan to compute the closures for all of the surviving frequent itemsets. The closure of a frequent itemset is the intersection of all transactions containing the itemset. The set of closures, after removing duplications, is the set of frequent closed itemsets.

A-close scans the transaction database multiple times. The major cost of the A-close is from two aspects. First, it has to generate a lot of candidates and scan the transaction database multiple times to count candidates. Second, in the last scan to compute closures, there could be a large number of surviving frequent itemsets. This makes the closure computation costly.

Charm (Zaki, 2002) is another algorithm to find closed frequent itemsets. Different from A-close, Charm explores a vertical data format, i.e., each item is associated with a set of transaction identifiers (tid for short). Charm does not use the Apriori framework. Initially, Charm builds a tree with multiple branches, corresponding to the number of frequent items. The item, as well as the transaction identifiers in which the item appears, is registered in the corresponding node. Then

Charm attempts to combine items in the same layer to form itemsets. When it combines, it computes the intersection of the sets of transaction identifiers of the two itemsets (called tid set). If the combined itemset does not have enough support, it is pruned. The efficiency of Charm is from the fact that the tid set of a superset itemset is derived from those of its subsets. It is easy to check whether they are identical. The major cost of Charm originates from the fact that it has to compute intersections of tid sets repeatedly in each combination step.

Closet (Pei, 2003) is an algorithm proposed for mining closed frequent itemsets. In the first step, it finds frequent items by scanning the entire database. The items are sorted in descending support order. Then, it divides the search space. All the frequent closed itemsets can be divided into non-overlapping subsets based on the item list derived in the first step. In the third step, it mines the subsets of frequent closed itemsets by constructing corresponding conditional pattern bases and mining each recursively.

All the above works are developed to mine closed itemsets for traditional static databases, where multiple scans are needed and whenever new transactions arrive, additional scans must be performed on the updated transaction database. Therefore, they are not suitable for data stream mining.

2.1.5 Closed Pattern Mining on Streaming Data

In (Chi, 2004), Chi et al considers the problem of mining closed frequent itemsets over a data stream sliding window in the Moment Algorithm. A synopsis

data structure is designed to monitor transactions in the sliding window so that it can output the current closed frequent itemsets at any time. A compact data structure, the Closed Enumeration Tree (CET) is introduced to maintain a dynamically selected set of itemsets over a sliding window. Moment Algorithm visits itemsets in lexicographical order. If a node is found to be infrequent, then it is marked as an infrequent gateway node. The support and tid_sum of an infrequent gateway node have to be stored because they will provide important information during a CET update when an infrequent itemset can potentially become frequent. When an itemset I is found to be non-closed because of another lexicographically smaller itemset, then n_I is an unpromising gateway node. In Explore, leftcheck(n_I) checks if n_I is an unpromising gateway node. It looks up the hash table to see if there exists a previously discovered closed itemset that has the same support as n_I and also subsumes I. And if so, it returns true (in this case n_I is an unpromising gateway node); otherwise, it returns false (in this case n_I is a promising node). If a node n_I is found to be neither infrequent nor unpromising, then the algorithm explores its descendants. After that, it can be determined if n_I is an intermediate node or a closed node.

Moment is an incremental algorithm. It takes one pass over the raw data, and can handle both addition and deletion of the data transactions. The output error is guaranteed. The disadvantage of Moment Algorithm is that it maintains not only closed frequent itemsets, but also additional boundary nodes which increase the memory usage as well as the computation time. And in (Li, 2006), the authors

proposed the NewMoment Algorithm which uses a bit-sequence representation of items to reduce the time and memory needed.

In Chapter 4, we propose an algorithm called CFI-Stream (Jiang, 2006), to directly compute closed itemsets online and incrementally without the help of any support information. Nothing other than closed itemsets is maintained in the derived data structure. When a new transaction arrives, it performs the closure check on the fly; only associated closed itemsets and their support information are incrementally updated. The current closed frequent itemsets can be output in real time based on any user-specified thresholds. And in (Li, 2008), Li et al proposed to improve the CFI-stream Algorithm with bitmap coding named CLIMB (Closed Itemset Mining with Bitmap) over data stream's sliding window to reduce the memory cost. We then use the discovered closed frequent itemsets to mine associations in data streams.

2.2 Association Mining

2.2.1 Association Mining based on Frequent Pattern Mining

There has been a lot of research in developing efficient association mining algorithms for static data. The first recognized association mining algorithm for traditional databases is Apriori (Agrawal, 1993).

Apriori is an influential algorithm for mining association rules, and a step-wise algorithm. It generates the candidate itemsets to be counted in the pass by using only the itemsets found frequently in the previous pass. The algorithm consists of

two steps, a join step and a prune step. In the join step, join L_{k-1} with L_{k-1}. In the prune step, delete all itemsets $X \in C_k$ such that some (k-1)-subset of X is not in L_{k-1}. During each iteration, only candidates found to be frequent in the previous iteration are used to generate a new candidate set during the next iteration. The candidate itemsets having k items (called candidate k-itemset) can be generated by joining frequent itemsets having k-1 items and deleting those itemsets that contain any subset that is not frequent. The algorithm terminates when there are no frequent k-itemsets. Apriori-based algorithms require multiple scans of the original database, which lead to high CPU and I/O costs. Therefore, they are not suitable for the data streaming environment, in which data can be scanned only once.

From the above discussions, we can see that traditional association mining techniques are not suitable for the data streaming environment due to several reasons. First, a huge amount of streaming data continuously arrives which produces massive rules; the cost of calculation to find association rules is high and they may not reflect the current situation. Second, traditional association rule mining algorithms perform multiple scans over the database, which is not suitable to apply to the data streaming environment that prefers a single scan. Furthermore, due to the continuous, unbounded, and high speed characteristics of data streams, there is a huge amount of data in both offline and online data streaming applications, and thus, there is not enough time to rescan the whole database or perform a multi-scan as in traditional data mining algorithms whenever an update occurs. Third, the mining

method of data streams needs to adapt to their changing data value distribution because the streaming data value distribution is usually changing with time.

Association mining technique based on frequent patterns produces many rules. With a large amount of rules being produced, the cost of calculation to find association rules is high. Also, it is difficult to evaluate the large amount of associations which may or may not all be meaningful to the end users. To solve these problems, many studies have been done. In (Toivonen 1995, Liu 1999), the authors proposed techniques to prune and summarize the discovered associations. In (Klemettinen 1994, Ng 1998, Liu 1999, Bayardo 1999), the authors proposed techniques to mine the most interesting rules incorporated with the user-specified constrains or defined by the object metrics of interest. But still they are aimed for the traditional databases and, thus, do not fit the data streaming environment. Furthermore, they do not address rule redundancy.

2.2.2 Association Mining based on Closed Pattern Mining

In (Bastide, 2000), the authors proposed the concept to mine minimal antecedent and maximal consequent association rules with the same support and same confidence. Using the closure of the Galois connection (Taouil, 2000), a generating set for all valid association rules with the support and confidence is setup using frequent closed itemsets and their generators; they consist of the non-redundant association rules having minimal antecedents and maximal consequents. This concept indicates to generate only the most informative rules.

In (Li, 2004), the authors proposed a technique to mine minimal non-redundant association rules from a quantitative closed itemset lattice. However, the algorithm is based on a landmark data processing model and no deletion operation can be performed over the entire history of data streams. Thus, when the amount of data streams is high, the closed itemset lattice can grow rapidly.

In (Zaki, 2005), Zaki et al proposed the concept to mine non-redundant association rules with minimal antecedent and minimal consequent with the same support and same confidence. However, all these association rule mining algorithms are based on the traditional association rule mining framework and require multiple scans, which are not suitable for the stream mining environment.

In (Yang, 2004), (Halatchev, 2005), and (Shin, 2007), the authors proposed using two, three, and multiple frequent pattern based methods to perform association rule mining. Instead of using frequent pattern mining, we proposed to perform association rule mining based on closed pattern mining technique we discussed in Chapter 4, where the rule generation is based on the current closed itemsets in data streams which are a condensed representation of the whole streaming data. Furthermore, the rule can be generated on demand, at different users' querying requests which is preferable in the distributed query processing data streaming environment.

2.3 Missing Data Estimation

Many articles have been published to deal with the missing data problem, and a lot of software has been developed based on these methods. Some of the methods totally delete the missing data before analyzing them, like listwise and pairwise deletion (Wilkinson, 1999), while other methods focus on estimating the missing data based on the available information. The most popular statistical estimation methods include mean substitution, imputation by regression (Cool, 2000), hot deck imputation (Iannacchione, 1982), cold deck imputation, expectation maximization (EM) (McLachlan, 1997), multiple imputations (Rubin 1987, Shafer 1995), etc.

Mean Substitution (Cool, 2000) replaces all missing instances of a given variable with the mean value for that variable. It is a good solution when data is both Missing At Random (MAR) and somewhat normally distributed. If we assume that a missing value for an individual on a given variable is best estimated by the mean for the non-missing observations of that variable, that is to say, for a given item, simply substitute the mean response of all valid cases providing data on that item.

The advantage of this method is that it is easy to implement, while the disadvantage of this method is that the sample size is overestimated. Also, the distribution of new values is an incorrect representation of the population values because the shape of the distribution is distorted by adding values equal to the mean.

Imputation by Regression (Cool, 2000) is the prediction of the missing data based on a regression equation that uses all other relevant variables as predictors.

The advantage of this method is that it preserves the variance and covariance of the variables with missing data. The disadvantage of this method is that if standard errors are ignored when predicting the missing values, it may inflate the predictive power of the model because the missing values of the dependent variables are presented as perfectly predicted.

We can also perform the estimation by developing a regression equation to predict the criterion of a variable with missing data using valid cases, and then apply the equation to the valid scores on other variables of missing scores for that given variable. This estimation is more sophisticated because it takes into account relationships among the variables.

Regression methods rely on the information contained in the non-missing values of variables to provide estimates of the missing values for the variable of interest. Each variable with a missing value, in turn, is treated as a criterion variable and is regressed onto all the other variables having observed values to predict the criterion variable.

The Hot Deck Imputation (Iannacchione, 1982) replaces the missing values with randomly selected values presented in a pool of similar complete cases. Because the replacement values are randomly selected, hot deck imputation introduces the variations seen in the pool of complete cases resulting in fewer tendencies toward the mean. There are two main areas of concern: selecting valid characteristic sets for identifying the potential pools containing values with reasonable variance, and

ensuring that characteristic sets will allow for large enough donor pools with reasonable variance. The technique has been used extensively by government agencies and has been widely accepted as providing accurate samples of study population. The Cold Deck Imputation replaces the missing value by a value that is independent of the dataset. For example, we can replace the missing value with population mean, or expected value under random response.

Expectation Maximization (EM) Algorithm (McLachlan, 1997) is a two step iterative approach that estimates the parameters of a model starting from an initial guess. Each iteration consists of two steps: an expectation step that finds the distribution for the missing data based on the known values for the observed variables and the current estimate of the parameters, and a maximization step that substitutes the missing data with the expected value. The procedure iterates through these two steps until convergence is obtained. Convergence occurs when the change of the parameter estimates from iteration to iteration becomes negligible.

But none of the above approaches is suitable for wireless sensor network environment, where streams of data are constantly sent from the sensors to the server, due to several reasons. First, how much old information should be based on to get the associated information for the missing data estimation? Using all of the old readings to perform the estimation is unreasonable, especially when using an iteration procedure until convergence to get the estimation like in the EM Algorithm. On the other hand, using only the previous round of sensor readings to perform the

estimation is also not a good choice because data streams often have a changing data distribution. Some of the statistical methods use all of the available data points in a database to construct the best possible results, in the wireless sensor networks, the missing sensor data may or may not be related to all of the available information, thus using all of the available information to process the result is not an optimal choice and would consume more time and memory space than necessary.

Second, which information should be used to perform the missing data estimation? In the wireless sensor network, data is collected within certain scopes and reported to the server during a certain period of time. Different sensors have different readings at different time periods. The current readings of one sensor may relate not only to its previous readings, but also to other sensors' previous or current readings. Therefore, it is difficult to replace the missing values with randomly selected values presented in a pool of similar complete cases or with a value which is independent of the dataset like in the hot/cold deck imputation. This is because even though we may get the complete set of information of a certain wireless sensor network, it is not easy to decide which information is similar to the current round of missing sensor's information. In other words, it is hard to draw the pool for a certain sensor's certain round of readings when the application needs to perform the data estimation.

Third, the missing data may or may not miss at random, while most of the statistical techniques are based on the MAR assumption. According to the definition

in (Little, 1987), Data on Y are missing at random if the probability that Y is missing does not depend on the value of Y after controlling other observed variables X. For example, we are modeling weight Y as a function of gender X. One gender may be less likely to disclose its weight, that is, the probability that Y is missing depends only on the value of X. Such data are MAR.

In (Deshpande, 2005), the authors proposed a model, called BBQ to provide efficient query answers in sensor networks. They use probabilistic models to answer queries. Such models can be learned from historical data using standard algorithms, e.g. (Mitchell, 1997). The basic model used in BBQ is a time-varying multivate Gaussians. A multivate Gaussian is the natural extension of the familiar unidimensional normal Probability Density Function (PDF). First, the historical data is used to construct the initial representation of the PDF. Once the initial PDF is constructed, the answer queries can be answered using the model. The model is updated as new observations are obtained from the sensor network, and as time passes. There are various different models that may be more suitable in different environments and for different classes of queries.

There are also some drawbacks of using the probabilistic models to answer the query. First, the probabilistic models are learned from some set of training data. The training data needs to be captured in advance before the model can be used to predict values. In general, a probabilistic model is only as good at prediction as the data used to train it. For models to perform accurate predictions, they must be trained

in the kind of environment where they will be used. Second, the model needs to be continuously updated as time goes by. Third, the suitable model needs to be selected, choosing the best model for the given queries, and environment is another issue that needs to be considered when using this approach.

As more and more data streaming applications emerge, proper data estimation algorithms for streaming data are needed. In (Papadimitriou, 2005), the authors proposed using pattern discovery in multiple time-series to estimate missing data, but it's not well suited for sensor networks, where the relationships between sensors are decided not only by the time trends, but also by some other factors, like locations and so on.

In (Halatchev, 2005), the authors proposed the Window Association Rule Mining (WARM) Algorithm for estimating missing sensor data. WARM uses a modified Apriori Algorithm for association rule mining to identify sensors that report the same data for a number of times in a sliding window, called related sensors, and then estimates the missing data from a sensor by using the data reported by its related sensors. WARM has been reported to perform better than the mean substitution approach where the average value reported by all sensors in the window is used for estimation. However, there exist some limitations in WARM. First, it is based on 2-frequent itemsets modified Apriori association rule mining algorithm, which means it can discover relationships only between two sensors and ignores the cases where missing values are related with multiple sensors. Second, it finds those relationships

only when both sensors report the same value and ignores the cases where missing values can be estimated by the relationships between sensors that report different values. In (Gruenwald, 2007), the authors propose to use two frequent itemset mining technique to perform estimation based on relationship between two sensors. In (Tarui, 2007), the author discussed how to find a duplicate and a single missing item in a stream.

In view of the above challenges, based on our proposed closed pattern and association mining technique discussed in Chapter 4 and 5, we develop a technique to perform missing data estimation based on the relationship between multiple sensor readings. Since as discussed before, association rules based on the closed patterns in data streams contain non-redundant and complete information, based on which relationships between sensor values in data streams can be derived.

2.4 Summary

Table 2-3, Table 2-4 and Table 2-5 summarize the features of the discussed algorithms in Section 2.1, 2.2, and 2.3 respectively.

			Mining Strategy	Mining Process	Data Stream Support	Scan
Mining Frequent Item	Static Data	Karp 03	Sampling based	Offline	No	Two
	Stream Data	Charikar 04	Hash based	Online	Yes	Single
Mining Frequent Itemsets	Static Data	Agrawal 93, Agrawal 94	Candidate based	Offline	No	Multiple
		Han 00	Non-candidate based	Offline	No	Two
	Stream Data	Manku 02, Chang 03, Jin 03, Yang 04, Dang 07	Landmark based	Online	Yes	Single
		Giannella 03, Chang 04, Lin 05, Mozafari 08	Sliding window based	Online	Yes	Single
Mining Closed Itemsets	Static Data	Pasquier 99	Key Pattern Browsing	Offline	No	Multiple
		Pei 00, Zaki 02, Pei 03	Closure Climbing	Online	No	Multiple
	Stream Data	Chi 04, Li 06	Indirect	Online	Yes	Single
		Proposed 06, Li 08	Direct	Online	Yes	Single

Table 2-3: Data pattern mining approaches

				Number of Itemsets	Mining Process	Data Stream Support	Scan
Mining Association Rule	Static Data	Frequent Itemsets	Agrawal 93, Agrawal 94, Liu 99, Han 00	Multiple	Offline	No	Multiple
		Closed Itemsets	Bastide 00, Li 04, Zaki 05	Multiple	Offline	No	Multiple
	Stream Data	Frequent Itemsets	Yang 04, Halatchev 05	Two/Three/ Multiple	Online	Yes	Single
		Closed Itemsets	Proposed 07	Multiple	Online	Yes	Single

Table 2-4: Association mining approaches

				Number of Itemsets	Data Stream Support
Data Estimation	Static Data	Statistics	Iannacchione 82, Rubin 96, Shafer 95, Cool 00	N/A	No
	Stream Data	Probabilistic Models	Deshpande 05	N/A	Yes
		Time Series	Papadimitriou 05	N/A	Yes
		Pattern and Association Mining	Tarui 07	One	Yes
			Halatchev 05, Gruenwald 07	Two	Yes
			Proposed 07	Multiple	Yes

Table 2-5: Data estimation approaches

3 Preliminary Concepts

In this chapter, we describe the notations and definitions that are used throughout this dissertation.

Let $I = \{i_1, i_2, \ldots, i_n\}$ be a set of n items. A subset $X \subseteq I$ is called an itemset. A k-subset is called a k-itemset. Each transaction t is a set of items from I. Given a set of transactions T, the support of an itemset X is the percentage of transactions that contain X. A frequent itemset is an itemset the support of which is above or equal to a user-specified support threshold.

Let T and X be subsets of all the transactions and items appearing in a data stream S, respectively. The concept of a closed itemset is based on the two following functions, f and g: $f(T) = \{i \in I \mid \forall\ t \in T, i \in t\}$ and $g(X) = \{t \in T \mid \forall\ i \in X, i \in t\}$. Function f returns the set of itemsets included in all the transactions belonging to T, while function g returns the set of transactions containing a given itemset X.

An itemset X is said to be closed if and only if $C(X) = f(g(X)) = f \bullet g(X) = X$ where the composite function $C = f \bullet g$ is called Galois operator or closure operator (Taouil, 2000).

Example 3.1 Let $I = \{a, b, c, d\}$ be a set of 4 items, and $T = \{cd, ab, abc, abc\}$ be a set of transactions in data streams, then the closed itemsets are $\{c^3, ab^3, cd^1, abc^2\}$. Each of the closed itemsets X satisfies $C(X) = f(g(X)) = f \bullet g(X) = X$. Take

ab as an example, $g(ab) = \{ab, abc, abc\}$, $f \bullet g(ab) = ab$, so $C(ab) = f(g(ab)) = f \bullet g(ab) = ab$. If the user-specified absolute support threshold is two, then the frequent closed itemsets are $\{c^3, ab^3, abc^2\}$. The frequent itemsets are $\{a^3, b^3, c^3, ab^3, ac^2, bc^2, abc^2\}$, from which we can see that closed frequent itemsets are a smaller subsets of frequent itemsets and contain all itemsets and support information in the frequent itemsets.

From the above discussion, we can see that a closed itemset X is an itemset whose closure $C(X)$ is equal to itself ($C(X) = X$). The closure check is to check the closure of an itemset X to see whether or not it is equal to itself, i.e., whether or not it is a closed itemset. We define a smallest itemset X_1 that satisfies $C(X_1) = X_2$, is called a minimum generator of X_2.

An association rule is an expression $X \xrightarrow{s,c} Y$, where X and Y are interesting itemsets, and $X \cap Y = \phi$. The parameter s represents the support of the rule which is the percentage of records that contain both X and Y in the database ($s = s(X \cup Y) = |g(X \cup Y)|/|T|$), and c is the percentage of records containing X that also contain Y, called the confidence of the rule ($c = s(X \cup Y)/s(X) = |g(X \cup Y)|/|g(X)|$). Association mining is to find all association rules, the support and confidence of which are above or equal to a user-specified minimum support and confidence, respectively (Agrawal, 1993).

An association rule $X_1 \xrightarrow{s1,c1} Y_1$ is equivalent to an association rule $X_2 \xrightarrow{s2,c2} Y_2$, if and only if $X_2 \xrightarrow{s2,c2} Y_2$ can be derived from $X_1 \xrightarrow{s1,c1} Y_1$, and

$s1$=$s2$, $c1$=$c2$ (Zaki, 2005). If $X_1 \xrightarrow{s,c} X_2$, $X_3 \xrightarrow{s,c} X_4$, $X_1 \subseteq X_3$, and $X_4 \subseteq X_2$, we say association rule $X_3 \xrightarrow{s,c} X_4$ is redundant (Bastide, 2000).

4 Closed Pattern Mining in Data Streams

In this Chapter we introduce the proposed method to mine closed frequent itemsets in data streams. First, we give an overview of the proposed algorithm and a data structure, called DIrect Update lattice (DIU), to mine closed frequent itemsets in data streams. Then, the conditions that are needed to check for closed itemsets and how to check for them when performing insertion and deletion operations on the DIU are discussed. Based on this, an online algorithm to discover and incrementally update closed itemsets is developed.

4.1 Overview

The proposed algorithm employs a sliding window, which is a buffer that holds a specified number of transactions that arrive from the input data stream. When a new transaction enters (and/or a previously stored transaction leaves) the sliding window, the algorithm updates the status of all associated closed itemsets' support values, on-the-fly. Current closed itemsets are maintained and updated in real time using a newly proposed data structure, the DIU. The closed frequent itemsets can be output at any time at user-specified thresholds by browsing the DIU.

Different from previous closure check techniques, which require multiple scans over data (Pasquier 1999, Pei 2000, Zaki 2002, Pei 2003), our proposed method performs the closure check on-the-fly with only one scan over the window. It updates only the supports of the closed itemsets associated with the entering (or

exiting) transactions, and it is able to provide real time updated results. The proposed algorithm is an incremental algorithm where we check for closed itemsets and update their associated supports based largely on the previously computed results, thus increasing efficiency and reducing computational and I/O costs.

In contrast with other data stream mining techniques (Manku 2002, Chi 2004, Lin 2005) , the proposed algorithm only stores the information of current closed itemsets in the DIU, which is a compact and complete representation of all itemsets and their support information. The current closed frequent itemsets can be output in real time based on users' specified thresholds by browsing the DIU. Also, the proposed algorithm solves the concept-drifting problem (Wang, 2003) in data streams by storing all current closed itemsets in the DIU from which all itemsets and their support information can be incrementally updated. We discuss the update of the DIU data structure and the closure check procedures for insertion and deletion operations in Section 4.2.

4.2 The Proposed Data Structure

4.2.1 The Direct Update Lattice

A lexicographical ordered direct update lattice is used to maintain the current closed itemsets. Each node in the DIU represents a closed itemset. There are *k* levels in the DIU, each level *i* stores the closed *i*-itemsets. The parameter *k* is maximum size of the current closed itemset. Each node in the DIU stores a closed itemset, its

current support information, and the links to its immediate parent and child nodes. Figure 4-1 illustrates the DIU after four transactions arrive. The support of each node is labeled in the upper right corner of the node itself. The figure shows that currently there are 4 closed itemsets *c*, *ab*, *cd*, *abc* in the DIU, and their associated supports are 3, 3, 1, and 2, respectively.

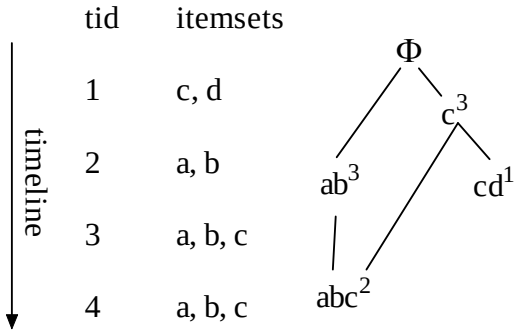

Figure 4-1: The lexicographical ordered direct update lattice

All transactions in the current sliding window are stored in a (FIFO) queue data structure; when the number of transactions exceeds the size of the sliding window, the first transaction that comes into the queue exits the queue to make room for the next arriving transaction to enter the queue.

