ASSOCIATION RULE HIDING OVER DATA STREAMS

by

Ufuk Günay

B.Sc. in Computer Engineering, Middle East Technical University, 2004

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Computer Engineering
Boğaziçi University

2007

ACKNOWLEDGEMENTS

I would like to express my special thanks to Prof. Taflan Gündem for his supervision in this MS study and helping during the preparation of this dissertation. I would like to mention his patience; giving me inspiration when I was stuck at dead-ends.

I would like to thank to the committee members Prof. Fikret Gürgen and Assoc. Prof. Birgül Egeli for their contributions to this work.

I have very much benefited from the discussions with my dear friends among whom I would like to mention especially Adem Altun and Mustafa Turan. I thank them for their special interest.

I would like to dedicate this work to my fiancée Gülnur Yılmaz and to my family.

ABSTRACT

ASSOCIATION RULE HIDING OVER DATA STREAMS

This study is about an endeavor towards combining association rule mining over data streams and association rule hiding for traditional databases. Mainly, a system for association rule hiding over data streams will be introduced, and related background will be developed in detail.

Although there are many algorithms, some of which perform very well, developed for both association rule mining over data streams and association rule hiding for traditional databases so far, we have not meet a work on stream association rule hiding, namely a work which combines these two research areas. In this work, we introduce a new system in which we merge these two interesting research areas of association rule mining. We apply our stream association rule hiding algorithm on synthetic data. We also run our algorithm over a template guided XML data. Our performance tests show that proposed system hides association rules for data streams efficiently.

ÖZET

VERİ KATARLARINDA BİRLİKTELİK KURALLARININ SAKLANMASI

Bu çalışma veri katarları için birliktelik kurallarının keşfi ile geleneksel veritabanları için birliktelik kurallarının saklanmasının birleştirilmesine yönelik bir çalışmadır. Temel olarak, veri katarında birliktelik kurallarının saklanmasını sağlayan bir sistem tanıtılacak ve ilgili altyapı detaylı olarak geliştirilecektir.

Bugüne kadar her iki alanla ilgili ve bazıları çok iyi sonuç veren birçok algoritma geliştirilmiş olmasına rağmen, veri katarları için birliktelik kurallarının saklanmasına yönelik bir çalışmayla karşılaşmadık. Bu çalışmada, bu iki ilginç araştırma alanlarını birleştiren yeni bir sistem tanıtılacaktır. Veri katarları için birliktelik kurallarının saklanması algoritmamızı sentetik veri üzerinde denemekteyiz. Aynı zamanda algoritmamızı şablon tabanlı XML verisi için de çalıştırmaktayız. Performans testlermiz göstermiştir ki kurduğumuz sistem veri katarları için birliktelik kurallarını etkin bir şekilde saklamaktadır.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

ARHDS	*association rule hiding over data streams*
ARH	*association rule hiding*
ARM	*association rule mining*
DM	*data mining*
KDD	*knowledge discovery in databases*
PSIF	*potentially sensitive itemset forest*
PSIT	*potentially sensitive itemset tree*
SARM	*stream association rule mining*
TDb	*temporary database*
TDbT	*temporary database tree*

1. INTRODUCTION

1.1. Motivation and Essentials

Data mining (DM), also called Knowledge-Discovery in Databases (KDD), is the process of automatically searching large volumes of data for patterns. Association rule mining, a technique for data mining, discovers elements that co-occur frequently within a dataset consisting of multiple independent selections of elements (such as purchasing transactions), and discovers rules, such as implication or correlation, which relate co-occurring elements. Questions such as "if a customer purchases product X, how likely is he to purchase product Y?" and "What products will a customer buy if he buys products A and E?" are answered by association-finding algorithms. This application of association rule mining is also known as market basket analysis. As with most data mining techniques, the task of association rule mining is to reduce a potentially huge amount of information to a small, understandable set of statistically supported statements.

One of the most interesting research issues in data mining is the data stream mining. A data stream is a massive, open-ended sequence of data elements continuously arriving at a rapid rate. Unique characteristics of data streams can be described as:

- Unbounded size of input data
- Usage of main memory is limited
- Input data can only be handled once
- Fast arrival rate
- The order data arrives can not be controlled
- Analytical results generated by algorithms should be instantly available when requested
- Errors of analytical results should be bounded in a range that can be tolerable.