4.2.2 Insert a Transaction to the DIU

In this subsection, the update and maintenance of the DIU when a new transaction arrives is discussed. The basic result is the derivation of conditions that define which itemsets, in the new transaction, need to be checked for closure and how to decide if it is closed and need to be inserted to the DIU. The efficiency of the

algorithm comes from the fact that not all itemsets need to be checked, but only a subset of itemsets that are related to the arriving transaction.

4.2.2.1 Conditions to Check for Closed Itemsets

First, we define and prove the following conditions in which we need to check whether an itemset is closed or not when a new transaction *t* arrives in the current sliding window.

Table 4-1 shows the following conditions we classify to decide if a closure check is needed when perform the addition operation.

Cases/Conditions			Closure Check
Case 1	Case 1.A	Case 1.A.1	No
		Case 1.A.2	Yes
	Case 1.B		No
Case 2	Case 2.A	Case 2.A.1	Yes
		Case 2.A.2	No
	Case 2.B	Case 2.B.1	No
		Case 2.B.2	No

Table 4-1: Conditions to check for insertion operation

From the above table, we can see that there are two conditions we need to perform closure check, which are as follows.

Condition 1 (Case 1.A.2): When the newly arrived transaction *t equals* {*X*}, X is not closed but has a support larger than zero in the old sliding window. If *X* is currently closed and exists in the DIU, then no closure check is necessary. If *X* does not currently exist in the DIU, then check all of *X*'s subsets *Y* to see whether they are

closed or not in the new sliding window (mathematically, the condition of Case 1.A.2 can be expressed as: $g_{T1}(X) \neq \phi$, $g_{T1}(Y) \neq \phi$, $C_{T1}(X) \supset X$, $C_{T1}(Y) \supset Y$ and $Y \subset X$).

Condition 2 (Case 2.A.1): When the newly arrived transaction t equals $\{X\}$, X has a support of zero in the old sliding window. Check all of X's subsets Y to determine whether they are closed or not (mathematically, the condition of Case 2.A.1 can be expressed as: $g_{T1}(X) = \phi$, $g_{T1}(Y) \neq \phi$, $C_{T1}(Y) \supset Y$ and $Y \subseteq X$).

Below we prove why we only need to perform closure checks for the itemsets specified in the above two conditions, and why we do not need to perform closure check in other conditions. We will use the Lemma 4.1 and Corollary 4.1 in subsequent proofs. The proof of Lemma 4.1 is given in (Lucchese, 2006); we use Lemma 4.1 in the proof of Corollary 4.1.

Lemma 4.1 Given an itemset X and an item $i \in I$, $g(X) \subseteq g(i) \Leftrightarrow i \in C(X)$.

Corollary 4.1 Assume $C_T(X)$ is X's closure within transaction set T. If $C_T(X) = X$ and if there exists a subset $Y \subset X$ such that $C_T(Y) \supset Y$ in transaction set T, then there exists an item i, where $i \in C_T(Y)$, $i \notin Y$, such that $i \in X$ and $C_T(Y) \subseteq X$.

Proof: Because $Y \subset X$, we have $g_T(X) \subseteq g_T(Y)$. If $i \in C_T(Y)$, from Lemma 4.1, we have $g_T(Y) \subseteq g_T(i)$. Therefore, we have $g_T(X) \subseteq g_T(i)$. Again from Lemma 4.1, we have $i \in C_T(X)$. So if $i \notin X$, we have $C_T(X) \neq X$, which is a contradiction with the given condition. Therefore, we have $i \in X$. Because $i \in C_T(Y)$, $i \notin Y$, $Y \subset X$, we have $C_T(Y) \subseteq X$. ↑

When a new transaction t in the data streams arrives, if t equals $\{X\}$, depends on whether X has or does not have a support larger than zero in the old transaction set there are two conditions. Below we discuss the update and maintenance rules under these two conditions. In the following proof, we assume X and Y are itemsets, *T1* is the old set of transactions, *T2* is the set of transactions after t arrives, $C_{T1}(X)$ is X's closure in transaction set *T1*, and $C_{T2}(Y)$ is Y's closure in the transaction set *T2*.

Case 1: When X has a support larger than zero in the old transaction set T1

For any new coming transaction t with the largest itemset X that already exists in the old transaction set *T1*, we have $g_{T1}(X) \neq \phi$. When $g_{T1}(X) \neq \phi$, for any itemset *Y and* $Y \subset X$, if $g_{T1}(Y) = \phi$. We have $Y \subset X \Rightarrow g_{T1}(Y) \supset g_{T1}(X) \neq \phi$. This is a contradiction with $g_{T1}(Y) = \phi$. Therefore, if $Y \subset X$, the condition $g_{T1}(Y) = \phi$ does not need to be discussed. If $Y \not\subset X \Rightarrow g_{T2}(Y) = g_{T1}(Y) = \phi$. Y's support is zero in *T2*. Thus, in both the cases $Y \subset X$ and $Y \not\subset X$, we do not need to discuss the case when $g_{T1}(Y) = \phi$. When $g_{T1}(X) \neq \phi$ and $g_{T1}(Y) \neq \phi$, we examine cases according to the following conditions: $Y \not\subset X$ and $Y \subseteq X$.

Case 1.A: When Y is a subset of X

When Y is a subset of X, $Y \subseteq X$, we divide it into two subconditions to analyze: X is or is not in the DIU.

Case 1.A.1: When X is in the old DIU

When X is in the old DIU, it is a closed itemset, therefore $C_{T1}(X) = X$. We have the following Lemma 4.2 and Lemma 4.3. From these two lemmas, we show

that if a closed itemset X, which already exists in the old DIU, arrives, for any itemset Y, $Y \subseteq X$, if Y is originally closed, it will remain closed; if Y is originally unclosed, Y will remain unclosed, and we only need to update Y's support. Therefore, for most of the existing closed itemsets, we do not need to update the DIU structure; we simply update their supports, which consume a small amount of time.

Lemma 4.2 Given $T2 = T1 \cup \{X\}$, if $C_{T1}(X) = X$ and $Y \subseteq X$ and $C_{T1}(Y) = Y$, then we have $C_{T2}(Y) = Y$.

In this lemma we prove that if both X and Y are closed itemsets in the old set of transactions $T1$, and $Y \subseteq X$, we have Y is also a closed itemset in the new transaction set $T2$.

Proof: Since $g_{T2}(Y) = g_{T1}(Y) \cup \{X\}$, we have $C_{T2}(Y) = f\bullet\, g_{T2}(Y) = f(g_{T1}(Y) \cup \{X\})$. Because $Y \subseteq X$, $f(g_{T1}(Y) \cup \{X\}) = f(g_{T1}(Y)) \cap f(\{X\}) = C_{T1}(Y) \cap X = Y \cap X = Y$. ↑

Lemma 4.3 Given $T2 = T1 \cup \{X\}$, if $C_{T1}(X) = X$ and $Y \subset X$ and $C_{T1}(Y) \supset Y$, then we have $C_{T2}(Y) \supset Y$.

In this lemma we prove that if X is a closed itemset in transaction set $T1$, and Y is not a closed itemset in transaction set $T1$, $Y \subset X$, we have Y is not a closed itemset in transaction $T2$.

Proof: $C_{T2}(Y) = f(g_{T2}(Y)) = f(g_{T1}(Y)) \cap f(\{X\}) = C_{T1}(Y) \cap \{X\}$. From Corollary 4.1, If $C_{T1}(X) = X$, $Y \subset X$, $C_{T1}(Y) \supset Y$. Given an item i, $i \in C_{T1}(Y)$, $i \notin Y$, we have $i \in X$. Therefore, $C_{T2}(Y) = C_{T1}(Y) \cap \{X\} \supseteq Y \cup \{i\} \supset Y$. ↑

From Lemma 4.2, we know that if Y is a closed itemset in transaction set $T1$ before X comes, and $Y \subseteq X$, Y will remain closed after X comes in transaction set $T2$. From Lemma 4.3, we know that if a closed itemset X which already exists on the DIU tree comes, its subset Y which originally unclosed will remain unclosed.

Case 1.A.2: When X is not in the old DIU

When X is not in the old DIU, it is not a closed itemset, therefore $C_{T1}(X) \supset X$. Similarly, we have the following Lemma 4.4 and Lemma 4.5. From Lemma 4.4, we show that if a new closed itemset, which is not originally in the old DIU, arrives and if its subsets are already in the DIU, they will remain closed, and thus we simply need to update their supports. From Lemma 4.5, we show that if a new closed itemset, which is not originally in the old DIU, arrives, then we need to insert it as a new closed itemset to the DIU.

Lemma 4.4 Given $T2 = T1 \cup \{X\}$, if $C_{T1}(X) \supset X$ and $Y \subset X$ and $C_{T1}(Y) = Y$, then we have $C_{T2}(Y) = Y$.

In this lemma, we prove that when X is not a closed itemset, if Y is closed itemsets in the old set of transactions $T1$, and $Y \subset X$, we have Y is also closed itemset in the new transaction set $T2$.

Proof: Since $g_{T2}(Y)=g_{T1}(Y) \cup \{X\}$, we have $C_{T2}(Y) = f \bullet g_{T2}(Y) = f(g_{T1}(Y) \cup \{X\})$. Because $Y \subset X$, $f(g_{T1}(Y) \cup \{X\}) = f(g_{T1}(Y)) \cap f(\{X\}) = C_{T1}(Y) \cap X = Y \cap X = Y$. ↑

From Lemma 4.5, we show that if a new closed itemset which is not originally in DIU arrives, we need to add itself as a new closed itemset in the DIU.

Lemma 4.5 Given $T2 = T1 \cup \{X\}$, if $C_{T1}(X) \supset X$ and $Y = X$, then we have $C_{T2}(Y) = Y = X$.

In this lemma, we prove that when X is not a closed itemset in the old transaction set $T1$, if $Y = X$, so Y is not a closed itemsets in the old set of transactions $T1$, we have Y is a closed itemset in the new transaction set $T2$.

Proof: $C_{T2}(Y) = f \bullet g_{T2}(Y) = f(g_{T1}(Y) \cup \{X\}) = f(g_{T1}(X)) \cap f(\{X\}) = C_{T1}(X) \cap f(\{X\}) = C_{T1}(X) \cap X = Y = X$. ↑

When $C_{T1}(X) \supset X$, $C_{T1}(Y) \supset Y$ and $Y \subset X$, we will perform the closure check to decide Y's closure, which will be discussed further in Section 4.2.2.2.

Case 1.B: When Y is not a subset of X

When Y is not a subset of X, $Y \not\subset X$, we have the following Lemma 4.6. In Lemma 4.6, we show that if Y is not a subset of X, Y's closure does not change. That is to say that if Y is an unclosed itemset before X arrives, then Y will remain unclosed after X arrives; and, if Y is a closed itemset before X arrives, then Y will remain closed after X arrives. Thus, the DIU structure does not need to be updated, and we only need to update Y's support.

Lemma 4.6 Given $T2 = T1 \cup \{X\}$, if $Y \not\subset X$, then we have $C_{T2}(Y) = C_{T1}(Y)$.

In this lemma we prove that when Y is not a subset of X, Y's closure doesn't change in transaction set $T2$.

Proof: If $Y \not\subset X$, $T2 = T1 \cup \{X\}$, we have $g_{T2}(Y) = g_{T1}(Y)$. Because $C_{T2}(Y) = f \bullet g_{T2}(Y)$, $C_{T1}(Y) = f \bullet g_{T1}(Y)$, $g_{T2}(Y) = g_{T1}(Y)$, we have $C_{T2}(Y) = C_{T1}(Y)$. ↑

Case 2: When X has a support equals to zero in the old transaction set T1

For any new coming transaction t with the largest itemset X that has not already appeared in the old transaction set *T1*, we have $g_{T1}(X) = \phi$. We discuss two sub cases according to the following conditions: $Y \not\subset X$ and $Y \subseteq X$.

Case 2.A: When Y is a subset of X

When Y is a subset of X, $Y \subseteq X$, we divide it into two subconditions to discuss: Y has a support greater than zero in the old transaction set *T1* or Y's support equals to zero in the old transaction set *T1*.

Case 2.A.1: When Y has a support greater than zero in the old transaction set T1

When Y is already in the old transaction set *T1*, then $g_{T1}(Y) \neq \phi$. Because $Y \subseteq X$, we have $g_{T2}(Y) = g_{T1}(Y) \cup \{X\}$. Therefore, $C_{T2}(Y) = C_{T1}(Y) \cap \{X\}$. If $C_{T1}(Y) = Y$, we have $C_{T2}(Y) = Y$ that means Y is also closed in *T2*. If $C_{T1}(Y) \supset Y$, we will perform the closure check to decide Y's closure, which will be discussed further in Section 4.2.2.2.

Case 2.A.2: When Y has a support equal to zero in the old transaction set T1

When Y does not have a support greater than zero in the old transaction set *T1*, then $g_{T1}(Y) = \phi$. We have the following Lemma 4.7. In this lemma, we prove that when Y is a subset of X, if $Y = X$, then Y is a closed itemset in the new transaction set *T2*; and, if $Y \subset X$, then Y is not a closed itemset in the new transaction set *T2*.

Lemma 4.7 Given $T2 = T1 \cup \{X\}$, if $Y = X$, then we have $C_{T2}(Y) = Y$; if $Y \subset X$, then we have $C_{T2}(Y) \supset Y$.

In this lemma we prove that when Y is a subset of X, if $Y = X$, Y is a closed itemset in transaction set *T2*; if $Y \subset X$, Y is not a closed itemset in transaction set *T2*.

Proof: If $Y = X$, then $g_{T2}(Y) = g_{T2}(X) = \{X\}$, from the given condition, we know $g_{T1}(X) = \phi$. Therefore after X arrives, we have support(Y) = support(X) =1. Because $g_{T1}(X) = \phi$, all X's supersets' supports = 0; from the definition of closed itemset, we have Y is a closed itemset after X arrives. If $Y \subset X$, then $g_{T2}(Y) = g_{T2}(X) = \{X\}$, from the given condition, we know $g_{T1}(X) = \phi$. Therefore we have support(Y) = support(X) = 1. Because X is a Y's superset, and they have the same support, we have Y as unclosed in transaction set *T2*. ↑

Case 2.B: When Y is not a subset of X

When Y is not a subset of X, $Y \not\subset X$, we divide it into two subconditions to discuss: Y has a support greater than zero in the old transaction set *T1* or Y's support equals to zero in the old transaction set *T1*.

Case 2.B.1: When Y has a support greater than zero in the old transaction set T1

If Y is already in the old transaction set *T1*, then $g_{T1}(Y) \neq \phi$. We have the following Lemma 4.8.

Lemma 4.8 Given $T2 = T1 \cup \{X\}$, if $Y \not\subset X$, then $C_{T2}(Y) = C_{T1}(Y)$.

In this lemma we prove that when Y is not a subset of X, Y's closure doesn't change in transaction set *T2*.

Proof: If $Y \not\subset X$, $Y \neq X$, we have $g_{T2}(Y)=g_{T1}(Y)$. Because $C_{T2}(Y) = f\bullet g_{T2}(Y)$, $C_{T1}(Y) = f\bullet g_{T1}(Y)$, $g_{T2}(Y) = g_{T1}(Y)$, we have $C_{T2}(Y) = C_{T1}(Y)$. ↑

Therefore, Y's closure doesn't change. That is to say if Y is an unclosed itemset before X comes, Y will remain unclosed after X comes; if Y is a closed itemset before X comes, Y will remain closed after X comes.

Case 2.B.2: When Y has a support equal to zero in the old transaction set T1

If Y is not in the old transaction set, then $g_{T1}(Y) = \phi$. If $Y \not\subset X$, we have $g_{T2}(Y) = g_{T1}(Y) = \phi$, which does not need to be discussed.

From the above proofs, we can see that when a new transaction arrives, for most of the above discussed cases, the DIU structure does not change and we only need to update the associated closed itemsets' supports in the DIU, which thus reduces the processing costs. There are only two cases out of thirteen total cases that we need to perform the closure check:

(1) Case 1.A.2: when $g_{T1}(X) \neq \phi$, $g_{T1}(Y) \neq \phi$, $C_{T1}(X) \supset X$, $C_{T1}(Y) \supset Y$ and $Y \subset X$; and

(2) Case 2.A.1: when $g_{T1}(X) = \phi$, $g_{T1}(Y) \neq \phi$, $C_{T1}(Y) \supset Y$ and $Y \subseteq X$.

We will discuss how to check for closed itemsets in the following Section 4.2.2.2.

4.2.2.2 Closure Check for Insertion

The CFI-Stream Algorithm checks whether an itemset is closed or not on the fly and incrementally updates the DIU based on the previous mining results with one scan of data streams. Below, we discuss the checking procedure when performing the insertion operation on the DIU. In the following Theorem 1, we show that for any

entering unclosed itemset Y, we can always find one and only one closed itemset X_c in the DIU that equals to Y's closure, i.e., $X_c = C(Y)$.

Theorem 4.1 For any itemset Y that satisfies with $C(Y) \supset Y$ and $g(Y) \neq \phi$, there exists one and only one closed itemset $X_c \in C$, where C is a set of existing closed itemsets, that satisfies with $C(Y) = X_c$, where $Y \subset X_c$.

Proof: To find X_c, we first find X_1, such that $X_1 \supset Y$, and support(X_1) = support(Y). According to the definition of closed itemsets, X_1 always exists. If X_1 is not closed, we can find X_2, where $X_2 \supset X_1$ and support(X_1) = support(X_2). Continuing this until we can find one X_c which is a closed itemset. This X_c is the itemset that satisfies $C(Y) = X_c$.

We also want to prove that there is only one such X_c, where support(X_c) = support(Y) in the existing closed itemsets. Assume there is another X_{c2}, where support(X_{c2}) = support(Y) in the existing closed itemsets. We know that for two different closed itemset X_c, and X_{c2}, $g(X_c) \neq g(X_{c2})$. Because $Y \subset X_c$ and $Y \subset X_{c2}$, we also know that $g(Y) \supseteq g(X_c)$ and $g(Y) \supseteq g(X_{c2})$. Therefore, $g(Y) \supseteq g(X_c) \cup g(X_{c2})$. The X_{c2} that we can find in the existing closed itemsets should satisfy with $g(Y) \supseteq g(X_c) \cup g(X_{c2})$, $g(Y) = g(X_c)$. From this we have $g(X_c) \supset g(X_{c2})$ because $g(X_c) \neq g(X_{c2})$, then this X_{c2} cannot have the same support as X_c. This conflicts with our assumption, support(X_c) = support(Y); so we could not find X_{c2}, thus X_c is unique.

We now prove $C(Y) = X_c$. For any $i \in C(Y)$, $i \notin Y$, from Lemma 4.1 we have $g(Y) \subseteq g(i)$. Because $Y \subset X_c$, we have $g(Y) \supseteq g(X_c)$. Therefore, we have $g(i) \supseteq g(X_c)$.

From Lemma 4.1, we have $i \in C(X_c) = X_c$, therefore we have $C(Y) \subseteq X_c$. For any $i \in X_c$, $i \notin Y$, because support(Y) = support(X_c), and from the given conditions we know $Y \subset X_c$, so we have $g(Y) = g(X_c)$. Also because $i \in X_c$, from Lemma 4.1, we have $g(i) \supseteq g(X_c) = g(Y)$. Therefore, we have $g(i) \supseteq g(Y)$. Again from Lemma 4.1 we know $i \in C(Y)$, thus we have $X_c \subseteq C(Y)$. From the above discussion, $C(Y) \subseteq X_c$ and $X_c \subseteq C(Y)$, we have $X_c = C(Y)$. ↑

From Theorem 4.1, we know that for any itemset Y that satisfies $C(Y) \supset Y$, we can find X_c with a minimum number of items in it and $X_c \supset Y$. For any other $X_{c1} \supset Y$, from the above discussion we know that $g(X_c) \supset g(X_{c1})$. Because $Y \subset X_c$, then $g(Y) \supseteq g(X_c) \supset g(X_{c1})$. To find $X_c = C(X_c) \subseteq C(Y)$, we have $g(X_c) = g(Y)$; only X_c will fulfill this requirement. In this way, $C(Y)$ can be found in the old transaction set *T1*. Below, we show how we use this $C(Y)$ to check if Y is a closed itemset in transaction set *T2* after X arrives.

Corollary 4.2 Given $T2 = T1 \cup \{X\}$, if $g_{T1}(Y) \neq \phi$, $Y \subseteq X$, $C_{T1}(Y) \supset Y$, $(C_{T1}(Y)/Y) \cap X = \phi$, then we have $C_{T2}(Y) = Y$.

Proof: $C_{T2}(Y) = f\bullet\, g_{T2}(Y) = f(g_{T1}(Y) \cup \{X\}) = f(g_{T1}(Y)) \cap f(\{X\}) = C_{T1}(Y) \cap f(\{X\}) = C_{T1}(Y) \cap X = Y$. ↑

From Corollary 4.2, we derive a way to check whether Y is closed in transaction *T2* or not. If $(C_{T1}(Y)/Y) \cap X = \phi$, then Y is a closed itemset in *T2*. We use this condition to perform the closed itemset check on the fly when a new transaction in the data streams arrives.

4.2.3 Delete a Transaction from the DIU

In this subsection, the update and maintenance of the DIU for the deletion operation, which occurs when a transaction leaves the sliding window is discussed. The result of the research is to define the conditions under which closed itemsets, currently stored in the DIU, need to be checked for closure when the old transaction leaves the current sliding window.

4.2.3.1 Conditions to Check for Closed Itemsets

First, we define and prove the following condition in which we need to check whether an itemset is closed or not when an old transaction X leaves the current sliding window.

Table 4-2 shows the conditions we classify to decide if a closure check is needed when perform the deletion operation.

Cases/Conditions			Closure Check
Case 1			No
Case 2	Case 2. A	Case 2.A.1	No
	Case 2. B	Case 2.B.1	Yes
		Case 2.B.2	No

Table 4-2: Conditions to check for deletion operation

From the above table, we can see that there is one condition we need to perform closure check, which is as in the following statement.

Condition1 (Case 2.B.1): When the number of the transactions with same itemset as X is equal to zero, for all subsets Y of X, where the number of transactions with same itemset as Y is equal to zero, and Y is a closed itemset in the old transaction set, we need to check whether Y remains closed or not (mathematically, when $\{X\} \notin T2$, $Y \subseteq X$, $\{Y\} \notin T2$, and $C_{T1}(Y) = Y$).

Below, we prove why we only need to perform closure check for closed itemsets specified in the above condition. In the following proof, we assume X and Y are itemsets, $T1$ is the old set of transactions, $T2$ is the new set of transactions after itemset X leaves, $C_{T1}(X)$ is X's closure within transaction set $T1$, and $C_{T2}(Y)$ is Y's closure under transaction set $T2$.

Case 1: When the number of the transactions with the same itemset X is greater than zero

When the number of transactions with the same itemset of X is greater than zero, we have the following Lemma 4.9. From this lemma, we know that Y's closure does not change when the number of transactions with the same itemset of X is greater than zero. That is to say that if Y is an unclosed itemset before X leaves, Y will remain unclosed after X leaves; and, if Y is a closed itemset before X leaves, Y will remain closed after X leaves.

Lemma 4.9 Given $T2 = T1 \setminus \{X\}$, $\{X\} \in T2$, we have $C_{T2}(Y) = C_{T1}(Y)$.

In this lemma we prove that when the number of the transactions with same itemset of X is greater than zero, Y's closure doesn't change in transaction set $T2$.

Proof: Because $\{X\} \in T2$, if $g_{T2}(X) \setminus \{X\} \neq \phi$, we have $f(g_{T2}(X)) = f(g_{T2}(X) \setminus \{X\}) \cap X$, so $C_{T2}(X) = f(g_{T2}(X) \setminus \{X\}) \cap X \subseteq X$. According to the definition, $C_{T2}(X) \supseteq X$. Therefore, we have $C_{T2}(X) = X$. If $g_{T2}(X) \setminus \{X\} = \phi$, we have $g_{T2}(X) = \{X\}$, $f(g_{T2}(X)) = f(\{X\})$, and $C_{T2}(X) = X$. Therefore, we have $C_{T2}(X) = X$.

(a) For $Y = X$, we have $C_{T2}(Y) = Y$, Y is a closed itemset in the transaction set $T2$.

(b) For $Y \subset X$, because $C_{T2}(X) = X$, $Y \subset X$, for $C_{T2}(Y) = Y$, we have $C_{T2}(Y) \subset X$; for $C_{T2}(Y) \supset Y$, from Corollary 4.1, we have $C_{T2}(Y) \subset X$. Therefore, $g_{T1}(Y) = g_{T2}(Y) \cup \{X\}$, so $C_{T1}(Y) = C_{T2}(Y) \cap \{X\}$. Because $C_{T2}(Y) \subset X$, $C_{T2}(Y) \cap \{X\} = C_{T2}(Y)$. Therefore, we have $C_{T2}(Y) = C_{T1}(Y)$.

(c) For $Y \not\subset X$, $Y \neq X$, we have $g_{T2}(Y)=g_{T1}(Y)$. Because $C_{T2}(Y) = f\bullet g_{T2}(Y)$, $C_{T1}(Y) = f\bullet g_{T1}(Y)$, $g_{T2}(Y) = g_{T1}(Y)$, we have $C_{T2}(Y) = C_{T1}(Y)$. ↑

Therefore, Y's closure doesn't change when the number of the transactions with same itemset of X is greater than zero. That is to say if Y was an unclosed itemset before X leaves, Y will remain unclosed after X leaves; if Y was a closed itemset before X leaves, Y will remain closed after X leaves.

Case 2: When the number of transactions with the same itemset X is equal to zero

When the number of the transactions with same itemset of X is equal to zero, $\{X\} \notin T2$, we divide this condition into the following two subconditions to discuss: Y is not a subset of X or Y is a subset of X.

Case 2.A: When Y is not a subset of X

If Y is not a subset of X, we have the following Lemma 4.10. In this lemma, we prove that when $\{X\}$ no longer exists in transaction set $T2$, and Y is not a subset of X, Y's closure does not change in transaction set $T2$.

Lemma 4.10 Given $T2 = T1 \setminus \{X\}$, if $\{X\} \notin T2$, $Y \not\subset X$, $Y \neq X$, then $C_{T2}(Y) = C_{T1}(Y)$.

In this lemma we prove that when $\{X\}$ is no longer exist in the transaction set $T2$, Y is not a subset of X, Y's closure doesn't change in transaction set $T2$.

Proof: If $\{X\} \notin T2$, $Y \not\subset X$, $Y \neq X$, we have $g_{T2}(Y) = g_{T1}(Y)$. Because $C_{T2}(Y) = f\bullet g_{T2}(Y)$, $C_{T1}(Y) = f\bullet g_{T1}(Y)$, $g_{T2}(Y) = g_{T1}(Y)$, we have $C_{T2}(Y) = C_{T1}(Y)$. ↑

Therefore, Y's closure doesn't change. That is to say if Y was an unclosed itemset before X leaves, Y will remain unclosed after X leaves; if Y was a closed itemset before X leaves, Y will remain closed after X leaves.

Case 2.B: When Y is a subset of X

If Y is a subset of X, we discuss according to the following subconditions: Y is a closed itemset in transaction set $T1$ and Y is not a closed itemset in transaction set $T1$.

Case 2.B.1: When Y is a closed itemset in transaction set T1

In the following Lemma 4.11, we prove that when Y is a subset of X, $Y \subset X$, $\{Y\} \in T2$. Y is a closed itemset in transaction set $T2$.

Lemma 4.11 For any itemset Y, if $Y \subset X$, $\{Y\} \in T2$, we have $C_{T2}(Y) = Y$.

In this lemma we prove that when Y is a subset of X, $Y \subset X$, $\{Y\} \in T2$. Y is a closed itemset in transaction set $T2$.

Proof: Because $g_{T2}(Y) = \{Y\} \cup (g_{T2}(Y) \setminus \{Y\})$, we have $C_{T2}(Y) = f(\{Y\}) \cap f(g_{T2}(Y) \setminus \{Y\}) \subseteq Y$. Also because $C_{T2}(Y) \supseteq Y$, we have $C_{T2}(Y) = Y$. ↑

From the above discussion, we can see that in the condition that we need to perform the closure check for the deletion operation, if $\{Y\} \in T2$, the Y is closed in the new transaction set $T2$. When Y is a closed itemset in the transaction set $T1$, that is to say when $Y \subseteq X$, $C_{T1}(Y) = Y$, and $\{Y\} \notin T2$, we need to perform the closure check, which we will discuss further in Section 4.2.3.2.

Case 2.B.2: When Y is not a closed itemset in transaction set T1

When Y is not a closed itemset in transaction set $T1$, we have the following Lemma 4.12.

Lemma 4.12 Given $T2 = T1 \setminus \{X\}$, if $Y \subset X$, $C_{T1}(Y) \subset Y$, then $C_{T2}(Y) \subset Y$.

In this lemma we prove that when Y is a subset of X, $Y \subset X$, and $C_{T1}(Y) \subset Y$, then Y is not a closed itemset in transaction set $T2$.

Proof: Because $Y \subset X$, $g_{T1}(Y) = g_{T2}(Y) \cup \{X\}$, $C_{T1}(Y) = f\bullet\, g_{T1}(Y) = f(g_{T2}(Y) \cup \{X\}) = C_{T2}(Y) \cap \{X\}$. Because $C_{T1}(Y) \supset Y$, $Y \subset X$, we have $C_{T2}(Y) \cap \{X\} \supset Y$. Therefore, $C_{T2}(Y) \supset Y$. ↑

From the above discussion, we can see that when an old transaction leaves the current sliding window, for most cases in the above discussions, the DIU structure does not change, and we need to update only the associated closed itemsets' supports, which thus reduces the update costs. There is only one case out of five total cases that we need to perform the closure check when an old transaction $\{X\}$ leaves the current sliding window: when $\{X\} \notin T2$, $Y \subseteq X$, and $\{Y\} \notin T2$, and $C_{T1}(Y) = Y$. We will discuss how to check for closed itemsets in the following section.

4.2.3.2 Closure Check for Deletion

The CFI-Stream Algorithm checks whether an itemset is closed or not on the fly, and incrementally updates the DIU based on the previous mining results with one scan of data streams. Below, we discuss the checking procedure for the deletion operation. In the following Theorem 4.2, we show that for any itemset Y, if $Y \subseteq X$, $C_{T1}(Y) = Y$, $\{X\} \notin T2$, then we can always find $C_{T2}(Y)$ in the original closed itemsets.

Theorem 4.2 For any itemset Y, if $Y \subseteq X$, $C_{T1}(Y) = Y$, $\{X\} \notin T2$, then $C_{T2}(Y) \in C_{T1}$. That is to say, we can always find $C_{T2}(Y)$ in C_{T1}.

Proof: $C_{T1}(C_{T2}(Y)) = f(g_{T1}(f(g_{T1}(Y) \setminus \{X\})))$

Because $\{X\} \notin T2$, there is one $\{X\}$ transaction in $T1$, we have $g_{T1}(Y) \setminus \{X\} \subseteq g_{T1}(f(g_{T1}(Y) \setminus \{X\})) \subseteq g_{T1}(Y)$. So we have either $g_{T1}(f(g_{T1}(Y) \setminus \{X\})) = g_{T1}(Y) \setminus \{X\}$ or $g_{T1}(f(g_{T1}(Y) \setminus \{X\})) = g_{T1}(Y)$.

In the first case, $g_{T1}(f(g_{T1}(Y) \setminus \{X\})) = g_{T1}(Y) \setminus \{X\}$. Because $C_{T1}(C_{T2}(Y)) = f(g_{T1}(f(g_{T1}(Y) \setminus \{X\}))) = f(g_{T1}(Y) \setminus \{X\}) = C_{T2}(Y)$, we have $C_{T2}(Y)$ as a closed itemset in C_{T1}.

In the second case, $g_{T1}(f(g_{T1}(Y) \setminus \{X\})) = g_{T1}(Y)$. Because $C_{T1}(C_{T2}(Y)) = f(g_{T1}(Y)) = C_{T1}(Y) = Y$. So we have $C_{T2}(Y) \subseteq Y$. Also because $Y \subseteq C_{T2}(Y)$, so we have $C_{T2}(Y) = Y$. So $C_{T2}(Y)$ is a closed itemset in C_{T1}.

Hence, for both cases $C_{T2}(Y) \in C_{T1}$, we definitely can find $C_{T2}(Y)$ in C_{T1}.

Below, we show how we perform the closure check when $\{Y\} \notin T2$ and to see if Y is a closed itemset in transaction set $T2$ after X leaves.

Corollary 4.3 If $Y \subseteq X$, $\{Y\} \notin T2$, for all $u_1, u_2, \ldots, u_i, \ldots, u_n$ which satisfies $C_{T2}(u_i) = u_i$, $Y \subset u_i$, we have $C_{T2}(Y) = u_1 \cap u_2 \cap \ldots \cap u_i \cap \ldots \cap u_n$.

Proof: First, we prove $C_{T2}(Y) \subseteq u_1 \cap u_2 \cap \ldots \cap u_i \cap \ldots \cap u_n$. Because $Y \subset u_i$, $C_{T2}(u_i) = u_i$ according to Corollary 4.1, $C_{T2}(Y) \subseteq u_i$. Therefore $C_{T2}(Y) \subseteq u_1 \cap u_2 \cap \ldots \cap u_i \cap \ldots \cap u_n$.

Next, we prove $C_{T2}(Y) \supseteq u_1 \cap u_2 \cap \ldots \cap u_i \cap \ldots \cap u_n$. For any transaction t, $t \in T2$, $Y \in t$. Because $\{Y\} \notin T2$, so we can find $Z \supset Y$, $Z \in t$. We know $C_{T2}(Z) \supseteq Z \supset Y$, $C_{T2}(Z) \in C_{T2}$. Because $u_1, u_2, \ldots, u_i, \ldots, u_n$ are all itemsets in C_{T2} which includes Y. So we can assume $C_{T2}(Z) = u_k$, so $g_{T2}(u_k) = g_{T2}(Z)$. So $t \in g_{T2}(Z)$, $t \in g_{T2}(u_k)$.

Therefore, we have $g_{T2}(Y) \subseteq g_{T2}(u_1) \cup g_{T2}(u_2) \cup \ldots \cup g_{T2}(u_i) \cup \ldots \cup g_{T2}(u_n)$. So $C_{T2}(Y) \supseteq C_{T2}(u_1) \cap C_{T2}(u_2) \cap \ldots \cap C_{T2}(u_i) \cap \ldots \cap C_{T2}(u_n) = u_1 \cap u_2 \cap \ldots \cap u_i \cap \ldots \cap u_n$.

Therefore, we have $C_{T2}(Y) = u_1 \cap u_2 \cap \ldots \cap u_i \cap \ldots \cap u_n$. ↑

From Corollary 4.3, we derive a way to check Y's closure: if $C_{T2}(Y) = u_1 \cap u_2 \cap \ldots \cap u_i \cap \ldots \cap u_n = Y$, then Y is a closed itemset. We use this rule to perform the closure check in the CFI-Stream Algorithm on the fly when an old transaction leaves the current sliding window.

4.3 The Proposed CFI-Stream Algorithm

Based on our above discussions, we derive an algorithm to perform online checking for closed itemsets over data streams. The CFI-Stream Algorithm performs an insertion operation when a new transaction arrives and a deletion operation when an old transaction leaves the current sliding window.

When a transaction arrives or leaves the current data stream sliding window, by performing the insertion and deletion operations, the CFI-Stream Algorithm checks each itemset in the transaction on the fly and updates the associated closed itemsets' supports. Current closed itemsets are maintained and updated in real time in the DIU. The closed frequent itemsets can be output at any time at users' specified thresholds by browsing the DIU.

4.3.1 The Insertion Procedure

The insertion procedure in Figure 4-2 illustrates the insertion process when an itemset *X* arrives. The algorithm first checks if *X* is in the current closed itemsets set *C*. If *X* is in *C*, it updates *X*'s support, and for all *X*'s subsets *Y* belonging to *C*, it updates *Y*'s supports (lines 3 to 8). Otherwise, if *X* is not in *C* and *X* has been included by at least one transaction in the old transaction set, it checks whether it is a closed itemset for itself and all its subsets after the new transaction arrives (lines 9 to 36); and, it updates the associated supports for all the closed itemsets (lines 37 to 40). If *X* is a newly arrived closed itemset and does not exist in the DIU, the algorithm inserts it as a new node to the DIU (lines 27 to 31). Otherwise, it inserts *X* into the closed itemset (lines 10-15); if *X* is the subset of the inserted transaction, a closure check is performed (lines 16-24). In the following algorithm description, *X* and *Y* represent itemsets, X_s and Y_s represent *X*'s support and *Y*'s support, len(*X*) represents the length of the itemset *X*, which is the number of items in an itemset *X*, *C* represents the original closed itemsets in the DIU, and C_{new} represents new closed itemsets in the DIU after itemset *X* arrives.

CFI-Stream – Insertion

```
1   X_close = true; C_new = ϕ;
2   procedure Insert(X, C, C_new)
3      if (X ∈ C)
4         for all (Y ⊆ X and Y ∈C)
5            Y_s ← support(Y, C) + 1;
6         end for
7         if (X_close = true)  return;
8      else
```

```
9         if (support(X, C) > 0 )
10              if(C_new = ϕ )
11                  X_0 ← X;
12                  C_new ← X;
13                  X_close = false;
14                  X_s ← support (X, C) + 1;
15              else
16                  X_c = ϕ;
17                  M = I;
18                  for all ( K ⊃ X and K ∈ C)
19                    if (len(K)<len(M)) M=K;
20                  end for
21                  X_c ← M;
22                  if ((X_c\X) ∩ X_0 = ϕ and X_c ≠ ϕ )
23                     C_new ← C_new ∪ X;
24                     X_s ← support(X, C) + 1;
25                  end if
26              end if
27        else
28              if (C_new = ϕ )
29                 X_0 ← X;
30                 C_new ← X;
31                  X_s = 1;
32              end if
33        end if
34     end if
35     for all (m ⊂ X and Len(m) = Len(X)-1)
36               call Insert(m, C, C_new);
37     end for
38     if (X = X_0)
39        C ← C ∪ C_new;
40        support(X, C) = X_s;
41     end if
42 end procedure
```

Figure 4-2: CFI-Stream algorithm – insertion

4.3.2 The Deletion Procedure

The deletion procedure in Figure 4-3 illustrates the procedure to perform the deletion operation when an itemset X leaves the current sliding window. CFI-Stream

first checks if X is in the current closed itemsets set C and its count is greater or equal to two; if so, it updates X's support and X's subsets' support belonging to C (lines 3 to 6). Otherwise, it checks the itemset X and all its subsets, which are in the current closed itemset set C, to see whether they are still closed itemsets (lines 8 to 26) and updates the support for all its subsets that are in the current closed itemsets (lines 28 to 29). If the subset Y exists in the transaction, Y should keep closed (lines 11-13); otherwise a closure check for the subset Y is performed (lines 14-22). In the following Figure 4-3, $C_{obsolete}$ represents the itemsets that are no longer closed after transaction $\{X\}$ leaves.

CFI-Stream – Deletion

```
1   C_obsolete = ϕ;
2   procedure Delete (X, C, C_obsolete)
3     if (count({X}) ≥ 2)
4       for all (Y ⊆ X and Y ∈C)
5         Y_s ← support(Y, C) – 1;
6       end for
7     else
8       length = len(X);
9       while (length ≥ 1)
10        for all ( Y ⊆ X and Y ∈C and len(Y) = length)
11          if (count({Y}) ≥ 2)
12            Y_s ← support(Y, C) – 1;
13          else
14            M = I;
15            for all ( U ⊃ Y and U ∈C)
16              M = M ∩ U;
17            end for
18            if (M = Y)
19              Y_s ← support(Y, C) – 1;
20            else
21              C_obsolete= C_obsolete ∪ Y;
22            end if
```

```
23                  end if
24              end for
25              length = length-1;
26          end while
27      end if
28      C ← C \ C_obsolete
29      support(Y, C) = Y_s;
30  end procedure
```

Figure 4-3: CFI-Stream algorithm – deletion

4.4 Comparing with Existing Literature

Table 4-3 summarizes the recent closed pattern mining approaches. From which we can see that according to different mining strategies, the proposed methods perform single or multiple scan through the entire dataset. In the data stream environment, as we discussed in Section 1.2, the single scan of data and compact memory usage of the mining technique are preferable. Chi et al proposed the Moment Algorithm to judge the closed itemsets indirectly through node property checking and excludes them from the other three types of boundary nodes stored in the data structure. And in (Li, 2006), the authors proposed the NewMoment Algorithm which uses a bit-sequence representation of items to reduce the time and memory needed. We proposed the CFI-Stream Algorithm in (Jiang, 2006) to directly compute the closed itemses online and incrementally without the help of any support information. In (Li, 2008), Li et al proposed to improve the CFI-Stream Algorithm with bitmap coding named CLIMB (Closed Itemset Mining with Bitmap) over data stream's sliding window to reduce the memory cost.

			Mining Strategy	Mining Process	Data Stream Support	Scan
Mining Closed Patterns	Static Data	Pasquier 99	Key Pattern Browsing	Offline	No	Multiple
		Pei 00, Zaki 02, Pei 03	Closure Climbing	Online	No	Multiple
	Stream Data	Chi 04, Li 06	Indirect	Online	Yes	Single
		Proposed 06, Li 08	Direct	Online	Yes	Single

Table 4-3: Recent closed pattern mining approaches

4.5 Summary

In this chapter an algorithm called CFI-Stream is proposed to directly compute closed itemsets online and incrementally, without requiring the user to provide support information. Once the closed itemsets are determined, the user's support information can be used to easily retrieve the desired frequent itemsets.

An in-memory data structure DIU is proposed to store and monitor the closed patterns in the current sliding window. Nothing other than closed itemsets and their support is maintained in the DIU. The proposed CFI-Stream Algorithm is a sliding window approach to maintain the DIU in an incremental fashion. When a new transaction arrives, it performs the closure check on the fly; only associated closed itemsets and their support information are incrementally updated. This achieves both time and space efficiency compared with the state of the art algorithm for closed pattern mining in data streams (Chi, 2004). The current closed itemsets can be output in real time based on any user's specified support thresholds.

5 Association Mining in Data Streams based on Closed Pattern Mining

Association mining can produce many association rules. It is widely recognized that the set of association rules can rapidly grow to be unwieldy, especially when the support requirements are relatively low. In general, mining a large set of frequent itemsets leads to a large number of rules being presented to the user, many of which are redundant and difficult to analyze. A primary goal of the proposed approach is to reduce the number and redundancy of the rules provided to the user.

Many researchers have considered various kinds of solutions to the above problem, and these can be divided into the following three categories: First is efficient association mining based on frequent itemsets. This category's research objective is to enumerate all frequent itemsets, and to produce association rules based on the derived frequent itemsets. Second is mining interesting association rules. This category's research objective is to incorporate user-specified constraints on the kind of rules generated or to define objective metrics of interest. Third is mining non-redundant association rules. This category's research objectives include the generation of non-redundant association rules.

In this research we focus on the combination of the second and third approaches, to mine non-redundant and informative association rules that match the

user interests. The generated association rules are evaluated by users for data analysis. Because the cost of evaluating a large number of rules can be very high, we attempt to reduce the non-informative association rules by generating only non-redundant association rules that match the user's interests.

5.1 Overview

The goal of association rule mining is to discover interesting associations and correlation relationships among a large set of items. With massive amounts of data continuously arriving in a data stream environment, it is possible for a huge number of rules to be generated continuously.

Although there are a lot of existing studies on association rule mining, traditional association rule mining techniques are not suitable for a data stream environment due to several reasons. These reasons are outlined in detail in Section 2.2.1. Different from the previous non-redundant association rule generation techniques that have been studied for the traditional database (Bastide, 2000, Zaki 2005), the proposed technique is to generate association rules with a single scan based on the closed pattern mining method we proposed in Chapter 4. The rule generation is based on the current closed itemsets in data streams derived from the DIU, which are a condensed representation of the whole stream data without loss of information. Compared with (Li 2004), the proposed technique involves the mining of minimal non-redundant association rules from the DIU based on a sliding window model, instead of a quantitative closed itemset lattice based on a landmark data

processing model as we discussed in Section 2.1.3. Thus both insertion and deletion operation can be performed on the data streams. Furthermore, the DIU contains all the closed patterns in the current sliding window. Therefore, the rules can be generated on demand, at different user-specified support and confidence thresholds.

Theoretical analysis and experimental results are also performed to show that our proposed technique can efficiently produce non-redundant rules in data streams, which provide a condensed set of association rules among itemsets in data streams and make it easier for data analysis. In addition, only correlated relationships are developed and user interest patterns are output from the pattern filters. The rules can be generated for multiple user query requests with different thresholds and pattern requirements which are especially suitable for the distributed data stream query environment.

5.2 The Proposed Rule Mining Framework based on Closed Pattern Mining

In this section, we present an online non-redundant and informative association rule mining framework based on the closed pattern mining method in data streams we proposed in Chapter 4. We first briefly describe the framework we are going to use to compute the closed frequent itemsets and mine non-redundant association rules in data streams. Then we discuss how we mine non-redundant and informative association rules based on the discovered closed patterns.

As illustrated in Figure 5-1, when data stream comes and leaves the server, the CFI-Stream Algorithm checks each itemset in the transaction on the fly and updates the associated closed itemsets' supports. Current closed itemsets are maintained and their support values are updated in real time in the DIU. We mine the minimal non-redundant association rules based on the closed patterns maintained in the lexicographical ordered direct update lattice. The derived rule set then goes through the correlation filter to leave out any non-correlated associations into user consideration. Based on different users' requests on interested patterns, minimum support and confidence thresholds, different association rule sets are output through the pattern filter.

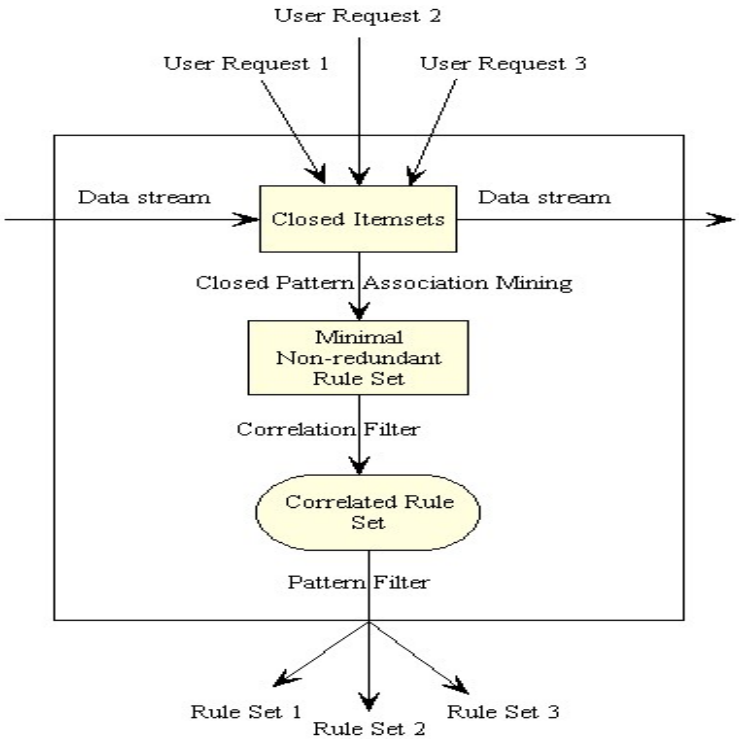

Figure 5-1: The proposed association mining framework based on closed pattern mining

5.3 Mining Informative Associations based on Closed Pattern Mining

It is widely recognized that the set of association rules can rapidly grow to be unwieldy, especially when support requirements are low. In this section, we show how frequent closed itemsets can help us form a basic set of rules, from which all other association rules can be inferred. Thus only a small and understandable set of rules need to be presented to the user that can later selectively derive other rules of interest. We show that the derived association rules in data streams are non-redundant rules that provide a minimum set of association rules among itemsets in data streams and make it easier for data analysis.

Lemma 5.1 The support of an itemset X is equal to the support of its closure, i.e. $s(X) = s(C(X))$.

This lemma, reported in (Pasquier, 1999) and (Zaki, 2000), states that all frequent itemsets are uniquely determined by the frequent closed itemsets. From Lemma 5.1, we know that the support of an itemset X equals the support of its closure $C(X)$. Thus it suffices to consider rules only among the frequent closed itemsets (Zaki, 2000). We show that they are equivalent in the following Lemma 5.2. In the following proofs, we use $|g(X)|$ to represent the number of transactions in $g(X)$.