Due to such large, high-speed and continuous characteristics of streaming data, mining data streams and also association rule mining over data streams are more difficult than mining static databases. The continuous characteristic of streaming data makes it

essential to use the algorithms which require only one scan over the stream for knowledge discovery. The huge nature of stream makes it impossible to store all the data into main memory or even in secondary storage. This motivates the design of summary data structure with small footprints that can support both one-time and continuous queries. In other words, one-pass data stream mining algorithms have to sacrifice the correctness in the analytical results by allowing some counting errors. Consequently, previous multi-pass algorithms studied for mining static datasets are not feasible for mining data streams [20].

Another important and interesting research issue in data mining is association rule hiding. Large amounts of data contain sensitive information that must be protected against unauthorized access. The protection of the confidentiality of this information has been a long-term goal for the database security research communities and for the government statistical agencies. Recent advances in data mining and machine learning algorithms have increased the disclosure risks that one may encounter when releasing data to outside parties [9].

In order to support their short and long term planning activities, many government agencies, businesses and non-profit organizations are searching for a way to collect, store, analyze and report data about individuals, households or businesses. Thus, information systems containing confidential information such as social security numbers, income, credit ratings, type of disease, customer purchases, etc., must be properly protected.

To illustrate the importance of association rule hiding, let us think such a scenario: Let us suppose that, as purchasing directors of our large supermarket chain, we are negotiating an agreement with Drink Company A. They offer their products in reduced price, if we agree to give them access to our database of customer purchases. We accept the deal and Drink Company A starts mining our customer purchases data. By using an association rule mining tool, they find that people who purchase products of Biscuit Company X also purchase Drink B. Drink Company A now runs a coupon marketing campaign saying that “you can get 60 kuruş off Biscuit X with every purchase of a Drink A product”. This campaign cuts heavily into the sales of Drink B, which increases the prices to us, based on the lower sales. During our next negotiation with Drink Company A, we find out that with reduced competition they are unwilling to offer us a low price.

Finally, we start to lose business to our competitors, who were able to negotiate a better deal with Drink B.

The scenario presented above indicates the need to prevent disclosure not only of confidential personal information from summarized or aggregated data, but also to prevent data mining techniques from discovering sensitive knowledge which is not even known by the database owners.

Our work purposes a system called Association Rule Hiding over Data Streams (ARHDS) for combining these two interesting research areas which are: association rule mining over data streams and association rule hiding for traditional databases. First we discover the sensitive association rules from the data stream to hide, then we hide these rules by using an efficient association rule hiding algorithm.

In coming sections we will introduce the algorithm formally and analyze its theoretical aspects. We have tested our system on data which we have generated by using IBM's synthetic data generator. Also we have run our algorithm on a template-guided XML data. By template-guided XML data we mean that the fields of XML data to be mined and hidden are supplied by the user with a template.

To discover the sensitive rules to hide, we use a prefix tree structure. For this purpose we use an approximation algorithm similar to the Landmark model, that is, we are discovering sensitive rules over the entire history of stream data with a small error. Later on we hide the sensitive rules we have found by a fast association rule hiding algorithm.

The use of ARHDS has offered us the possibility to hide sensitive association rules for data streams. We hope the ARHDS will prove to be a useful tool in the future for privacy preserving data mining over streaming data.

1.2. Current Techniques

Many previous studies contributed to the efficient mining of frequent itemsets in streaming data. According to the stream processing model, the research of mining frequent itemsets in data streams can be divided into three categories: *landmark windows*, *sliding windows*, and *damped windows*. In the landmark window model, knowledge discovery is performed based on the values between a specific timestamp called landmark and the present. In the sliding window model, knowledge discovery is performed over a fixed number of recently generated data elements which is the target of data mining. In the damped window model, recent sliding windows are more important than previous ones.

There are some approaches which have been adopted for privacy preserving data mining. Association rule hiding is one of these approaches. Basically hiding association rules is done by either decreasing the support or confidence of a rule.