Lemma 5.2 The rule $X_1 \xrightarrow{s,c} X_2$ is equivalent to the rule $C(X_1) \xrightarrow{s,c} C(X_2)$.

Proof: For support we have $s = s(X_1 \cup X_2) = |g(X_1 \cup X_2)| = |g(X_1) \cap g(X_2)|$. By Lemma 5.1, we have $s = |g(C(X_1)) \cap g(C(X_2))|$, because the support of an itemset and its closure is the same. The last expression can be rewritten as $s = |g(C(X_1) \cup C(X_2))| = s(C(X_1) \cup C(X_2))$. For confidence, we have $c = s/|g(X_1)| = s/|g(C(X_1))|$. Therefore, the rule $X_1 \xrightarrow{s,c} X_2$ is equivalent to the rule $C(X_1) \xrightarrow{s,c} C(X_2)$.

Lemma 5.3 (Zaki, 2000) The rule $X_1 \xrightarrow{s,c} X_2$ is equivalent to the rule $X_1 \xrightarrow{s1,c1} X_1 \cup X_2$, i.e., $s = s1$ and $c = c1$.

Proof: For support we have $s = |g(X_1 \cup X_2)| = |g(X_1 \cup (X_1 \cup X_2))| = s1$. For confidence, we have $c = s/|g(X_1)| = s1/|g(X_1)| = c1$. Therefore, the rule $X_1 \xrightarrow{s,c} X_2$ is equivalent to the rule $X_1 \xrightarrow{s1,c1} X_1 \cup X_2$, i.e., $s = s1$ and $c = c1$.

In the following discussions, we consider two cases of association rules, those with 100% confidence, i.e. with $c = 1.0$, and those with $c < 1.0$.

Case 1: Rule with confidence = 100%

Lemma 5.4 The rule $X_1 \xrightarrow{s,c=1.0} X_1 \cup X_2$ is equivalent to the rule $X_1 \xrightarrow{s1,c1=1.0} C(X_1 \cup X_2)$, i.e., $s = s1$ and $c = c1$.

Proof:

(a) Because $X_1 \xrightarrow{s,c=1.0} X_1 \cup X_2$, we have $c = |g(X_1 \cup (X_1 \cup X_2))|/|g(X_1)| = 1.0$. Therefore, we have $|g(X_1 \cup X_2)|/|g(X_1)|=1.0$.

(b) Now let's look at the rule $X_1 \xrightarrow{s1,c1=1.0} C(X_1 \cup X_2)$. For the confidence, $c1 = |g(X_1 \cup C(X_1 \cup X_2))|/|g(X_1)| = |g(C(X_1 \cup X_2))|/|g(X_1)|$. From Lemma 5.1, $s(C(X_1 \cup X_2)) = s(X_1 \cup X_2)$, we have $|g(X_1 \cup X_2)|= |g(C(X_1 \cup$

$X_2))|$. Therefore, $c1 = |g(X_1 \cup C(X_1 \cup X_2))|/|g(X_1)| = |g(C(X_1 \cup X_2))|/|g(X_1)| = |g(X_1 \cup X_2)|/|g(X_1)|=1.0 = c$. For the support, $s1 = |g(C(X_1 \cup X_2))| = |g(X_1 \cup X_2)| = s$. Therefore, the rule $X_1 \xrightarrow{s,c=1.0} X_1 \cup X_2$ is equivalent to the rule $X_1 \xrightarrow{s,c=1.0} C(X_1 \cup X_2)$.

Lemma 5.5 The rule $X_1 \xrightarrow{s,c=1.0} X_2$ is equivalent to the rule $X_1 \xrightarrow{s,c=1.0} C(X_1)$, and also the rule $X_1 \xrightarrow{s,c=1.0} X_2$ is redundant.

Proof:

(a) From Lemma 5.3, we have $X_1 \xrightarrow{s,c} X_2$ is equivalent to the rule $X_1 \xrightarrow{s1,c1} X_1 \cup X_2$. From (Luxenburger, 1991), we know that an association rule $X_1 \xrightarrow{s,c} X_2$ has confidence $c = 1.0$ if and only if $g(X_1) \subseteq g(X_2)$, or equivalently if and only if $C(X_2) \subseteq C(X_1)$. Therefore, we have rule $X_1 \xrightarrow{s,c} X_1 \cup X_2$ has confidence $c = 1.0$ if and only if $g(X_1) \subseteq g(X_1 \cup X_2)$, or equivalently if and only if $C(X_1 \cup X_2) \subseteq C(X_1)$.

(b) Because $X_1 \subseteq X_1 \cup X_2$, from the monotonicity property of Galois connection (Luxenburger, 1991), we have $C(X_1) \subseteq C(X_1 \cup X_2)$.

(c) From (a) and (b) we know that $C(X_1) = C(X_1 \cup X_2)$, also from Lemma 5.3, we have $X_1 \xrightarrow{s,c} X_2$ is equivalent to the rule $X_1 \xrightarrow{s,c} X_1 \cup X_2$. From Lemma 5.4, we have rule $X_1 \xrightarrow{s,c=1.0} X_1 \cup X_2$ is equivalent to the rule $X_1 \xrightarrow{s,c=1.0} C(X_1 \cup X_2)$. Therefore, we have the rule $X_1 \xrightarrow{s,c=1.0} X_2$ is equivalent to the rule $X_1 \xrightarrow{s,c=1.0} C(X_1 \cup X_2)$. Also because $C(X_1) = C(X_1 \cup$

X_2), we have the rule $X_1 \xrightarrow{s,c=1.0} X_2$ is equivalent to the rule $X_1 \xrightarrow{s,c=1.0} C(X_1)$.

From Lemma 5.5, we proved that any association rule $X_1 \xrightarrow{s,c=1.0} X_2$ is equivalent to the rule $X_1 \xrightarrow{s,c=1.0} C(X_1)$. Therefore, the set of association rules in the format $X_1 \xrightarrow{s,c=1.0} C(X_1)$ is complete. Because $C(X_1) = C(X_1 \cup X_2)$, and from the extension property of Galois connection $X \subseteq C(X)$, we have $X_2 \subseteq C(X_2) \subseteq C(X_1)$. Thus, from the rule redundancy definition in Chapter 3, the rule $X_1 \xrightarrow{s,c=1.0} X_2$ is redundant.

In the following Lemma 5.6, we prove that the rules from all non-minimum generators to its closure are redundant.

Lemma 5.6 The rules $X_1 \xrightarrow{s,c=1.0} C(X_1)$, X_1 is not a minimum generator, are redundant.

Proof: From the minimum generator definition in Chapter 3, we know that the smallest itemset X_1 that satisfies with $C(X_1) = X_2$, is called X_2's minimum generator. If X_1 is not the minimum generator, we can find a minimum generator X_1', such that $X_1' \xrightarrow{s,c=1.0} C(X_1)$. $s' = s(X_1' \cup C(X_1)) = |g(X_1' \cup C(X_1))| = |g(X_1') \cap g(C(X_1))| = |g(C(X_1))| = |g(X_1) \cup g(C(X_1))| = s$, and $c' = |g(X_1' \cup C(X_1))| / |g(X_1')| = |g(X_1 \cup C(X_1))| / |g(X_1)| = c$. Therefore, the rules $X_1 \xrightarrow{s,c=1.0} C(X_1)$ are equivalent with the rules $X_1' \xrightarrow{s,c=1.0} C(X_1)$. Also because $X_1' \subseteq X_1$, the rules from non-minimum generator $X_1 \xrightarrow{s,c=1.0} C(X_1)$ are redundant.

We use the method in (Li, 2004) to find the minimum generator of a given closed itemset. In the following Lemma 5.7, we prove that all association rules from the minimum generator to closed itemsets are non-redundant.

Lemma 5.7 The rules $X_1 \xrightarrow{s,c=1.0} C(X_1)$, X_1 is a minimum generator, are non-redundant.

Proof: Assume that we have two rules $X_1 \xrightarrow{s,c=1.0} C(X_1)$, and $X_2 \xrightarrow{s,c=1.0} C(X_2)$. If $X_1 \supset X_2$, and $C(X_1) \subset C(X_2)$, then $X_1 \rightarrow C(X_1)$ is redundant. We show this is impossible. Because X_1 and X_2 are minimum generators, if $X_1 \supset X_2$, from the monotonicity of Galois connection property (Luxenburger, 1991), we have $C(X_1) \supset C(X_2)$. This is contrary with the given assumption $C(X_1) \subset C(X_2)$. Therefore the rules $X_1 \rightarrow C(X_1)$ are non-redundant.

From the above discussions, we show that when confidence of the association rule is equal to 1, the set of association rules in the format $X_1 \xrightarrow{s,c=1.0} C(X_1)$, X_1 is a minimum generator, is complete and non-redundant. In the following, we discuss the conditions when the confidence of the rule is less than 1.

Case 2: Rule with confidence < 100%

Lemma 5.8 The rule $X_1 \xrightarrow{s,c<1.0} X_2$ is equivalent to the rule $X_1 \xrightarrow{s,c<1.0} C(X_1 \cup X_2)$, and $X_1 \subset C(X_1 \cup X_2)$.

Proof: For the rule $X_1 \xrightarrow{s,c<1.0} X_2$, the support $s = s(X_1 \cup X_2)) = |g(X_1 \cup X_2)|$, the confidence $c = |g(X_1 \cup X_2)| / |g(X_1)|$. For the rule $X_1 \xrightarrow{s,c<1.0} C(X_1 \cup X_2)$, we have its support is $s(X_1 \cup C(X_1 \cup X_2)) = |g(X_1 \cup C(X_1 \cup X_2))| = | g(X_1) \cap g(C(X_1 \cup X_2))| = |$

$g(X_1) \cap g(X_1 \cup X_2)| = | g(X_1) \cap g(X_1) \cap g(X_2)| = | g(X_1) \cap g(X_2)| = |g(X_1 \cup X_2)| = s$, the confidence is $|g(X_1 \cup C(X_1 \cup X_2))|/ |g(X_1)| = |g(X_1 \cup X_2)| / |g(X_1)| = c$. In all above association rules, c is less than 1, the support of X_1 is greater than the support of $C(X_1 \cup X_2)$, therefore we have $X_1 \subset C(X_1 \cup X_2)$. From the above discussion, we proved that any association rule $X_1 \xrightarrow{s,c<1.0} X_2$ is equivalent to the rule $X_1 \xrightarrow{s,c<1.0} C(X_1 \cup X_2)$, and $X_1 \subset C(X_1 \cup X_2)$. Therefore, if we can find all association rules with the format of $X_1 \xrightarrow{s,c<1.0} C(X_1 \cup X_2)$, $X_1 \subset C(X_1 \cup X_2)$, these association rules should provide complete information.

Lemma 5.9 The rules $X_1 \xrightarrow{s,c<1.0} C(X_1 \cup X_2)$, $X_1 \subset C(X_1 \cup X_2)$, X_1 is not a minimum generator, are redundant.

Proof: From the definition of minimum generator in Chapter 3, we know that the most minimal generator X_1 is the itemset that satisfies with $C(X_1) = X_2$. If X_1 is not the minimum generator, we can find a minimum generator X_1', such that $X_1' \xrightarrow{s,c<1.0} C(X_1 \cup X_2)$. $s' = |g(X_1' \cup C(X_1 \cup X_2))| = |g(X_1') \cap g(C(X_1 \cup X_2))| = |g(X_1') \cap g(X_1) \cap g(X_2)| = |g(X_1) \cap g(X_2)| = s$, and $c' = |g(X_1') \cup C(X_1 \cup X_2)|/|g(X_1)| = |g(X_1') \cap g(X_1) \cap g(X_2)| /|g(X_1)| = |g(X_1 \cup X_2)|/ |g(X_1)| = c$. Also because $X_1' \subseteq X_1$, the rules from non-minimum generator $X_1 \xrightarrow{s,c<1.0} C(X_1 \cup X_2)$, $X_1 \subset C(X_1 \cup X_2)$, are redundant.

In the following Lemma 5.10, we prove that all association rules from the minimum generators to their closed supersets are non-redundant.

Lemma 5.10 The rules $X_1 \xrightarrow{s,c<1.0} C(X_1 \cup X_2)$, $X_1 \subset C(X_1 \cup X_2)$, X_1 is a minimum generator, are non-redundant.

Proof: Assume that we have two rules $X_1 \xrightarrow{s,c<1.0} C(X_1 \cup X_2)$, and $X_1' \xrightarrow{s,c<1.0} C(X_1 \cup X_2)$, $X_1 \subset C(X_1 \cup X_2)$. If $X_1 \supset X_1'$, then $X_1 \rightarrow C(X_2)$ is redundant. We show this is impossible. Because X_1 and X_1' are generators, if $X_1 \supset X_1'$, from the monotonicity property of Galois connection, we have $C(X_1) \supset C(X_1')$. This is contrary with the given condition that both X_1 and X_1' are generators of the same close itemset, i.e. $C(X_1) = C(X_1')$. Therefore the rules $X_1 \rightarrow C(X_1 \cup X_2)$, $X_1 \subset C(X_1 \cup X_2)$, X_1 is minimum generator are non-redundant.

From the above discussions, we show that when confidence of the association rule is less than 1, the set of association rules in the format $X_1 \xrightarrow{s,c<1.0} C(X_1 \cup X_2)$, $X_1 \subset C(X_1 \cup X_2)$ and X_1 is minimum generator, is a complete and non-redundant association rule. In this relationship, $C(X_1 \cup X_2)$ is X_1's closed supersets. From (Zaki 2005), we know that for closed itemsets related by the subset relation, it is sufficient to consider rules among adjacent closed itemsets, since other rules can be inferred by transitivity (Luxenburger, 1991). Therefore, in our proposed algorithm, we derive rules only among immediate parent and child nodes.

Not all association rules are correlated with each other, to determine all the correlated association rules, we introduce the lift formula to calculate the correlation of two closed patterns. The lift of two closed patterns X and Y can be measured as $\text{lift}(X, Y) = s(X \cup Y) / s(X)s(Y) = |g(X \cup Y)| / |g(X)||g(Y)|$. As discussed in (Han, 2001),

if the resulting value is less than 1, then the occurrence of X is negatively correlated with the occurrence of Y. Otherwise if the resulting value is greater than 1, then X and Y are positively correlated. If the resulting value is equal to 1, then X and Y are independent and there is no correlation between them. We calculate and output all the positively correlated rules, having lift values greater than 1.

Furthermore, different users often have different query requests at the same time to the same stream of data. This is due to the fact that each user may have different needs and interest information. To match different users' query requests at the server, we derive mechanism to output only the rule sets that match different user-specified support and confidence thresholds. We also include a pattern filter in the proposed association rule mining framework, which outputs the particular patterns that the user interests about. For example, the electronic department manager of a wholesale store may particular interests about the rule sets that imply the following information: if a customer buys a camera, what other products that he or she may also want to buy? In this specific query, camera is the user interest input pattern. Based on this information, we derive the rule sets that match the input and output patterns specified by different users. Figure 5-2 shows how we mine the non-redundant and informative association rules in data streams from the DIU.

Input: (1) DIU: All closed itemsets in the DIU
(2) $S_{specify}$: the user-specified minimum support
(3) $C_{specify}$: the user-specified minimum confidence
(4) P_{in}: the user-specified input pattern
(5) P_{out}: the user-specified output pattern

Output: R: The output informative association rule set

Method:

```
1 for each node X in the DIU
2    if (S(X) ≥ S_specify)
3          find X's minimum generator Y
4          for each Y, and P_in ⊆ Y
5                if (Y ≠ X and P_out ⊆ X and lift(Y, X) > 1)
6                     R = R ∪ (Y → X)
7                     S = S(X)
8                     C = 1
9                     for each X's immediate upper-level node X_p
10                         if (S(X_p) ≥ S_specify and S(X_p)/S(X) ≥ C_specify
                               and P_out ⊆ X_p and lift(Y, X_p) > 1)
11                             R = R ∪ Y → X_p
12                             S = S(X_p)
13                             C = S(X_p)/S(X)
14                        end if
15                    end for
16               end if
17               if (Y = X and P_out ⊆ X and lift(Y, X) > 1)
18                    for each X's immediate upper-level node X_p
19                         if (S(X_p) ≥ S_specify and S(X_p)/S(X) ≥ C_specify
                               and P_out ⊆ X_p and lift(Y, X_p) > 1)
20                             R = R ∪ Y → X_p
21                             S = S(X_p)
22                             C = S(X_p)/S(X)
23                        end if
24                   end for
25              end if
26         end for
27    end if
28 end for
```

Figure 5-2: The informative association mining algorithm

5.4 Comparing with Existing Literature

Table 5-1 summarizes recent association rule mining approaches. The mining algorithms can be categorized based on the mining processes, the number of itemsets the association rule mines, the number of scans the algorithm performs, etc. Traditional rule mining algorithms based on frequent and closed patterns are performed offline and need multiple scans over the entire dataset. In (Yang, 2004), (Halatchev, 2005), and (Shin, 2007), the authors proposed using two, three, and multiple frequent pattern based methods to perform association rule mining. Instead of using frequent pattern mining, we proposed to perform association rule mining based on closed pattern mining technique we discussed in Chapter 4, which is a multiple closed pattern mining based algorithm, and be able to answer multiple requests from different users' specified interest query criteria at the same time.

				Number of Itemsets	Mining Process	Data Stream Support	Scan
Mining Association Rule	Static Data	Frequent Itemsets	Agrawal 93, Agrawal 94, Liu 99, Han 00	Multiple	Offline	No	Multiple
		Closed Itemsets	Bastide 00, Li 04, Zaki 05	Multiple	Offline	No	Multiple
	Stream Data	Frequent Itemsets	Yang 04, Halatchev 05, Shin 07	Two/Three/ Multiple	Online	Yes	Single
		Closed Itemsets	Proposed 07	Multiple	Online	Yes	Single

Table 5-1: Recent association mining approaches

5.5 Summary

In this chapter we propose a framework to produce non-redundant and informative association rules based on closed itemset mining in data streams. Based on the discovered closed itemsets derived and maintained in DIU, we perform non-redundant association and informative rule mining using an association mining framework. Theoretical analysis and experimental results show that our proposed technique can efficiently produce non-redundant rules in data streams that provide a minimum set of association rules among itemsets in data streams and thus make it easier for data analysis. Furthermore, the rules can be generated on demand, at different users' request thresholds, and different input and output patterns.

6 Missing Data Estimation in a Sensor Network Database Based on Closed Pattern Association Mining

In this chapter, a data estimation technique is developed based on association rules derived from closed frequent patterns generated by sensors, to discover relationships between sensors and use them to perform missing data estimation. By discovering the relationships between multiple sensors when they have the same or different values, this technique can perform data estimation for more cases than the state of the art technique (Halatchev, 2005) and improve the estimation accuracy.

6.1 Overview

Recent advances in sensor technology have made possible the development of relatively low cost and low-energy-consumption micro sensors which can be integrated in a wireless sensor network. These devices - Wireless Integrated Network Sensors (WINS) - will enable fundamental changes in applications spanning the home, office, clinic, factory, vehicle, metropolitan area, and the global environment (Asada, 1998).

Many research projects have been conducted by different organizations regarding wireless sensor networks; however, few of them discuss how to estimate the missing data when data is lost or corrupted. Traditional methods to handle the situation when data is missing are to ignore the missing data, make sensors send them again or use some statistical methods to perform the estimation. As we

discussed in Chapter 2.3, these methods are not especially suited for wireless sensor networks.

In this chapter, a data estimation technique is developed using Closed Association Rule Mining (CARM) on stream data to discover relationships between sensors and use them to compensate for missing and corrupted data. Different from other existing techniques (Dempster 1977, Gelman 1995, Halatchev 2005, McLachlan 1997, Rubin 1996), CARM can find out the relationships between two or more sensors when they have the same or different values. The derived association rules provide complete and non-redundant information; therefore it can improve the estimation accuracy and achieve both time and space efficiency. Furthermore, CARM is an online and incremental algorithm, which is especially beneficial when users have different specified support thresholds in their online queries.

6.2 The Data Structure and Online Closed Pattern Association Mining in Data Streams

In this section, an online data estimation technique called CARM is developed based on the closed frequent pattern mining algorithm we proposed. When a transaction arrives or leaves the current data stream sliding window, the proposed closed pattern mining algorithm checks each itemset in the transaction on the fly and updates the associated closed itemsets' supports. The current closed itemsets are maintained and updated in real time in the DIU, and can be output at any time at users' specified thresholds by browsing the DIU.

A lexicographical ordered direct update lattice is used to maintain the current closed itemsets. Each node in the DIU represents a closed itemset. There are *k* levels in the DIU, where each level *i* stores the closed *i*-itemsets. The parameter *k* is the maximum length of the current closed itemsets. Each node in the DIU stores a closed itemset, its current support information, and the links to its immediate parent and child nodes. We assume in this chapter that all current closed itemsets are already derived, and based on these closed itemsets, we generate association rules for data estimation.

6.3 Missing Data Estimation based on Closed Pattern Association Mining

The closed itemset mining provides the foundation for our data estimation algorithm, CARM. The reason we based CARM on the closed itemsets mining is because not only it forms a non-redundant set of association rules (Zaki, 2000), which helps to achieve the time and space efficiency, but also it provides compact and complete information, which helps to achieve the estimation accuracy. Because without losing any information, we are able to find out all the relationships (rules) between sensors.

Lemma 6.1 The support of an itemset X is equal to the support of its closure, i.e. $s(X) = s(C(X))$.

This lemma, reported in (Pasquier, 1999) and (Zaki, 2000), states that all frequent itemsets are uniquely determined by the frequent closed itemsets.

From Lemma 6.1, we can derive all itemsets' supports through their closed itemsets' supports in the DIU.

Lemma 6.2 The rule $X_1 \xrightarrow{s,c} X_2$ is equivalent to the rule $C(X_1) \xrightarrow{s,c} C(X_2)$.

Proof: For support we have $s = s(X_1 \cup X_2) = |g(X_1 \cup X_2)| = |g(X_1) \cap g(X_2)|$. By Lemma 6.1, we have $s = |g(C(X_1)) \cap g(C(X_2))|$, because the support of an itemset and its closure is the same. The last expression can be rewritten as $s = |g(C(X_1) \cup C(X_2))| = s(C(X_1) \cup C(X_2))$. For confidence, we have $c = s/|g(X_1)| = s/|g(C(X_1))|$.

From Lemma 6.2, we can derive all association rules between itemsets through their closed itemsets in the DIU.

Instead of generating all possible association rules, we generate the rules that have strong relationships with the current round of sensor readings where one or more readings are missing. We achieve this through browsing the DIU, which stores all of the closed itemsets. Based on the users' specified support and confidence thresholds, we find out rules through paths (links) of closed itemsets that suit the users' needs, i.e., satisfy the users' specified support and confidence thresholds. The mining process is online and incremental, which is especially beneficial when the users have different specified threshold criteria in their online queries. The CARM Algorithm is shown in Figure 6-1.

CARM proceeds in the following manner. First, it checks if there are missing values in the current round of readings of stream data. If yes, it uses the current round of readings X that contains the missing items to find out its closure online. If

the rules from X to its immediate upper level supersets satisfy the user-specified support and confidence criteria, these upper level supersets are treated as starting points to explore more potential itemsets until CARM estimates all missing sensor data. Following this method, CARM continues to explore and find all closed itemsets that can generate association rules satisfying the users' specified support and confidence criteria. All these closed itemsets are the supersets of the exploration set and have the support and confidence along the path above or equal to the users' specified support and confidence thresholds.

CARM generates the estimated value based on the rules and selected closed itemsets, which contain item value(s) that are not included in the original readings X. It weighs each rule by its confidence and calculates the summation of these weights multiplied with their associated item values as the final estimated result. These item values can be expected as the missing item values with the support and confidence values equal to or greater than the users' specified thresholds. In this way, CARM takes into consideration all the possible relationships between the sensor readings and weighs each possible missing value by the strength (confidence) of each relationship (rule). This enables CARM to produce a final estimated result near the actual sensor value based on all of the previous sensor relationships information.

Before introducing the CARM Algorithm, we define the symbols to be used in the algorithm. Let $D = \{d_1, d_2,\ldots, d_n\}$ be a set of n item identifiers, and $V = \{v_1, v_2,\ldots, v_m\}$ be a set of m item values. An item J is a combination of D and V, denoted

as $J = D.V$. For example, $d_n.v_m$ means that an item with identifier d_n has the value v_m. In the following figure, X is the itemset in the current round of sensor readings, Y represents all supersets of X, $Conf_y$ represents the strength of the rule from itemset X to Y, support(X) represents X's support, closure(X) is the closure of itemset X in the current transaction sets, min(X) represents X's immediate upper level supersets in the DIU, S represents the support of association rule, C represents the confidence of association rule, $V_{(N)}$ represents the value $V_{(N)}$ of sensor identifier $S_{(N)}$, $X_{estimate}$ represents the returned estimation itemset which contains the senor identifiers with missing values in the current round of readings of stream data and their corresponding estimated values. $S_{specify}$ represents the user-specified support, and $C_{specify}$ represents the user-specified confidence.

Input: (1)X_{input}: the current round of sensor readings that contains missing values
(2)$S_{specify}$: the user-specified minimum support
(3)$C_{specify}$: the user-specified minimum confidence

Output: $X_{estimate}$: a set containing the senor ids with missing values in the current round of sensor readings and their corresponding estimated values

Method:

1 $X_{estimate}=\phi$;
2 $C_{input}=1$;
3 Procedure Estimate(X_{input}, C_{input}, $S_{specify}$, $C_{specify}$)
4 if ($X_{input} \neq \phi$ and $X_{input} = C(X_{input})$)
5 $C=C_{input}$;
6 for all (Y = min(X_{input}))
7 $C=C*(S(Y)/S(X_{input}))$;
8 $X_{new}=Y \setminus X_{input}$;
9 if ($S(Y) > S_{specify}$ and $C>C_{specify}$ and $X_{new} \neq \phi$)
10 for all ($Z \in X_{new}$, Z's new value V)

```
11                          N=index(Z);
12                          V(N)= V(N)+C*value(Z) ;
13                        end for
14                       Estimate(Y, C, Sspecify, Cspecify) ;
15                    end if
16                 end for
17     end if
18     if (Xinput ≠ ϕ and Xinput ≠ C(Xinput))
19             Y=closure(Xinput) ;
20             Xnew=Y\ Xinput;
21             C=1;
22             if (S(Y)> Sspecify and C>Cspecify and Xnew ≠ ϕ)
23                  for all (Z∈Xnew , Z's new value V)
24                     N=index(Z) ;
25                     V(N)= V(N)+C*value(Z) ;
26                   end for
27                   Estimate(Y, C, Sspecify, Cspecify) ;
28             end if
29       end if
30       Xestimate = Xinput ∪ Xnew
31 end procedure
```

Figure 6-1: The online data estimation algorithm

6.4 Comparing with Existing Literature

Table 6-1 summarizes the recent data estimation approaches, which can be categorized according to the different methodologies. As we discussed in Section 2.3, the traditional statistical methods do not suitable to be used in the data stream environment. Methods based on time series estimate the missing data based on its time trends, but in the sensor stream database, sensor data is not only related with time trends, other factors such as location can also affect the data relationships. Methods based on pattern and association mining can discover implicit relationships

between data. In (Tarui, 2007), the author discussed how to find a duplicate and a single missing item in a stream. In (Halatchev, 2005) (Gruenwald, 2007), the authors propose to use two frequent itemset mining technique to perform estimation based on relationship between two sensors. Based on our proposed pattern and association mining technique discussed in Chapter 4 and 5, we developed a technique to perform missing data estimation considering the relationship between multiple sensor readings.

				Number of Itemsets	Data Stream Support
Data Estimation	Static Data	Statistics	Iannacchione 82, Rubin 96, Shafer 95, Cool 00	N/A	No
	Stream Data	Time Series	Papadimitriou 05	N/A	Yes
		Pattern and Association Mining	Tarui 07	One	Yes
			Halatchev 05, Gruenwald 07	Two	Yes
			Proposed 07	Multiple	Yes

Table 6-1: Recent data estimation approaches

6.5 Summary

In this chapter we proposed a novel algorithm, called CARM, to perform data estimation in sensor network databases based on closed pattern mining in sensor streams. The algorithm offers an online method to derive association rules based on the discovered closed itemsets, and estimates the missing sensor values based on the derived association rules. It can find out the relationships between multiple sensors

not only when they report the same sensor readings but also when they report different sensor readings.

7 Performance Study

7.1 Overview

In this chapter, we describe experimental study and results of our proposed techniques. Section 7.2, 7.3 and 7.4 describe the performance study and analysis for the content discussed in Chapter 4, 5, and 6 respectively. Section 7.5 summarizes this chapter.

For the performance study, nine synthetic datasets *T*5.*I*6.*D*1*K*, *T*5*I*6*D*10*K*, *T*5*I*6*D*20*K*, *T*5*I*6*D*100*K*, *T*5*I*10*D*10*K*, *T*5*I*12*D*10*K*, *T*10*I*6*D*10*K*, *T*12*I*6*D*10*K*, *T*5.*I*6.*D*10*K-AB* and two real datasets are used to evaluate the performance of proposed techniques. Each synthetic dataset is generated by the same method as described in (Agrawal 1993), where the three numbers of each dataset denote the average transaction size (*T*), the average maximal potential frequent itemset size (*I*) and the total number of transactions (*D*), respectively. The first real dataset was collected in year 2000 at various locations throughout the city of Austin, Texas. The data represents the current location, the time interval, and the number of vehicles detected during this interval. All sensor nodes report to a single server. The sensors are deployed on city streets, collect and store the number of the vehicles detected for a given time interval. The vehicle counts taken as sensor readings that are used as input for our simulation experiments are traffic data provided by (Austin, 2003). The second real dataset was sensor data collected in the Huntington Botanical Garden in

Sam Marino, California (Huntington, 2008). The sensor reports the air temperature of several places in the gardens for different time intervals. In the experiments, the transactions of each dataset are looked up one by one in sequence to simulate the environment of an online data stream. All our experiments were done on a 1.60 GHz Intel Core 2 CPU with 2GB memory.

7.2 Performance Study for Closed Pattern Mining

We compare our algorithm with Moment (Chi 2004), which is the state-of-the-art algorithm to mine closed itemsets in data streams and closet+ (Pei 2003), which is the state-of-the-art closed itemsets mining algorithm for traditional databases. For the performance study, synthetic datasets *T*5.*I*6.*D*1*K*, *T*5*I*6*D*10*K*, *T*5*I*6*D*100*K*, *T*5*I*10*D*10*K*, *T*5*I*12*D*10*K*, *T*10*I*6*D*10*K*, *T*12*I*6*D*10*K*, *T*5.*I*6.*D*10*K-AB* are used to evaluate the performance of the CFI-Stream Algorithm. The figures and tables in this section show the average running time per transaction and memory usage in terms of the number of stored itemsets in the above synthetic datasets.

7.2.1 Performance under Different Total Number of Transactions

In this experiment, we compare CFI-Stream, Moment (Chi 2004) and Closet+ (Pei 2003) under different total number of transactions. As shown in Figure 7-1 and Table 7-1, as the total number of transaction size increases, the running time per transaction of CFI-Stream, Moment and Closet+ fluctuate in a certain range, among which Closet+ fluctuates the most. From Figure 7-1, we can also see that for the

given three datasets with specified parameters, CFI-Stream gives the fast running time, follows by Closet+ and Moment.

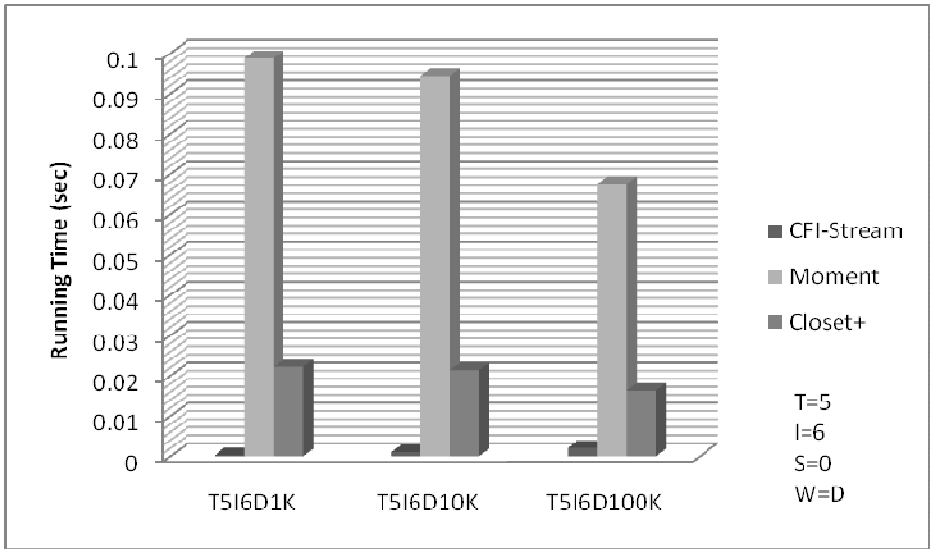

Figure 7-1: Running time per transaction under different total number of transaction size in seconds

	T5I6D1K	*T5I6D10K*	*T5I6D100K*
CFI-Stream	0.000303	0.000997	0.00205596
Moment	0.09875	0.09444	0.0676
Closet+	0.022188	0.0213424	0.01635217

Table 7-1: Running time per transaction under different total number of transaction size in seconds

Figure 7-2 and Table 7-2 show that as the total number of transaction size increases, for CFI-Stream and Closet+, the number of itemsets stored in the memory is the same as the number of closed itemsets, which increases when the transaction size increases. While for Moment, the memory space usage increased faster than CFI-Stream and Closet+; this is because the Moment Algorithm needs to store all the

boundary nodes, which include all the infrequent gateway nodes, unpromising gateway nodes, intermediate nodes, and closed nodes. The number of boundary nodes as well as the closed nodes increase while the total number of transaction size increases.

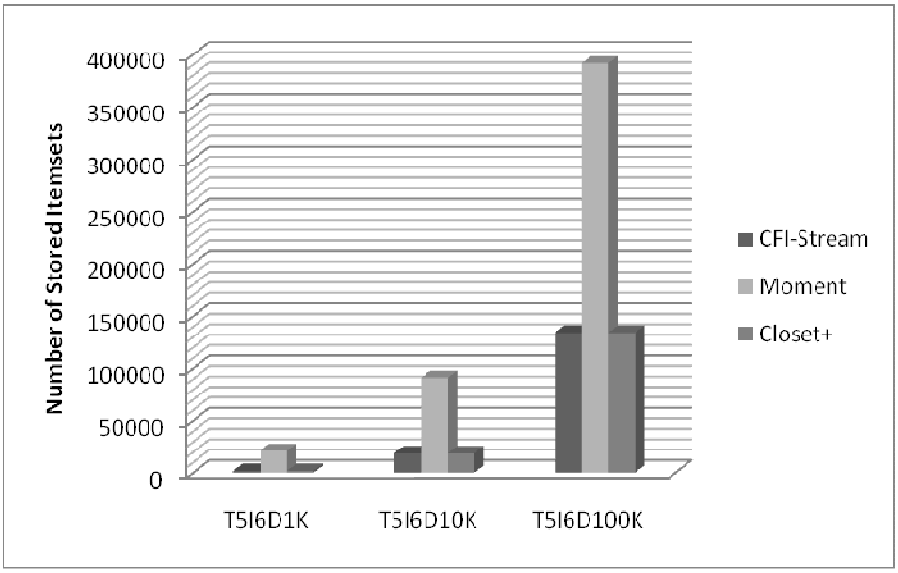

Figure 7-2: Memory usage in terms of number of stored itemsets under different total number of transaction size

	T5I6D1K	*T5I6D10K*	*T5I6D100K*
CFI-Stream	1925	18728	134010
Moment	21198	91000	391456
Closet+	1925	18728	134010

Table 7-2: Memory usage in terms of number of stored itemsets under different total number of transaction size

7.2.2 Performance under Different Sliding Window Size

In this experiment, we compare CFI-Stream, Moment (Chi 2004) under different sliding window sizes. As shown in Figure 7-3 and Table 7-3, as the sliding window size increases, the running time per transaction of CFI-Stream and Moment

fluctuate in a certain range. Also we can see from Figure 7-3 and Table 7-3 that CFI-Stream runs faster than Moment when processing the closed pattern mining with different sliding window size under the given datasets and parameters.

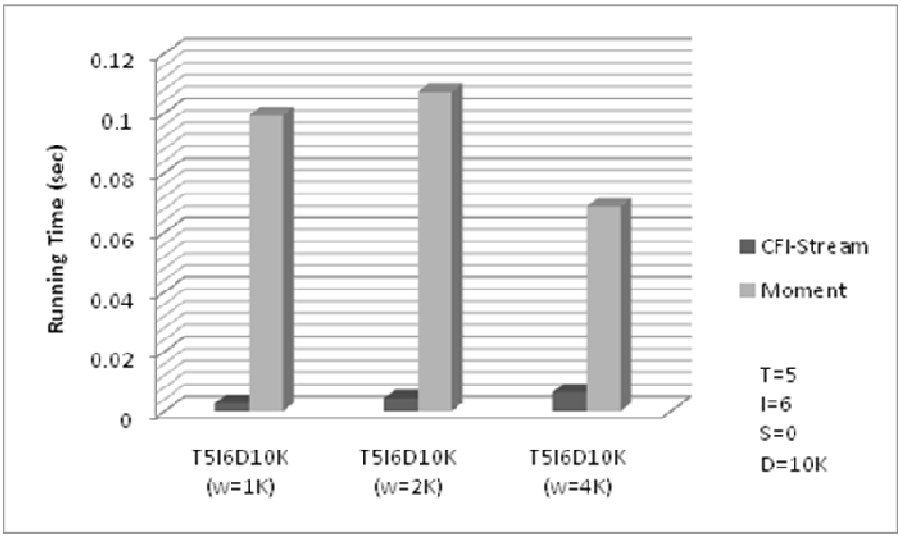

Figure 7-3: Running time per transaction under different sliding window size in seconds

	T5I6D10K (*w*=1*K*)	*T5I6D10K* (*w*=2*K*)	*T5I6D10K* (*w*=4*K*)
CFI-Stream	0.0027569	0.0043946	0.0064299
Moment	0.09929	0.10713	0.06874

Table 7-3: Running time per transaction under different sliding window size in seconds

Figure 7-4 and Table 7-4 show that as the sliding window size increases, for CFI-Stream, the number of itemsets stored in the memory is the same as the number of closed itemsets, which increases when the transaction size increases. While for Moment, the memory space usage increased faster than CFI-Stream; this is because the Moment Algorithm needs to store all the boundary nodes, which include the

infrequent gateway nodes, unpromising gateway nodes, intermediate nodes, and closed nodes. The number of boundary nodes as well as the closed nodes increases when the sliding window size increases.

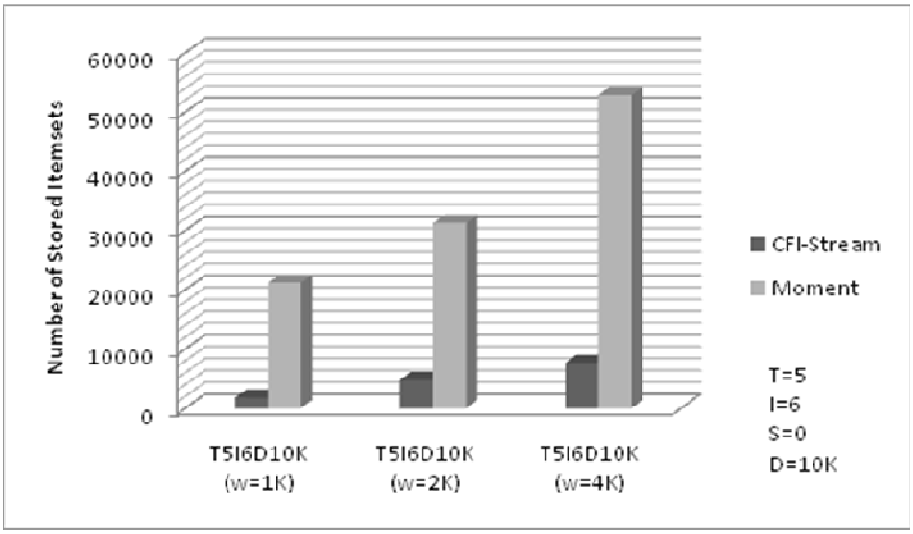

Figure 7-4: Memory usage in terms of number of stored itemsets under different sliding window size

	*T*5*I*6*D*10*K* (*w*=1*K*)	*T*5*I*6*D*10*K* (*w*=2*K*)	*T*5*I*6*D*10*K* (*w*=4*K*)
CFI-Stream	1768	4810	7660
Moment	21198	31271	52878

Table 7-4: Memory usage in terms of number of stored itemsets under different sliding window size

7.2.3 Performance under Different Minimum Support Threshold

Figure 7-5 and Table 7-5 show the average processing time per transaction for Closet+, Moment and CFI-Stream under different minimum support thresholds. As the minimum support threshold decreases, the running time per transaction for

Moment, CFI-Stream and Closet+ decreases as illustrated in Figure 7-5 for the given datasets and parameters.

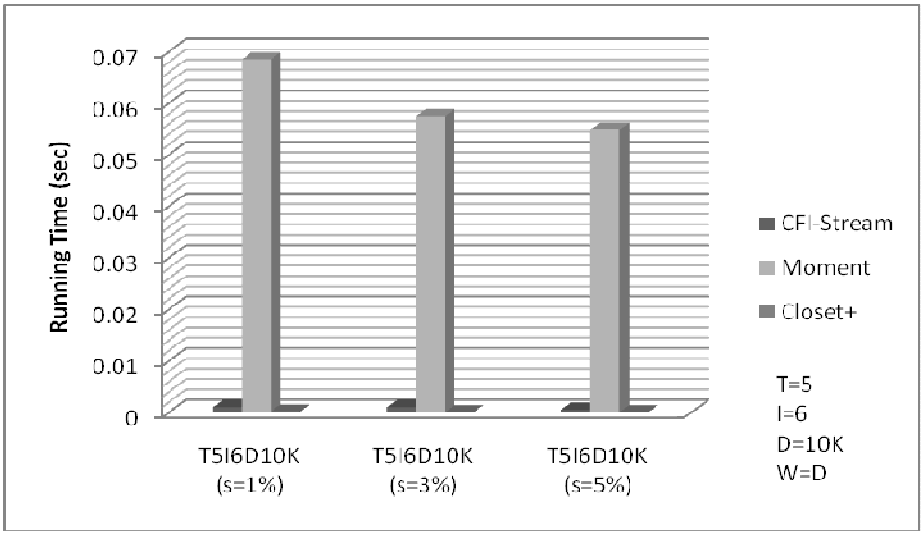

Figure 7-5: Running time per transaction under different minimum support threshold in seconds

	*T*5*I*6*D*10*K* (*s*=1%)	*T*5*I*6*D*10*K* (*s*=3%)	*T*5*I*6*D*10*K* (*s*=5%)
CFI-Stream	0.0009549	0.0009521	0.0004796
Moment	0.06848	0.05752	0.05479
Closet+	0.000138	0.0000077	0.00000355

Table 7-5: Running time per transaction under different minimum support threshold in seconds

Figure 7-6 and Table 7-6 show the memory usage in terms of the number of stored itemsets of Closet+, Moment and CFI-Stream under different minimum support thresholds. As shown in this figure, the memory usage for Closet+ and Moment decreases when the minimum support threshold increases. This is because the number of itemsets it keeps track of decreases. For CFI-Stream, it keeps track of

all the current closed itemsets independent of support information, therefore the number of stored itemsets did not change with the support information.

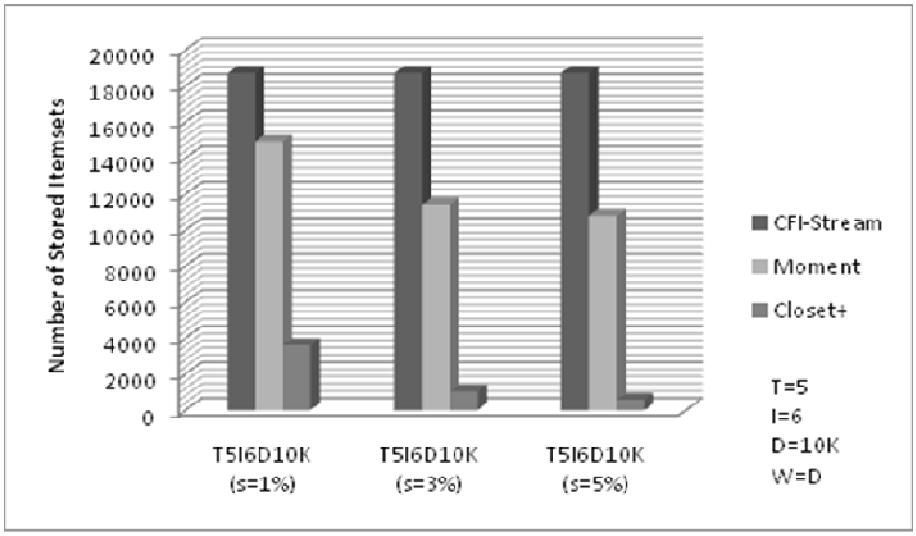

Figure 7-6: Memory usage in terms of number of stored itemsets under different minimum support threshold

	*T*5*I*6*D*10*K* (*s*=1%)	*T*5*I*6*D*10*K* (*s*=3%)	*T*5*I*6*D*10*K* (*s*=5%)
CFI-Stream	18728	18728	18728
Moment	14926	11424	10801
Closet+	3608	1019	581

Table 7-6: Memory usage in terms of number of stored itemsets under different minimum support threshold

7.2.4 Performance under Different Average Transaction Size

Figure 7-7 and Table 7-7 show the average processing time for Closet+, Moment and CFI-Stream under different average transaction sizes. As the average transaction size increases, the running time for CFI-Stream, Moment and Closet+ increases as illustrated in Figure 7-7.

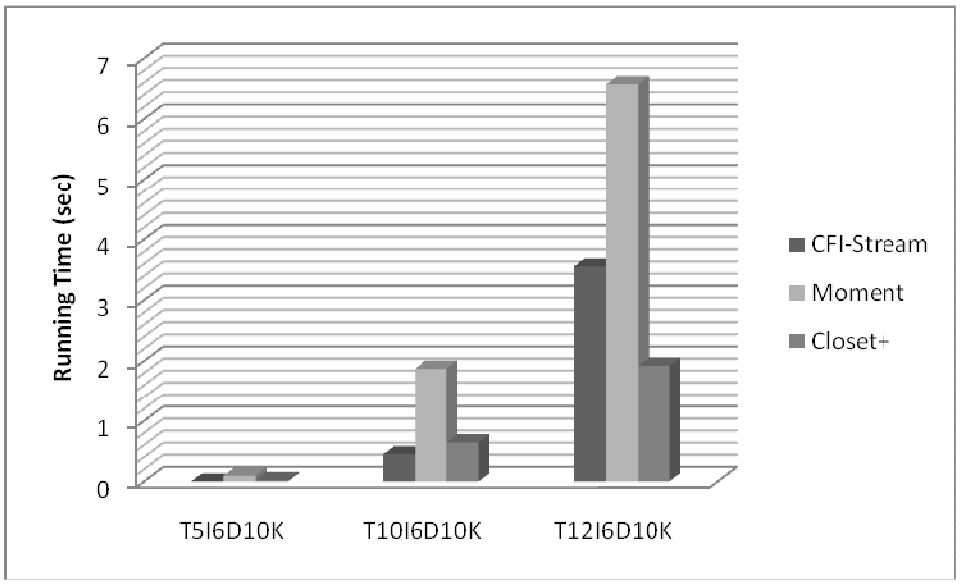

Figure 7-7: Running time per transaction under different average transaction size in seconds

	*T*5*I*6*D*10*K*	*T*10*I*6*D*10*K*	*T*12*I*6*D*10*K*
CFI-Stream	0.000997	0.445898	3.55638
Moment	0.09444	1.87323	6.56796
Closet+	0.0213424	0.644135	1.9165038

Table 7-7: Running time per transaction under different average transaction size in seconds

Figure 7-8 and Table 7-8 show the memory usage in terms of the number of stored itemsets of Closet+, Moment and CFI-Stream while the average transaction size increases. As shown in this figure, the memory usage for the three algorithms increases when the average transaction size increases. This is because the number of itemsets it keeps track of increases. Also we can see from the figure that the CFI-Stream and Closet+ Algorithm consumes less memory space than the Moment Algorithm, because they only need to keep track of the closed itemsets. While

Moment keeps track of all the infrequent gateway nodes, unpromising gateway nodes, intermediate nodes, and closed nodes.