Although there are many studies either in stream mining and association rule hiding, we have not met a research which combines these two challenging and interesting areas. So here, we are not able to give a current technique which does association rule hiding over data streams to compare with our system ARHDS.

1.3. Main Contributions

When releasing data to outside parties, information systems should take the privacy issues into account. With the improving technology, releasing such data to other parties may be in a streaming fashion in the near future. Thus, a system for association rule hiding over streaming data should be considered.

Possibly our most important contribution is the introduction of such a system; ARHDS. We implemented and had the chance to test our algorithm for ARHDS with different parameter settings. Our system runs efficiently for both raw data and template guided XML data.

1.4. Experimental Setup

For evaluating our work we have generated some synthetic datasets generated by using IBM's data generator [2]. Also by using one of these synthetic datasets, we have generated XML dataset to run our system ARHDS on XML data with a user specified template.

In the experiments, both the synthetic data and the XML data streams are broken into different size of blocks such as 25K, 50K, etc. for simulating the continuous characteristic of streaming data, where 1K denotes 1,000.

We used Microsoft Visual Studio 2005 as the application development environment and Microsoft Visual C# as the programming language.

We have run our system under different parameter settings. Additionally, we compared the execution time of ARHDS on synthetic data and the XML data we generated.

1.5. Organization of the Document

The rest of the document is organized as follows. Section 2 contains all the related literature survey and preliminary offered. First we cover Stream Association Rule Mining and Association Rule Hiding and then we formally present the techniques and algorithms developed for both areas so far.

Section 3 introduces our system Association Rule Hiding over Data Streams (ARHDS). We closely examine ARHDS here and our algorithm is presented step by step in detail. Also a running example is given to explain our system.

Section 4 is devoted to experimental results we obtained by running ARHDS with different parameter settings. Also we run ARHDS on both synthetic data and XML data we generated. The conclusion and discussion sections are in Section 5**.**

2. RELATED RESEARCH AND OTHER PRELIMINARIES

In this section we try to develop the foundation for main topics of interest in our work, and present further the details of our system background necessary for the rest of the this thesis.

In section 2.1, first we give the definition of data mining, and then we make a complete formal introduction into association rule mining. Also, we give brief information about a special kind of association rule mining; stream association rule mining.

In section 2.2, we present an interesting research area for data mining; privacy preserving data mining. We introduce association rule hiding which we will use in our proposed system.

Ultimately, since we develop a system which combines stream association rule mining and association rule hiding in this thesis; techniques and algorithms for both of these interesting research areas are explained in section 2.3.

2.1. Data Mining

Data mining can be defined as the variety of techniques to identify nuggets of information or decision-making knowledge in bodies of data, and extracting these in such a way that they can be put to use in the areas such as decision support, prediction, forecasting and estimation. The data is often voluminous, but as it stands of low value as no direct use can be made of it; it is the hidden information in the data that is useful [10]. In other words, data mining is the extraction of implicit, previously unknown, and potentially useful information from large databases. It is a powerful technology with great potential to help us focus on the most important information in our data warehouses. By data mining, we can predict future trends, behaviors and make proactive, knowledge-driven decisions. Data mining can answer questions that traditionally were too time-consuming to resolve. Consequently, data mining consists of more than collecting and managing data, it also includes analysis and prediction.

Data mining can be used for a variety of purposes in both the private and public sectors. Industries such as banking, insurance, medicine, and retailing commonly use data mining to reduce costs, enhance research, and increase sales. For example, data mining applications can be used in insurance and banking industries to detect fraud and assist in risk assessment (e.g., credit scoring). Using customer data collected over several years, companies can develop models that predict whether a customer is a good credit risk, or whether an accident claim may be guilty and should be investigated more closely. To predict the effectiveness of a procedure or medicine, data mining can also be used by the medical community. Pharmaceutical firms use data mining of chemical compounds and genetic material to help guide research on new treatments for diseases. Retailers can use information collected through close programs (e.g., shoppers' club cards, frequent flyer points, contests) to discover effectiveness of product selection and placement decisions, coupon offers, and which products are often purchased together. Companies such as telephone service providers and music clubs can use data mining to create a "churn analysis," to assess which customers are likely to remain as subscribers and which ones are likely to switch to a competitor.