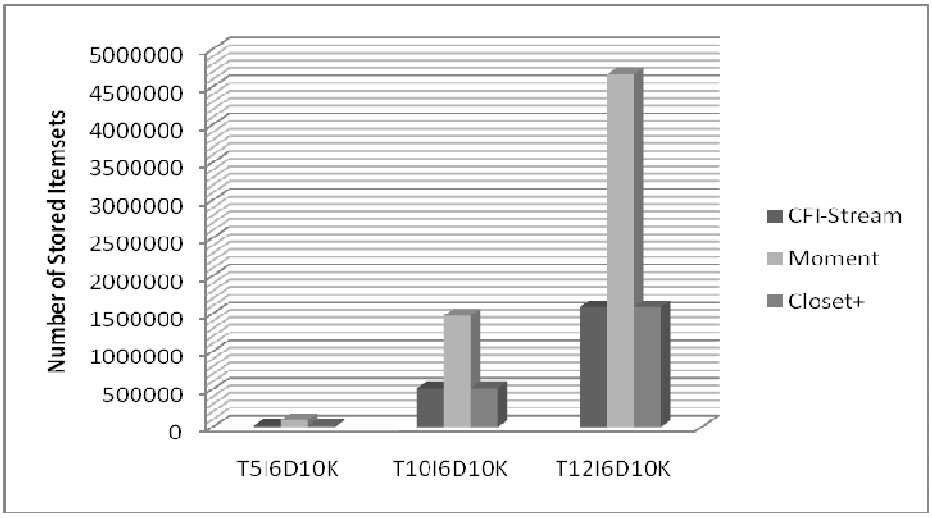

Figure 7-8: Memory usage in terms of number of stored itemsets under different average transaction size

	*T*5*I*6*D*10*K*	*T*10*I*6*D*10*K*	*T*12*I*6*D*10*K*
CFI-Stream	18728	512923	1583586
Moment	91000	1472744	4667617
Closet+	18728	512923	1583586

Table 7-8: Memory usage in terms of number of stored itemsets under different average transaction size

7.2.5 Performance under Different Average Maximal Potential Frequent Itemset Size

Figure 7-9 and Table 7-9 show the running time for Closet+, Moment and CFI-Stream under different average maximal potential frequent itemset sizes. As the average maximal potential frequent itemset size increases, the running time for CFI-

Stream, Moment and Closet+ increases as illustrated in Figure 7-9 with the given datasets and parameters.

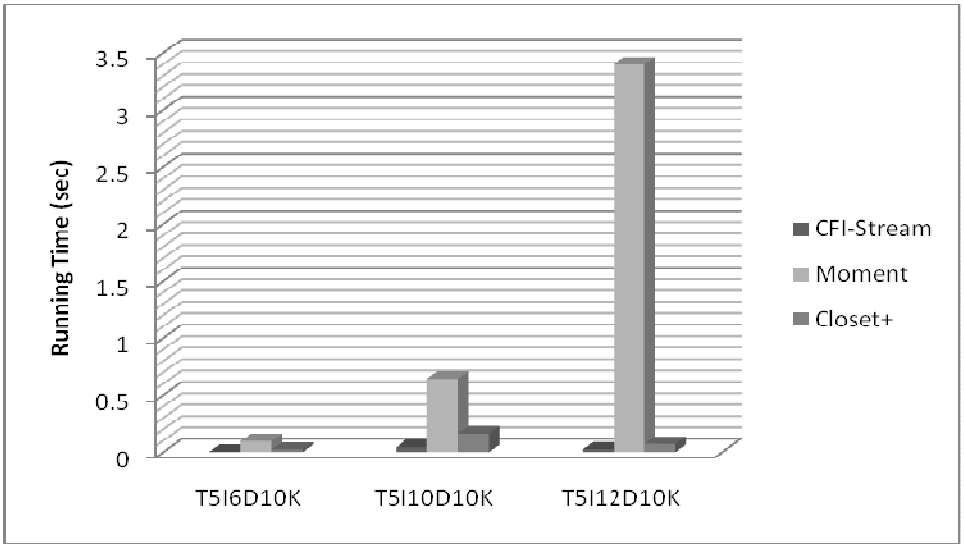

Figure 7-9: Running time per transaction under different average maximal potential frequent itemset size in seconds

	*T*5*I*6*D*10*K*	*T*5*I*10*D*10*K*	*T*5*I*12*D*10*K*
CFI-Stream	0.000997	0.0422233	0.023927
Moment	0.09444	0.64178	3.39715
Closet+	0.0213424	0.1622659	0.0704573

Table 7-9: Running time per transaction under different average maximal potential frequent itemset size in seconds

Figure 7-10 and Table 7-10 show the memory usage in terms of the number of stored itemsets of Closet+, Moment and CFI-Stream under different average maximal potential frequent itemset sizes. As shown in this figure, the memory usage for the three algorithms increases when the average maximal potential frequent itemset size increases. This is because the number of itemsets it keeps track of increases. Also we can see from the figure that the CFI-Stream and Closet+

Algorithm consume less memory space than the Moment Algorithm, because they only need to keep track of the closed itemsets. While Moment keeps track of all the infrequent gateway nodes, unpromising gateway nodes, intermediate nodes, and closed nodes.

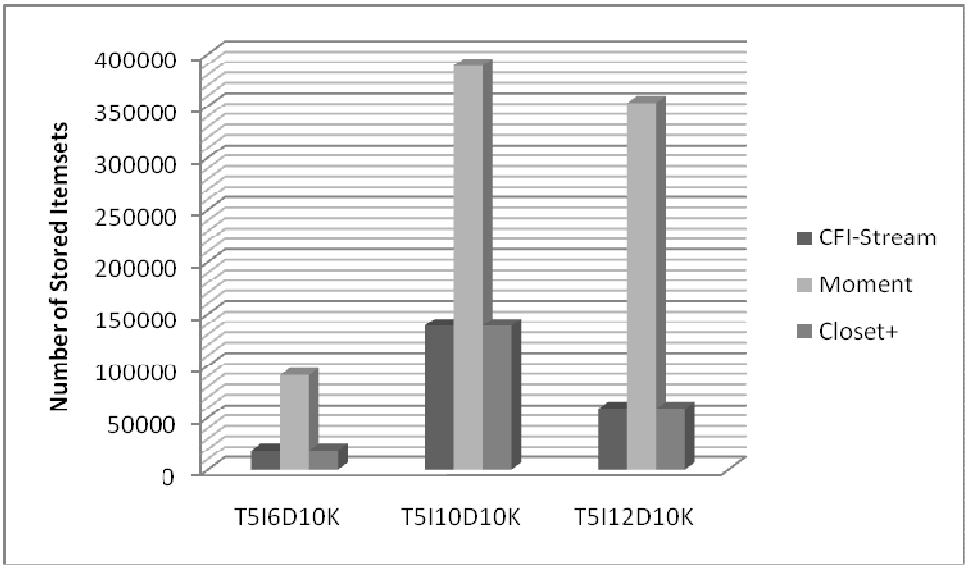

Figure 7-10: Memory usage in terms of number of stored itemsets under different average maximal potential frequent itemset size

	*T*5*I*6*D*10*K*	*T*5*I*10*D*10*K*	*T*5*I*12*D*10*K*
CFI-Stream	18728	138363	58785
Moment	91000	388602	353126
Closet+	18728	138363	58785

Table 7-10: Memory usage in terms of number of stored itemsets under different average maximal potential frequent itemset size

7.2.6 Performance under Data Variation

Figure and Table 7-11 and 7-12 show the adaptability of the CFI-Stream method to the change in data streams. In this experiment, the dataset *T*5*I*6*D*10*K* and

*T*5.*I*6.*D*10*K-AB* is used. The dataset *T*5.*I*6.*D*10*K-AB* is composed of two consecutive subparts. The first part is a set of 5,000 transactions generated by an item set *A*, while the second part is a set of 5,000 transactions generated by an item set *B*. There are no common items in the item sets *A* and *B*. We use the coverage rate CR(*X*) proposed by Chang et al in (Chang, 2003) to illustrate the concept drift property of dataset *T*5.*I*6.*D*10*K-AB*. CR(*X*) denotes the ratio of closed frequent itemsets introduced by an item set *X* in all closed frequent itemsets as follows:

$$\mathrm{CR}(X) = \frac{\#\ of\ \ closed\ \ frequent\ itemsets\ induced\ by\ an\ item\ set\ X}{|R|} \times 100(\%)$$

where |*R*| denotes the total number of closed frequent itemsets in a data stream. In the first 5,000 transactions, which are generated by an item set *A*, all the new coming closed frequent itemsets are introduced by the item set *A*, therefore the coverage rate CR(*A*) is a hundred percent, while the coverage rate CR(*B*) is zero. In the second 5,000 transactions, all closed itemsets are generated by the item set *B*, not containing any item from set *A*, therefore the final coverage rate CR(*A*) is 50%, and CR(*B*) is 50%. From Figure 7-11 and 7-12, we can see that the running time and memory space consumption of CFI-Stream didn't fluctuate much while using the dataset with concept drift, which is favorable when processing data streams with different data distribution.

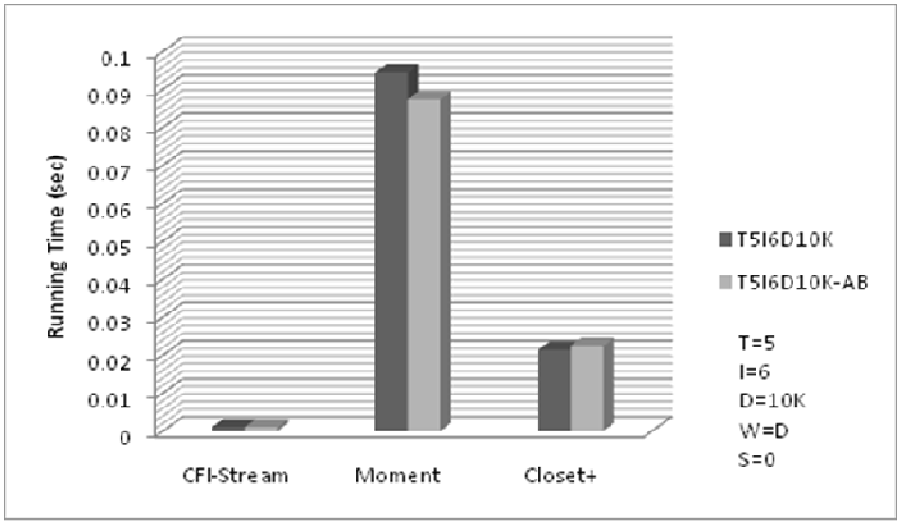

Figure 7-11: Running time per transaction under data variation in seconds

	T5I6D10K	*T5I6D10K-AB*
CFI-Stream	0.000997	0.0009332
Moment	0.09444	0.08734
Closet+	0.0213424	0.0224317

Table 7-11: Running time per transaction under data variation in seconds

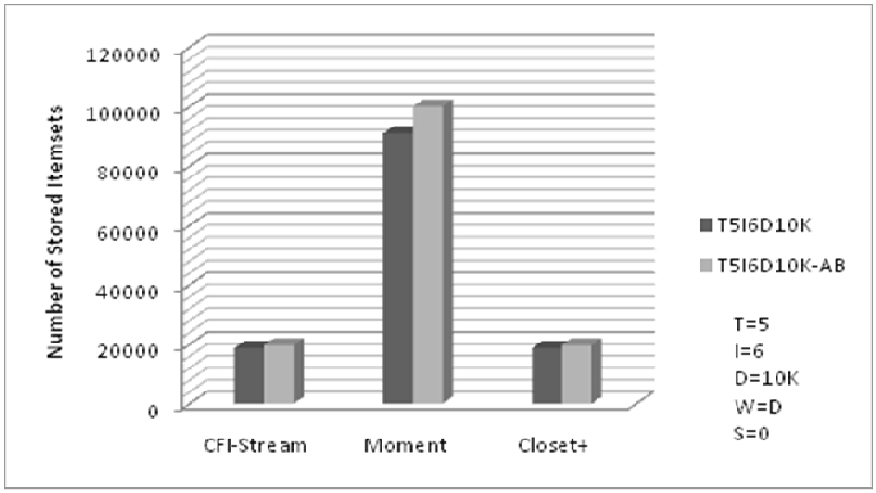

Figure 7-12: Memory usage in terms of number of stored itemsets under data variation

	*T*5*I*6*D*10*K*	*T*5*I*6*D*10*K-AB*
CFI-Stream	18728	19767
Moment	91000	100038
Closet+	18728	19767

Table 7-12: Memory usage in terms of number of stored itemsets under different data variation

7.3 Performance Study for Association Mining

In this section, we describe the experimental study and results of the proposed informative association mining framework. We compare our algorithm in the proposed association mining framework with the fast implementation of the Apriori Algorithm presented in (Fedor 2003), and the Charm Algorithm, which is a non-redundant association rule mining algorithm for traditional databases proposed in (Zaki, 2005) in traditional association mining framework. For the performance study, synthetic datasets *T*5.*I*6.*D*1*K*, *T*5*I*6*D*10*K*, *T*5*I*6*D*20*K*, *T*5*I*10*D*10*K*, *T*10*I*6*D*10*K*, *T*5.*I*6.*D*10*K-AB* are used to evaluate the performance of the informative association rule mining algorithm. The dataset is generated by the same method as described in (Agrawal, 1994), where the three numbers of each dataset denote the average transaction size (*T*), the average maximal potential frequent itemset size (*I*) and the total number of transactions (*D*), respectively. In each of the following studies, we compare the number of rules generated and the computation time under different experimental parameters. The figures and tables in this section show the total running time and number of generated rules performance under different association frameworks in the above synthetic datasets. In our proposed association

mining framework as described in Chapter 5, we calculate the average running time for each transaction to update the DIU, and the total association mining time for the above synthetic datasets. In the comparing traditional sequential association mining framework, we calculate the total running time to generate frequent or closed itemsets and associations in the above synthetic datasets.

7.3.1 Performance under Different Total Number of Transactions

From Figure 7-13 and Table 7-13, we can see that as the total number of transaction size increases, the number of rules generated by the three comparing algorithms increases. The number of rules generated by CFI-R is less than the number of rules generated by Charm and is much smaller than those generated by Apriori. This is because CFI-R and Charm derived the non-redundant association rules using the closed frequent itemsets according to different non-redundant rule definitions, while Apriori uses all the frequent itemsets to generate association rules, which contain a lot of redundant information (Zaki, 2000).

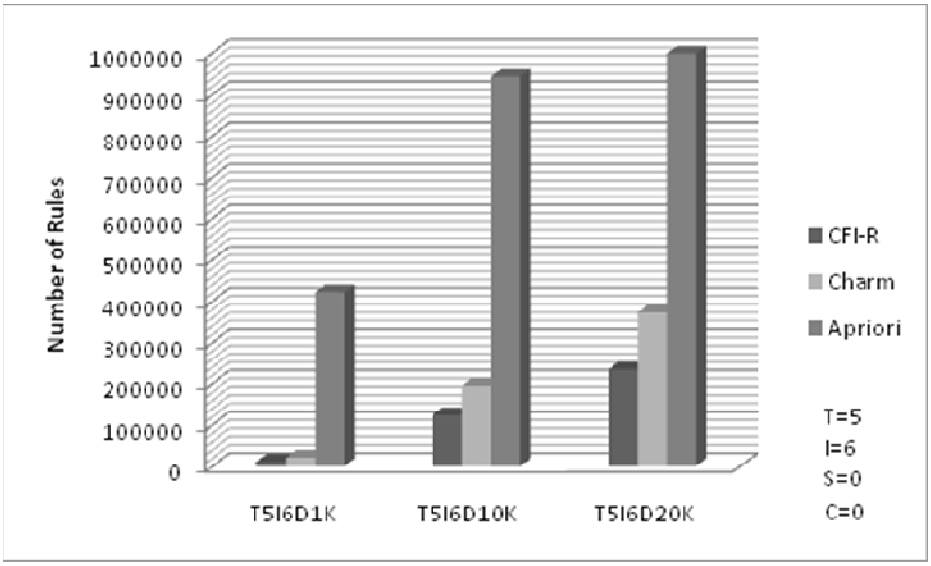

Figure 7-13: Number of rules generated under different total number of transactions

	T5I6D1K	*T5I6D10K*	*T5I6D20K*
CFI-R	10397	123688	233931
Charm	20986	194798	372276
Apriori	421822	944569	998049

Table 7-13: Number of rules generated under different total number of transactions

From Figure 7-14 and Table 7-14, we can see that the running time of Apriori is smaller than Charm and CFI-R. That is because the rules generated by Apriori directly come from all frequent itemsets, while both Charm and CFI-R need to generate closed frequent itemsets to produce the non-redundant association rules. Therefore the calculation time increases.

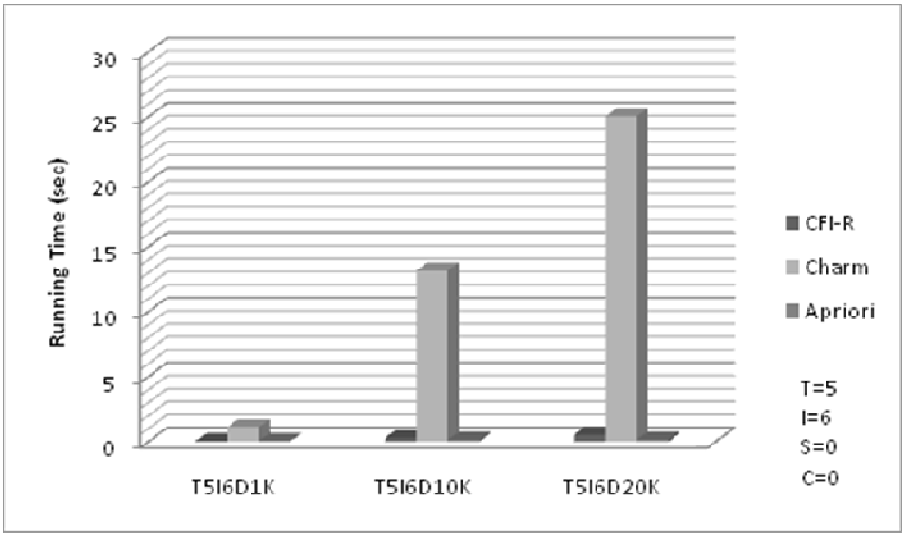

Figure 7-14: Running time under different total number of transactions in seconds

	T5I6D1K	*T5I6D10K*	*T5I6D20K*
CFI-R	0.183303	2.812997	5.83540325
Charm	1.11934	13.16552	25.04471
Apriori	0.06	0.14	0.17

Table 7-14: Running time under different total number of transactions in seconds

7.3.2 Performance under Different Minimum Support Threshold

Figure 7-15 and Table 7-15 show that the number of rules generated decreases as the minimum support threshold increases in Apriori, Charm and CFI-R, because when the user-specified support threshold increases, the number of rules that satisfy the criteria will decrease as well.

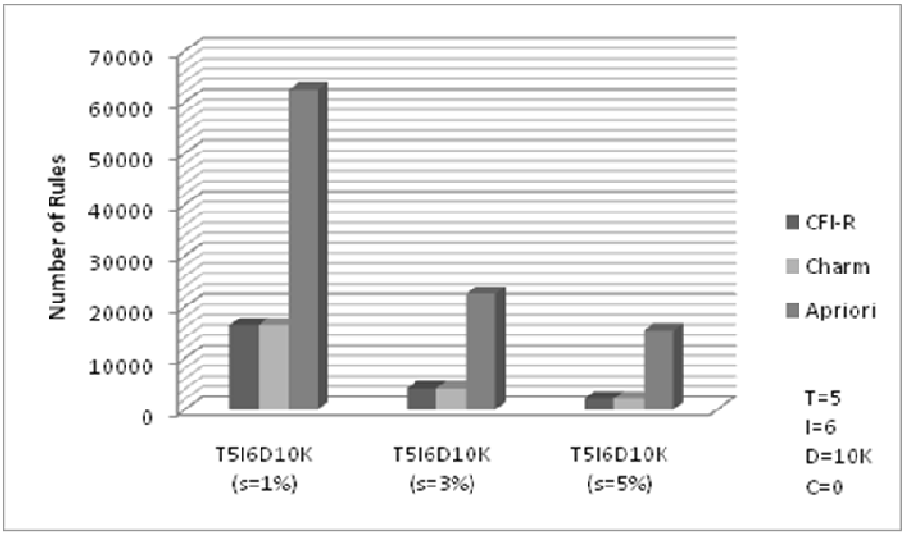

Figure 7-15: Number of rules generated under different minimum support threshold

	*T*5*I*6*D*10*K* (*s*=1%)	*T*5*I*6*D*10*K* (*s*=3%)	*T*5*I*6*D*10*K* (*s*=5%)
CFI-R	16430	4111	2110
Charm	16430	4110	2110
Apriori	62453	22624	15351

Table 7-15: Number of rules generated under different minimum support threshold

Figure 7-16 and Table 7-16 show that for both Apriori and Charm, the running time decreases as the user-specified support threshold increases. That is because when the user-specified support threshold increases, the number of rules generated will be decreased, and therefore the calculation time decreases as well. The running time for CFI-R didn't change much because it finds out complete closed itemsets independent of support information, and in the rule mining stage it filters out the rules whose support and confidence is less than the user-specified thresholds.

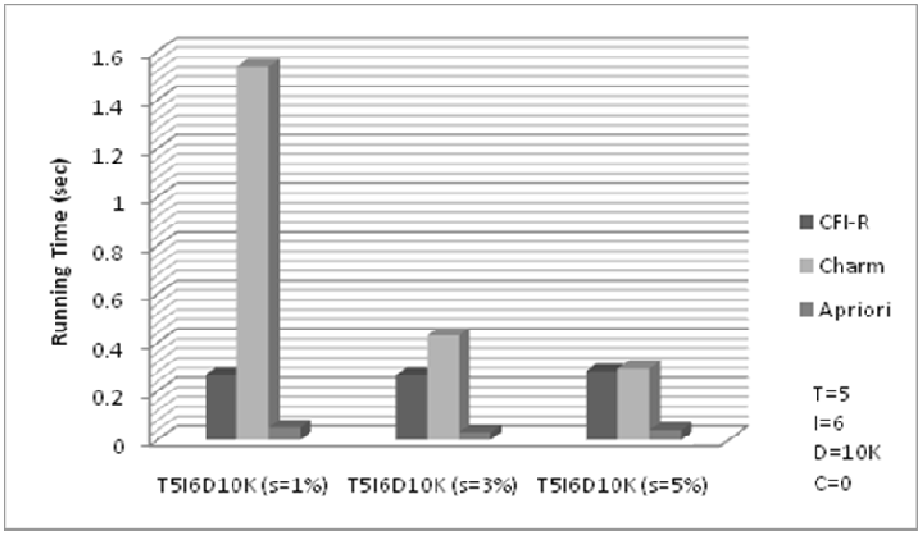

Figure 7-16: Running time under different minimum support threshold in seconds

	*T*5*I*6*D*10*K* (*s*=1%)	*T*5*I*6*D*10*K* (*s*=3%)	*T*5*I*6*D*10*K* (*s*=5%)
CFI-R	0.265997	0.265997	0.281997
Charm	1.537489	0.430442	0.29365
Apriori	0.05	0.03	0.04

Table 7-16: Running time under different minimum support threshold in seconds

7.3.3 Performance under Different Minimum Confidence Threshold

From Figure 7-17 and Table 7-17, we can see that the number of rules generated decreases under different minimum confidence thresholds. Because when the user-specified confidence threshold increases, the number of rules that satisfies the query criteria will decrease. The amount of rules generated by Apriori Algorithm is largest, because it is generated based on frequent itemsets. The number of rules generated by Charm and CFI-R Algorithms are smaller, because they are generated based on closed itemsets.

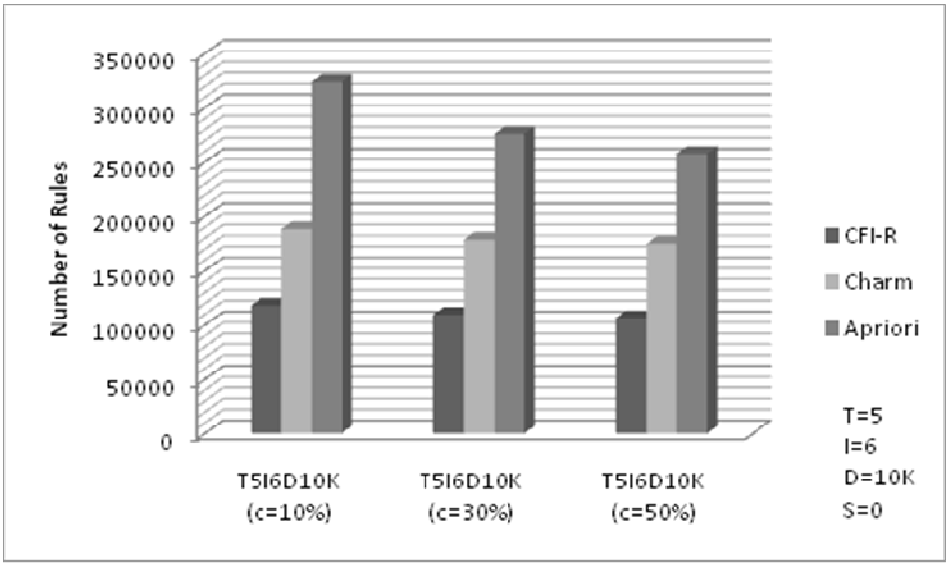

Figure 7-17: Number of rules generated under different minimum confidence threshold

	*T*5*I*6*D*10*K* (*c*=10%)	*T*5*I*6*D*10*K* (*c*=30%)	*T*5*I*6*D*10*K* (*c*=50%)
CFI-R	117941	109407	106375
Charm	188039	178616	174830
Apriori	324121	275989	257584

Table 7-17: Number of rules generated under different minimum confidence threshold

Figure 7-18 and Table 7-18 illustrate the running time under different minimum confidence thresholds. We can see that the running time for Charm and CFI-R Algorithm is greater than the Apriori Algorithm. This is because both Charm and CFI-R need to generate closed frequent itemsets to produce the non-redundant association rules. Therefore the calculation time increases.

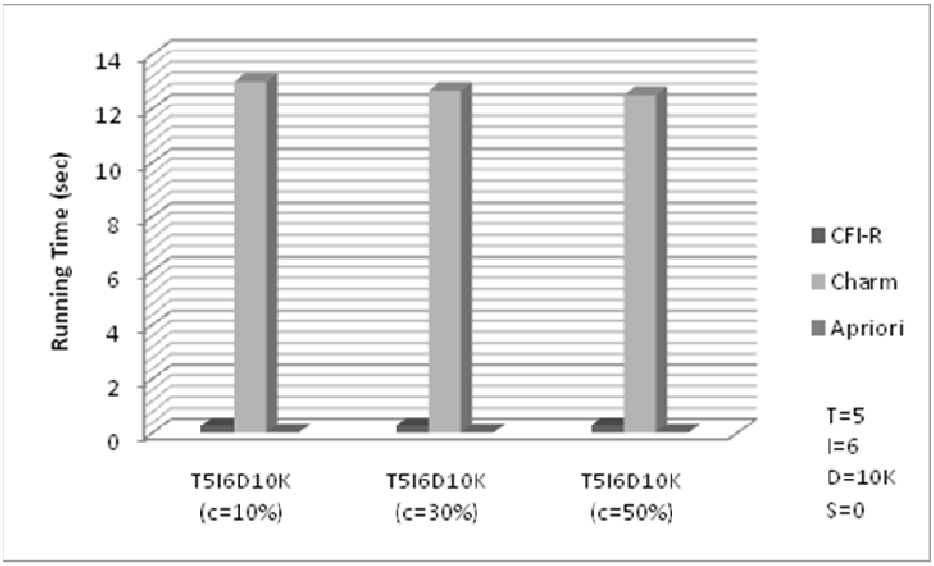

Figure 7-18: Running time under different minimum confidence threshold in seconds

	*T*5*I*6*D*10*K* (*c*=10%)	*T*5*I*6*D*10*K* (*c*=30%)	*T*5*I*6*D*10*K* (*c*=50%)
CFI-R	0.265997	0.266997	0.281997
Charm	12.98839	12.68256	12.51883
Apriori	0.04	0.04	0.04

Table 7-18: Running time under different minimum confidence threshold in seconds

7.3.4 Performance under Different Average Transaction Size

Figure 7-19 and Table 7-19 show the number of rules generated under different average transaction sizes. We can see that for all three algorithms the number of rules generated increases when the average transaction size increases, because the number of frequent and closed itemsets increases as the average transaction size increases.

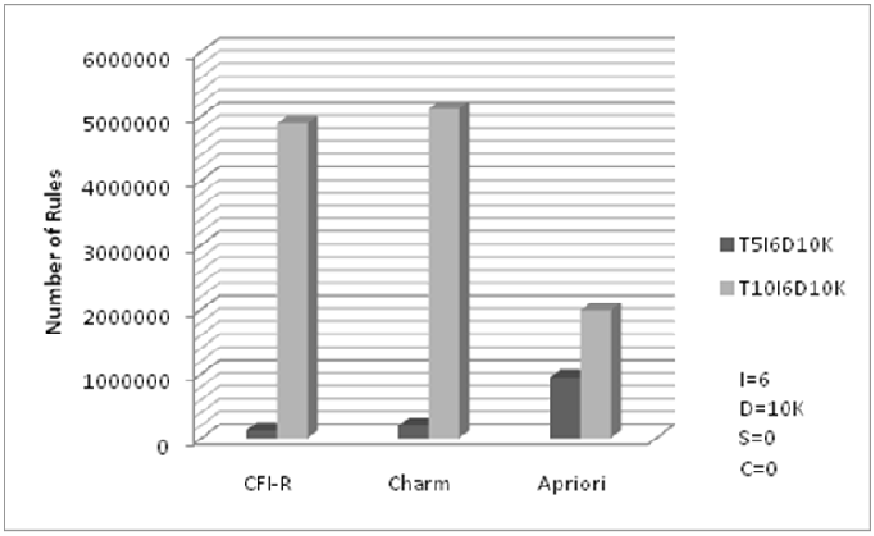

Figure 7-19: Number of rules generated under different average transaction size in seconds

	*T*5*I*6*D*10*K*	*T*10*I*6*D*10*K*
CFI-R	123688	4887155
Charm	194798	5112739
Apriori	944569	1981482

Table 7-19: Number of rules generated under different average transaction size in seconds

Figure 7-20 and Table 7-20 show the running time under different average transaction sizes for CFI-R, Charm and Apriori Algorithm. We can see that as the average transaction size increases, the running time increases for all three algorithms. This is because both the number of closed itemsets and frequent itemsets increase while the average transaction size increases, and the calculation time increases with the increment of the number of frequent and closed itemsets.

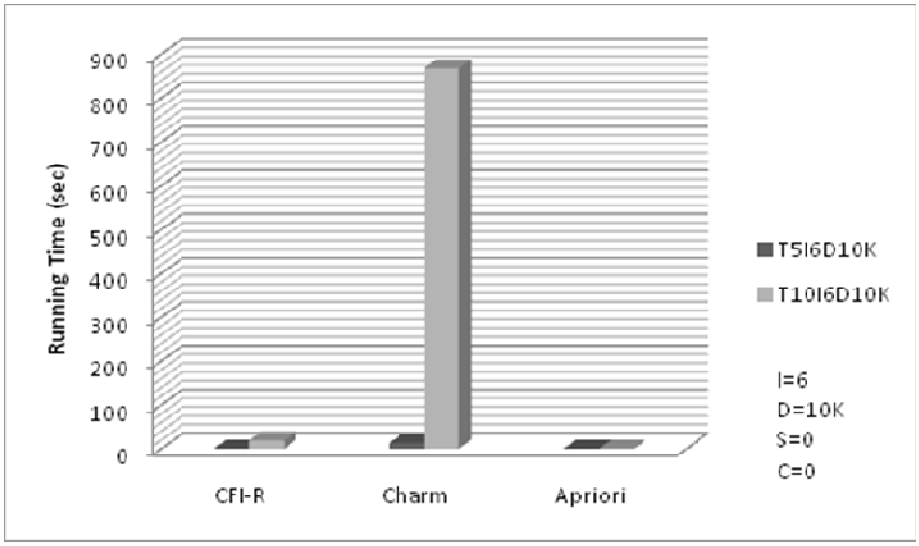

Figure 7-20: Running time under different average transaction size in seconds

	*T*5*I*6*D*10*K*	*T*10*I*6*D*10*K*
CFI-R	0.265997	19.040898
Charm	13.16552	868.031
Apriori	0.14	0.24

Table 7-20: Running time under different average transaction size in seconds

7.3.5 Performance under Different Average Maximal Potential Frequent Itemset Size

Figure 7-21 and Table 7-21 show the number of rules generated under different average maximal potential frequent itemset sizes for CFI-R, Charm and Apriori Algorithm. We can see that as the average maximal potential frequent itemset size increases, the number of rules generated increases for all three algorithms. This is because both the number of closed itemsets and frequent itemsets increase while the average maximal potential frequent itemset size increases.

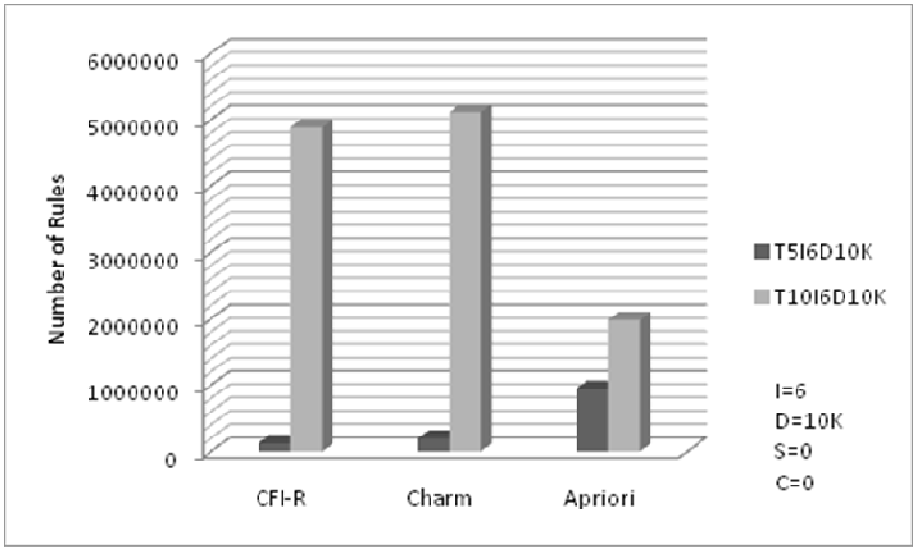

Figure 7-21: Number of rules generated under different average maximal potential frequent itemset size

	*T*5*I*6*D*10*K*	*T*5*I*10*D*10*K*
CFI-R	123688	1503616
Charm	194798	1546412
Apriori	944569	5302210

Table 7-21: Number of rules generated under different average maximal potential frequent itemset size

Figure 7-22 and Table 7-22 show the running time under different average maximal potential frequent itemset sizes for CFI-R, Charm and Apriori Algorithm. We can see that as the average maximal potential frequent itemset size increases, the running time increases for all three algorithms. This is because both the number of closed itemsets and frequent itemsets increase while the average maximal potential frequent itemset size increases, and the calculation time increases with the increment of the number of frequent and closed itemsets.

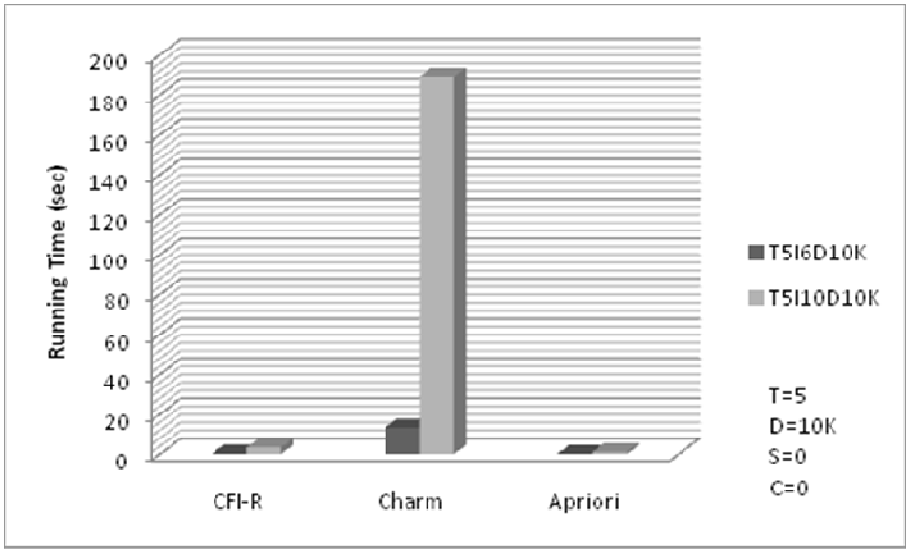

Figure 7-22: Running time under different average maximal potential frequent itemset size in seconds

	*T*5*I*6*D*10*K*	*T*5*I*10*D*10*K*
CFI-R	0.265997	3.3812233
Charm	13.16552	189.1896
Apriori	0.14	1.15

Table 7-22: Running time under different average maximal potential frequent itemset size in seconds

7.3.6 Performance under Data Variation

Figure and Table 7-23 and 7-24 show that number of rules generated and running time for CFI-R, Charm, and Apriori Algorithm. We can see that the performance of CFI-R Algorithm didn't fluctuate much under the data variation, which is a preferable characteristic in data streaming applications.

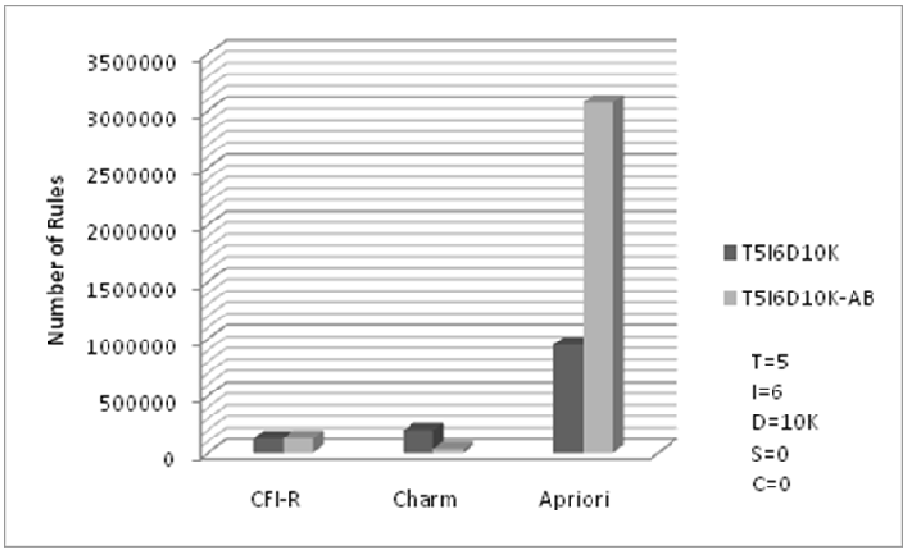

Figure 7-23: Number of rules generated under data variation

	T5I6D10K	*T5I6D10K*-AB
CFI-R	123688	126918
Charm	194798	32454
Apriori	944569	3071352

Table 7-23: Number of rules generated under data variation

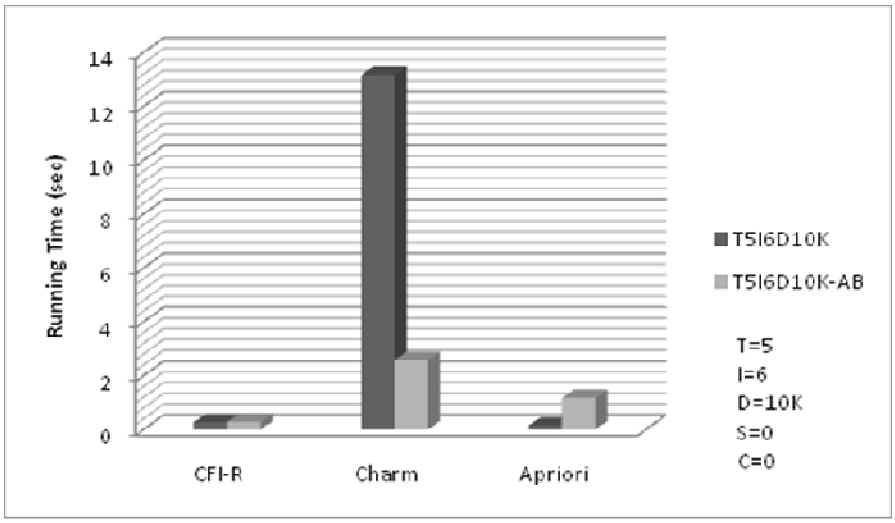

Figure 7-24: Running time under data variation in seconds

	*T*5*I*6*D*10*K*	*T*5*I*6*D*10*K-AB*
CFI-R	0.265997	0.2819332
Charm	13.16552	2.572267
Apriori	0.14	1.17

Table 7-24: Running time under data variation in seconds

7.4 Performance Study for Missing Data Estimation

The performance of our proposed approach, CARM, is studied by means of simulation. Several different simulation experiments are conducted in order to evaluate the proposed technique and compare it with the Average Window Size (AWS) approach, the linear interpolation approach, the linear trend approach, and with the WARM approach, the state-of-the-art data estimation algorithm in sensor databases using 2-frequent itemsets based association mining (Halatchev, 2005). We compared the estimation accuracy, running time and memory space usage when applying different methods to each application dataset.

The first dataset was collected in year 2000 at various locations throughout the city of Austin, Texas. The data represents the current location, the time interval, and the number of vehicles detected during this interval. All sensor nodes report to a single server. The sensors are deployed on city streets, collect and store the number of the vehicles detected for a given time interval. The vehicle counts taken as sensor readings that are used as input for our simulation experiments are traffic data provided by (Austin, 2003).

A second experiment was performed over sensor data collected in the Huntington Botanical Garden in Sam Marino, California (Huntington, 2008). The simulation data of the environmental monitoring application was collected in year 2008 at various locations throughout the sensor network in Huntington Botanical Garden. The data represents the current location, the time interval, and the air temperature of detected environment during this interval. All sensor nodes report to a single server. The sensors are deployed on different places of the botanical garden, collect and store the air temperature detected for a given time interval. The air temperatures are taken as sensor readings that are used as input for our simulation experiment.

7.4.1 Performance Study of Estimation Accuracy

The evaluation of the estimation accuracy of the missing values is done by using the average Root Mean Square Error (RMSE):

$$RMSE = \frac{1}{numStates}\sqrt{\frac{\sum_{i=1}^{\#estimations}(Xa_i - Xe_i)^2}{\#estimations}}$$

where Xa_i and Xe_i are the actual value and the estimated value, respectively; #estimations is the number of estimations performed in a simulation run; and numStates is the number of subsets, in which the actual readings are distributed.

The expression $\sqrt{\frac{\sum_{i=1}^{\#estimations}(Xa_i - Xe_i)^2}{\#estimations}}$ represents the standard error and is an estimate of the standard deviation under the assumption that the errors in the estimated values (i.e.

Xai - *Xei*) are normally distributed. From the definition, we can see the smaller the RMSE, the better the estimation accuracy.

From Figure 7-25 and Table 7-25, we can see that CARM gives the best average estimation result of the above approaches regarding the accuracy, followed by the WARM approach. The linear interpolation, AWS, and linear trend approaches perform no better than WARM and CARM approaches. From Figure 7-25, we can also see that CARM gives the best estimation result on the maximum estimation accuracy, which is the root square error for the maximum difference between the estimated and accurate values.

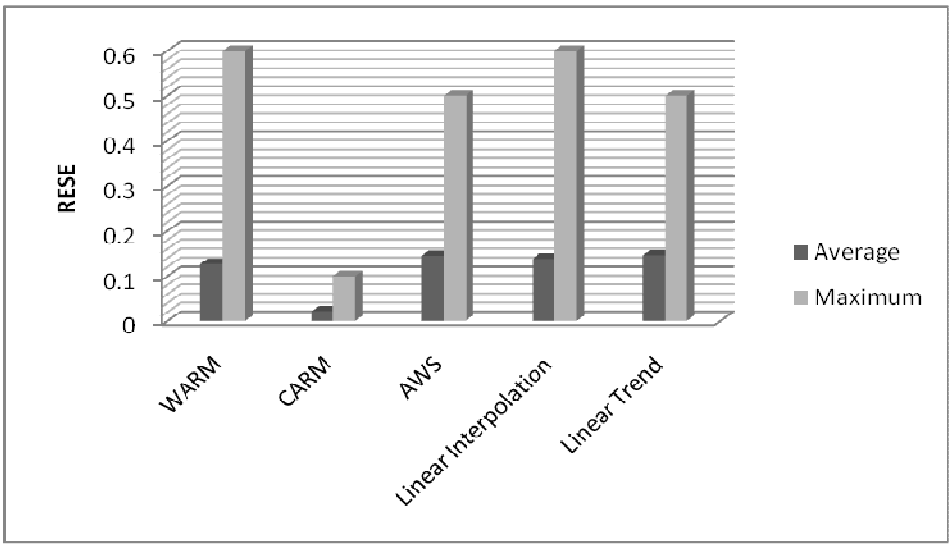

Figure 7-25: Performance study of average and maximum estimation accuracy for traffic monitoring application

	Average	Maximum
WARM	0.1266228	0.6
CARM	0.021517	0.1
AWS	0.144978	0.5
Linear Interpolation	0.138109	0.6
Linear Trend	0.145933	0.5

Table 7-25: Performance study of average and maximum estimation accuracy for traffic monitoring application

From Figure 7-26 and Table 7-26, we can see that CARM gives the best result of the above approaches regarding the estimation accuracy. The linear interpolation, AWS, and linear trend approaches perform no better than CARM approach.

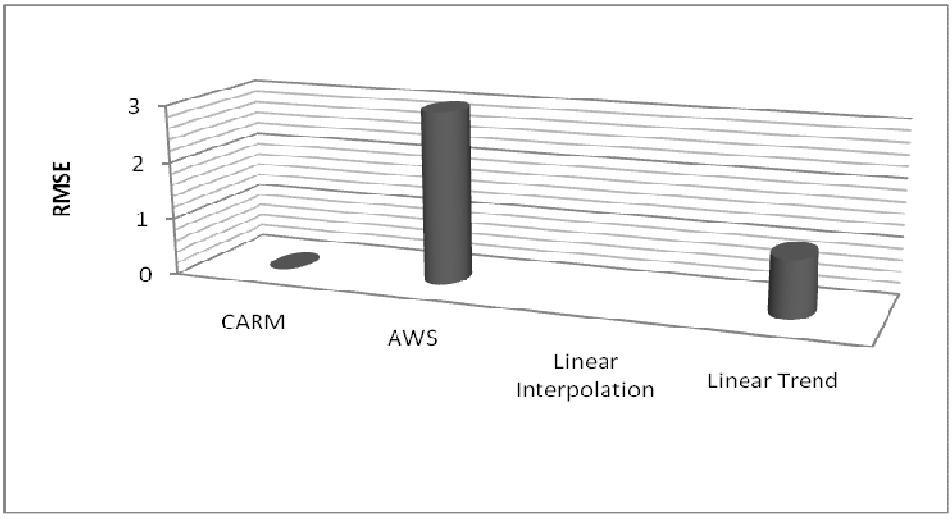

Figure 7-26: Performance study of average estimation accuracy for environmental monitoring application

CARM	0
AWS	3
Linear Interpolation	
Linear Trend	1

Table 7-26: Performance study of average estimation accuracy for environmental monitoring application

7.4.2 Performance Study of Running Time

Figure 7-27 and Table 7-27 illustrate the running time in seconds of AWS, linear interpolation, linear trend, WARM and CARM approaches. The experimental results show that in terms of running time, the WARM and CARM approaches are outperformed by AWS, linear interpolation and linear trend approaches. The CARM approach is faster than the WARM technique.

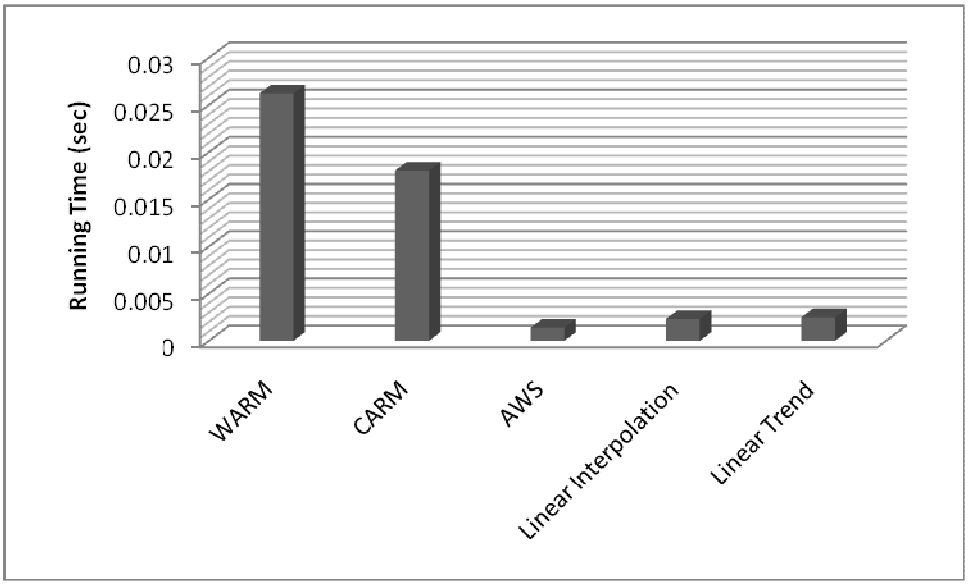

Figure 7-27: Performance study of running time for traffic monitoring application in seconds

WARM	0.026222222
CARM	0.018046296
AWS	0.001388889
Linear Interpolation	0.002314815
Linear Trend	0.0025

Table 7-27: Performance study of running time in seconds for traffic monitoring application in seconds

Figure 7-28 and Table 7-28 illustrate the running time in seconds of AWS, linear interpolation, linear trend, and CARM approaches. The experimental results show that in terms of running time, the CARM approach is outperformed by AWS, linear interpolation and linear trend approaches.

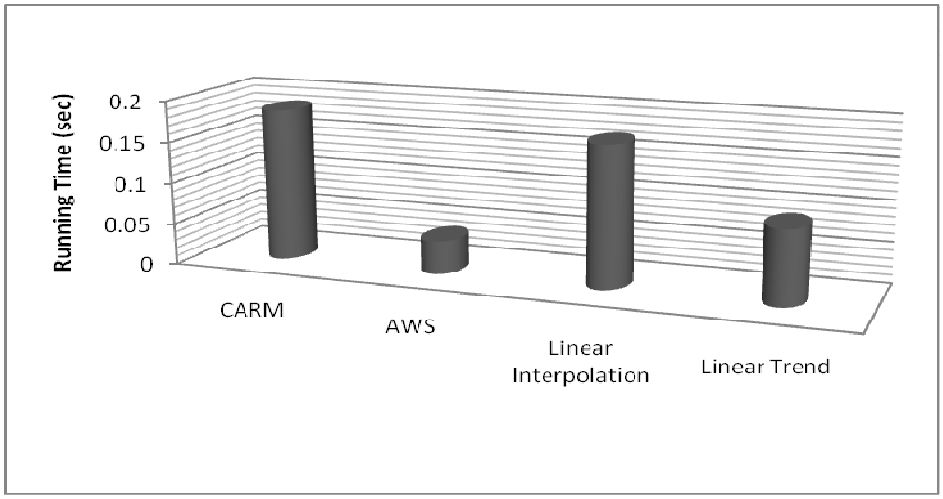

Figure 7-28: Performance study of running time for environmental monitoring application in seconds

CARM	0.185
AWS	0.04
Linear Interpolation	0.17
Linear Trend	0.09

Table 7-28: Performance study of running time for environmental monitoring application in seconds

7.4.3 Performance Study of Memory Usage

Figure 7-29 and Table 7-29 illustrate the memory usage of AWS, linear interpolation, linear trend, WARM and CARM approaches in MB. The experimental results show that in terms of memory space, the WARM approach is outperformed by all the other four approaches. The results of the simulation experiments show that for 108 sensors the needed memory space using WARM is much higher than that using CARM. This is because the DIU data structure uses less memory space than the cube data structures, and it only stores the condensed closed itemsets information.

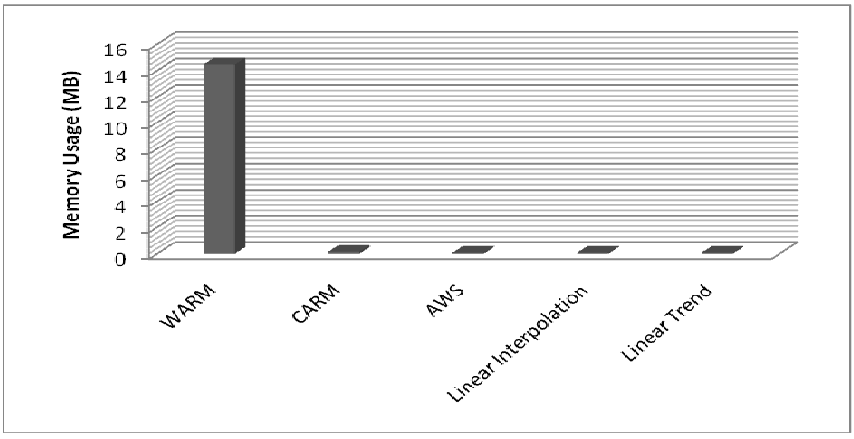

Figure 7-29: Performance study of memory usage for traffic monitoring application in MB

WARM	14.463792
CARM	0.153084
AWS	0.080352
Linear Interpolation	0.080352
Linear Trend	0.080352

Table 7-29: Performance study of memory usage for traffic monitoring application in MB

Figure 7-30 and Table 7-30 illustrate the memory usage of AWS, linear interpolation, linear trend, and CARM approaches in MB. The experimental results show that in terms of memory space, the CARM approach is outperformed by all the other three approaches.

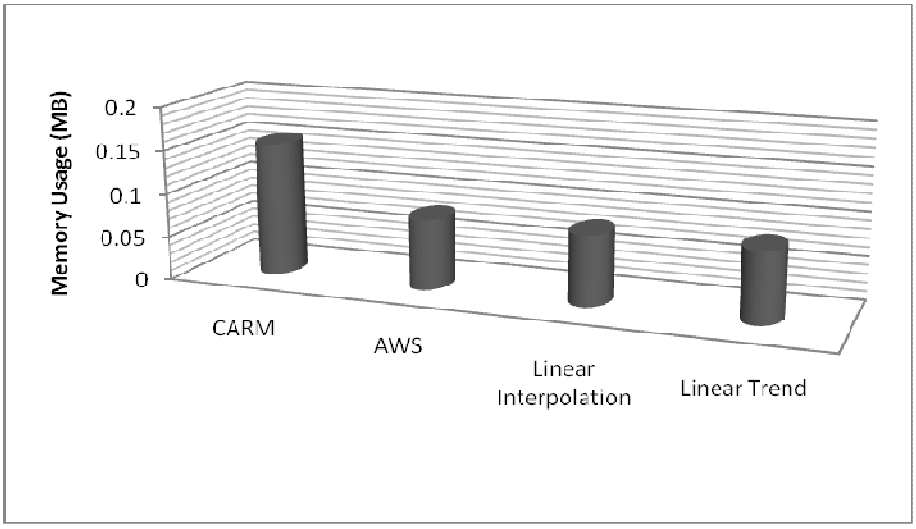

Figure 7-30: Performance study of memory usage for environmental monitoring application in MB

CARM	0.153084
AWS	0.080352
Linear Interpolation	0.080352
Linear Trend	0.080352

Table 7-30: Performance study of memory usage for environmental monitoring application in MB

7.5 Summary

In this chapter we perform different simulation experiments to study the performance of proposed algorithms and comparing them with the state-of-art algorithms in the literature.

The CFI-Stream Algorithm is an incremental method to check and maintain closed itemsets online. It mines and maintains a pool of current closed itemsets in the DIU. The performance study demonstrates the performance advantage of the proposed technique in terms of both computation time and memory usage to mine closed itemsets. Its maintained sets remain the same independent of the support threshold, which could be a disadvantage in application on single user query request with high support threshold, since it’s designed to mine complete information and be able to fulfill multiple support thresholds at the same time.

The performance study of the association mining framework based on closed pattern mining shows that our proposed technique can efficiently produce a minimum set of non-redundant association rules in data streams and thus makes it easier for data analysis. Furthermore, the rules can be generated on demand, at different users' request thresholds, and different input and output patterns. The

proposed association mining framework is especially suitable for a distributed data stream query environment.

Our performance study shows that the application of closed pattern based association mining to estimate missing sensor data online is an area worth to explore. Our designed algorithm CARM is able to estimate missing sensor value with both time and space efficiency, and greatly improves the estimation accuracy.

8 Conclusions and Future Work

In this dissertation, a novel algorithm, CFI-Stream, is developed to perform closure check and discover closed patterns in the current data stream sliding window. The algorithm offers an incremental method to check and maintain closed patterns online. All closed frequent itemsets in data streams can be output in real time based on different users' specified thresholds.

The performance studies show that this algorithm is able to mine data streams online with both time and space efficiency independent of support information, and it can adapt to the concept drift in data streams. Experimental results show that our method can achieve better performance than a representation algorithm for the state-of-the-art approaches in terms of both time and space overhead. In the future, we plan to extend our proposed algorithm to different data streaming applications.

Also, a framework is developed to mine non-redundant and informative associations based on the derived closed itemsets in data streams. The rule generation is based on the current closed itemsets in data streams which are a condensed representation of the stream data. Theoretical analysis and experimental results show that our proposed framework can efficiently produce non-redundant association rules in data streams which provide a minimum set of associations among itemsets in data streams and thus make it easier for data analysis. Furthermore, the association rules can be generated on demand, at different users'

request support and confidence thresholds, and input and output patterns, which is especially suitable for the distributed data stream query environment.

Finally, a novel algorithm, called CARM, is proposed to perform data estimation in sensor network databases based on closed pattern association mining in sensor streams. The algorithm offers an online method to derive association rules based on the discovered closed patterns, and estimates the missing values based on derived associations. It can find out the relationships between multiple sensors not only when they report the same sensor readings but also when they report different sensor readings. Our performance study shows that CARM is able to estimate missing sensor readings online with both time and space efficiency, and greatly improves the estimation accuracy.

There are more future works can be done in this research area. For example, to develop more data mining techniques for stream data, such as clustering, classification, and finding outliers in data streams. Also these derived techniques can be applied to more data streams applications. Some applications have special processing needs, for example, mining the stream sequence, time series in data streams and so on.

Reference

(Agrawal, 1993) Agrawal, R., & Imielinski, T., & Swami, A. (1993). Mining association rules between sets of items in massive databases. *International Conference on Management of Data*.

(Agrawal, 1994) Agrawal R., & Srikant, R. (1994). Fast algorithms for mining association rules. *International Conference on Very Large Databases*.

(Allison, 2002) Allison, P. D. (2002). Missing data. *Thousand Oaks*, CA: Sage.

(Angiulli, 2004) Angiulli, F., & Ianni, G., & Palopoli, L. (2004). On the complexity of inducing categorical and quantitative association rules. *Theoretical Computer Science*. Vol. 314 Issue 1.

(Asada, 1998) Asada, G., & Dong, M., & Lin, T. S., & Newberg, F., & Pottie, G., & Kaiser, W. J., & Marcy, H. O. (1998). Wireless integrated network sensors: Low power systems on a chip. *European Solid State Circuits Conference*.

(Austin, 2003) Austin, F. I. (2000). *Austin Freeway ITS Data Archive*. Retrieved January, 2003 from http://austindata.tamu.edu/default.asp.

(Bayardo, 1999) Bayardo, R. J., & Agrawal, R. (1999). Mining the most interesting rules. *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

(Bastide, 2000) Bastide, Y., & Pasquier, N., & Taouil, R., & Stumme, G., & Lakhal, L. (2000). Mining minimal non-redundant association rules using frequent closed itemsets. *First International Conference on Computational Logic*.

(Cai, 2004) Cai, Y. D., & Pape, G., & Han, J., & Welge, M., & Auvil, L. (2004). MAIDS: Mining alarming incidents from data streams. *International Conference on Management of Data*.

(Chang, 2003) Chang, J. H., & Lee, W. S., & Zhou, A. (2003). Finding recent frequent itemsets adaptively over online data streams. *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

(Chang, 2004) Chang, J. H., Lee, & W. S. (2004). A sliding window method for finding recently frequent itemsets over online data streams. *Journal of Information Science and Engineering*.

(Charikar, 2004) Charikar, M. & Chen, K., & Farach-Colton, M. (2004). Finding frequent items in data streams. *Theoretical Computer Science, Vol. 312, Issue 1*.

(Chi, 2004) Chi, Y., & Wang, H. X., & Yu, P. S., & Muntz R. R. (2004). Moment: Maintaining closed frequent itemsets over a stream

sliding window. *IEEE International Conference on Data Mining*.

(Chi, 2006) Chi, Y., & Wang, H. X., & Yu, P. S., & Muntz R. R. (2006). Catch the moment: maintaining closed frequent itemsets over a stream sliding window. *Knowledge and Information Systems*.

(Cool, 2000) Cool, A. L. (2000). A review of methods for dealing with missing data. *The Annual Meeting of the Southwest Educational Research Association*.

(Dang 2007) Dang, X. H., & Ng, W. K., & Ong, K. L. (2007). Online mining of frequent sets in data streams with error guarantee. *Knowledge and Information Systems*.

(Demaine, 2002) Demaine, E. D., & Ortiz, A. L., & Munro, J. I. (2002). Frequency estimation of internet packet streams with limited space. *European Symposium on Algorithms*.

(Dempster, 1977) Dempster, N. L., & Rubin, D. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*.

(Deshpande, 2005) Deshpande, A., & Guestrin C., & Madden, S. (2005). Using probabilistic models for data management in acquisitional environments. *The Conference on Innovative Data Systems Research*.

(Gelman , 1995) Gelman, J., & Carlin, H. S., & Rubin, D. (1995). Bayesian data analysis. *Chapman & Hall*.

(Guha, 2001) Guha S., & Koudas, N., & Shim, K. (2001). Data streams and histograms. *ACM Symposium on Theory of Computing*.

(Guha, 2002) Guha, S., & Koudas, N. (2002). Approximating a data stream for querying and estimation: Algorithms and performance evaluation. *International Conference on Data Engineering*.

(Giannella, 2003) Giannella, C., & Han, J. W., & Pei, J., & Yan, X. F., & Yu, P. S. (2003). Mining frequent patterns in data streams at multiple time granularities. *Data Mining: Next Generation Challenges and Future Directions*, AAAI/MIT.

(Gruenwald, 2007) Gruenwald, L., & Chok, H, & Aboukhamis, M. (2007). Using data mining to estimate missing sensor data. *IEEE International Conference on Data Mining Workshop.*

(Halatchev, 2005) Halatchev, M., & Gruenwald, L. (2005). Estimating missing values in related sensor data streams. *International Conference on Management of Data*.

(Han, 2000) Han, J. W., & Pei, J., & Yin, Y. W. (2000). Mining frequent patterns without candidate generation. *International Conference on Management of Data*.

(Han, 2001) Han, J. W., & Kamber, M. (2001). *Data mining: Concepts and techniques*. San Francisco: Morgan Kaufmann.

(Huntington, 2008) Huntington Botanical Gardens SensorWare Systems. (2008). *Huntington Botanical Gardens Sensor Web Live Data*. Retrieved November, 2008 from http://caupanga.huntington.org/swim/.

(Iannacchione, 1982) Iannacchione, V. G. (1982). Weighted sequential hot deck imputation macros. *Proceedings of the SAS Users Group International Conference*.

(Inokuchi, 2000) Inokuchi, A., & Washio, T., & Motoda, H. (2000). An Apriori-based algorithm for mining frequent substructures from graph data. *In proceedings of the 4th European Conference on Principles of Data Mining and Knowledge Discovery*.

(Jiang, 2006) Jiang, N., & Gruenwald, L. (2006). CFI-Stream: Mining Closed Frequent Itemsets in Data Streams. *ACM SIGKDD international conference on knowledge discovery and data mining*.

(Jiang, 2007) Jiang, N. & Gruenwald, L. (2007). Estimating missing data in data streams. *the International Conference on Database* Systems for Advanced Applications.

(Jin, 2003) Jin, C. & Qian, W. N., & Sha, C. F., & Yu, J. X., & Zhou, A. Y. (2003). Dynamically maintaining frequent items over a

data stream. *International Conference on Information and Knowledge Management*.

(Kargupta, 2004) Kargupta, H., & Bhargava, R., & Liu, K., & Powers, M., & Blair, P., & Bushra, S., & Dull, J., & Sarkar, K., & Klein, M., & Vasa, M., & Handy, D. (2004). VEDAS: A mobile and distributed data stream mining system for real-time vehicle monitoring. *SIAM International Conference on Data Mining*.

(Karp, 2003) Karp, R. M., & Shenkerm, S., & Papadimitriou, C. H. (2003). A simple algorithm for finding frequent elements in streams and bags. *ACM Transactions on Database Systems, vol. 28, No. 1, pages 51-55.*

(Klementinen, 1994) Klementinen, M., & Mannila, H., & Ronkainen, P., & Toivonen, H., & Verkamo, A. I. (1994). Finding interesting rules from large sets of discovered association rules. *In Third International Conference on Information and Knowledge Management*.

(Koh, 2006) Koh, J. L., & Shin, S. N. (2006). An approximate approach for mining recently frequent itemsets from data streams. *The 8th International Conference on Data Warehousing and Knowledge Discovery*.

(Li, 2004) Li, H. F., & Lee, S. Y., & Shan, M. K. (2004). An efficient algorithm for mining frequent itemsets over the entire history

of data streams. *The International Workshop on Knowledge Discovery in Data Streams*.

(Li, 2004) Li, Y., & Liu, Z. T., & Chen, L., & Cheng, W., & Xie, C.H. (2004). Extracting minimal non-redundant association rules from QCIL. *The 4th International Conference on Computer and Information Technology*.

(Li, 2006) Li, H. F., & Ho, C. C., & Kuo, F. F., & Lee, S. Y. (2006). A new algorithm for maintaining closed frequent itemsets in data streams by incremental updates. *Six IEEE International Conference on Data Mining Workshop*.

(Li, 2008) Li, H. F., & Cheng, H. (2008). Improve frequent closed itemsets mining over data stream with bitmap. *Ninth ACIS International conference on software engineering, artificial intelligence, networking, and parallel/distributed computing*.

(Lin, 2005) Lin, C. H., & Chiu, D. Y., & Wu, Y. H., & Chen, A. L. P. (2005). Mining frequent itemsets from data streams with a time-sensitive sliding window. *SIAM International Conference on Data Mining*.

(Little, 1987) Little, R. J. A., & Rubin, D. B. (1987). *Statistical analysis with missing data*. New York: John Wiley and Sons.

(Liu, 1999) Liu, B., & Hsu, W., & Ma, Y. (1999). Pruning and summarizing the discovered association rules. *In International Conference on Knowledge Discovery and Data Mining*.

(Lucchese, 2004) Lucchese, C., & Orlando, S., & Perego, R. (2004). DCI closed: A fast and memory efficient algorithm to mine frequent closed itemsets. *Workshop on Frequent Itemset Mining Implementations*.

(Lucchese, 2006) Lucchese, C., & Orlando, S., & Perego R. (2006). Fast and Memory Efficient Mining of Frequent Closed Itemsets. *Knowledge and Data Engineering, IEEE Transactions*.

(McLachlan, 1997) McLachlan, G., & Thriyambakam, K. (1997). *The EM algorithm and extensions*. New York: John Wiley & Sons.

(Manku, 2002) Manku, G. S., & Motwani, R. (2002). Approximate frequency counts over data streams. *International Conference on Very Large Databases*.

(Mitchell, 1997) Mitchell, T. (1997). *Machine Learning*. McGraw Hill.

(Mozafari, 2008) Mozafari, B., & Thakkar, H., & Zaniolo, C. (2008). Verifying and mining frequent patterns from large windows over data streams. *IEEE International Conference on Data Engineering*.

(Ng, 1998) Ng, R. T., & Lakshmanan, L. V. S., & Han, J. W., & Pang, A. (1998). Exploratory mining and pruning optimizations of

constrained associations rules, *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

(Papadimitriou, 2005) Papadimitriou, S., & Sun, J., & Faloutsos, C. (2005). Streaming pattern discovery in multiple time-series; *The International Conference on Very Large Databases*.

(Pasquier, 1999) Pasquier, N., & Bastide, Y., & Taouil, R., & Lakhal, L. (1999). Discovering frequent closed itemsets for association rules. *The International Conference on Database Theory*.

(Pei 2000) Pei, J., & Han, J. W., & Mao, R. (2000). Closet: An efficient algorithm for mining frequent closed itemsets. *SIGMOD International Workshop on Data Mining and Knowledge Discovery*.

(Pei 2003) Pei, J., & Han J. W., & Wang, J. (2003). Closet+: Searching for the best strategies for mining frequent closed itemsets *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

(Rubin, 1987) Rubin, D. (1987). *Multiple imputations for nonresponce in surveys*. New York: John Wiley & Sons.

(Rubin, 1996) Rubin, D. (1996). Multiple imputations after 18 years; *Journal of the American Statistical Association*.

(Shafer, 1995) Shafer, J. (1995). Model-Based Imputations of Census Short-Form Items. *In Proceedings of the Annual Research Conference*.

(Shin, 2007) Shin, S. J., & Lee, W. S. (2007). An online interactive method for finding assoication rules data streams. *ACM 16th Conference on Information and Knowledge Management*.

(Tan, 2005) Tan P. N., & Steinbach, M., & Kumar, V. (2005). *Introduction to Data Mining*. Addison Wesley.

(Taouil, 2000) Taouil, R., & Pasquier, N., & Bastide, Y., & Lakhal, L. (2000). Mining bases for association rules using closed sets. *International Conference on Data Engineering*.

(Tarui, 2007) Tarui, J. (2007). Finding a dulicate and a missing item in a stream. *Theory and Applications of Models of Computation*.

(Toivonen, 1995) Toivonen, H., & Klemettinen, M., & Ronkainen, P., & Hatonen, K., & Mannila, H. (1995). Pruning an grouping discovered association rules. *In Mlnet: Familiarisation Workshop on Statistics, Machine Learning and Knowledge Discovery in Databases*.

(Wang, 2003) Wang, H. X., & Fan, W., & Yu, P. S., & Han, J. W. (2003). Mining concept-drifting data streams using ensemble classifiers; *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

(Wijsen, 1998) Wijsen, J., & Meersman, R. (1998). On the complexity of mining quantitative association rules. *Data Mining and Knowledge Discovery*. Vol. 2, Issue 3.

(Wilkinson, 1999) Wilkinson & The APA Task Force on Statistical Inference.

(Yang, 2004) Yang, L., & Sanver, M. (2004). Mining short association rules with one database scan. *International Conference on Information and Knowledge Engineering*.

(Yang, 2004) Yang, G. Z. (2004). The complexity of mining maximal frequent itemsets and maximal frequent patterns. *International Conference on Knowledge Discovery and Data Mining*.

(Yoshio, 2002) Yoshio, N., & Takashi, W., & Tetsuya, Y., & Hiroshi, M., & Akihiro, I. (2002). A Faster Apriori-based Graph Algorithm. *SIG-KBS*.

(Yu, 2004) Yu, J. X., & Chong, Z. H., & Lu, H. J., & Zhou, A. Y. (2004). False positive or false negative: Mining frequent itemsets from high speed transactional data streams. *Intermational Conference on Very Large Databases*; 2004.

(Zaki, 1998) Zaki, M. J., & Ogihara, M. (1998). Theoretical foundations of association rules. *ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*.

(Zaki, 2000) Zaki, M. J. (2000). Generating non-redundant association rules; *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

(Zaki, 2002) Zaki, M. J., Hsiao, C. J. (2002). Charm: an efficient algorithm for closed itemsets mining. *SIAM International Conference on Data Mining*.

(Zaki, 2005) Zaki, M. J., Hsiao, C. J. (2005). Efficient algorithms for mining closed itemsets and their lattice structure. *IEEE Transactions on Knowledge and Data Engineering*.