Data mining can be performed on data which is represented in different forms such as quantitative, textual, or multimedia. Also to examine the data, data mining applications can use a variety of parameters. They include **association** (patterns where one event is connected to another event, such as purchasing a pen and purchasing paper), **classification** (identification of new patterns, such as coincidences between duct tape purchases and plastic sheeting purchases), **clustering** (finding and visually documenting groups of previously unknown facts, such as geographic location and brand preferences), **sequence or path analysis** (patterns where one event leads to another event, such as the birth of a child and purchasing diapers), and **forecasting** (discovering patterns from which one can make reasonable predictions regarding future activities, such as the prediction that people who join an athletic club may take exercise classes.

As our work is based on Association Rule Mining (ARM), in the next subsection, we will introduce the necessary terminology and concepts for ARM on which we later on build our work.

2.1.1. Association Rule Mining (ARM)

Association rule mining finds interesting associations and/or correlation relationships among large set of data items. Association rules show attributes value conditions that occur frequently together in a given dataset. A typical and widely-used example of association rule mining is Market Basket Analysis. For example, data are collected using bar-code scanners in supermarkets. Such 'market basket' databases consist of a large number of transaction records. Each record lists all items bought by a customer on a single purchase transaction. Managers would be interested to know if certain groups of items are consistently purchased together. They could use this data for adjusting store layouts (placing items optimally with respect to each other), for cross-selling, for promotions, for catalog design and to identify customer segments based on buying patterns.

Association rules provide information of this type in the form of "if-then" statements. These rules are computed from the data and, unlike the if-then rules of logic, association rules are probabilistic in nature.

The formal statement of association rule mining problem was firstly stated in [1] by Agrawal. Let I=I_1, I_2, I_3, …… , I_m be a set of m distinct attributes, T be transaction that contains a set of items such that T $\subseteq$ I, D be a database with different transaction records T*s*. A sample database of transactions is shown in Table 2.1. Each row in the table represents a transaction. There are 3 items, and 6 transactions. XY is an itemset, and transaction T_1 supports that itemset.

Table 2.1. Sample Database

Transactions	Items
T_1	XYZ
T_2	XYZ
T_3	XZ
T_4	XY
T_5	X
T_6	XYZ

Association rule mining is to find out association rules that satisfy the predefined minimum support and confidence from a given database [2]. An *association rule* is an implication in the form of A =>B, where A, B $\subset$ I are sets of items called itemsets, and A $\cap$ B =$\varnothing$. A is called antecedent while B is called consequent; the rule means A implies B.

There are two important basic measures for association rules: *support*(*s*) and *confidence*(*c*). Since the database is large and users concern about only those frequent items, usually thresholds of support and confidence are predefined by users to drop those rules that are not so interesting or useful. The two thresholds are called *minimal support* and minimal confidence respectively, additional constraints of interesting rules also can be specified by the users.

Support(*s*) of an association rule is defined as the percentage/fraction of records that contain A $\cup$ B to the total number of records in the database. The count for each item is increased by one every time the item is encountered in different transaction T in database D during the scanning process. It means the support count does not take the quantity of the item into account. From the definition we can see that support of an item is a statistical significance of an association rule. Support(*s*) is calculated by the following formula:

$$Support(AB) = \frac{\text{Support count of } AB}{\text{Total number of transaction in } D}$$

All possible itemsets and their supports obtained from the database in Table 2.1 are listed in Table 2.2. For example, itemset YZ is supported by 3 transactions out of 6, and therefore has 50% support.

Table 2.2. Support of itemsets obtained from the sample database

Itemset	Support
X	100%
Y	66%
Z	66%
XY	66%
XZ	66%
YZ	50%
XYZ	50%

Confidence of an association rule is defined as the percentage/fraction of the number of transactions that contain A⁄B to the total number of records that contain X, where if the percentage exceeds the threshold of confidence an interesting association rule A =>B can be generated. Confidence is a measure of strength of the association rules.

$$Confidence(AB) = \underline{Support(AB)} \ / \ \underline{Support(A)}$$

All possible rules obtained from the sample database and their confidences obtained from Table 2.2 are shown in Table 2.3. For example, Rule YZ => X has support 50%, since it appears in 3 out of 6 transactions in the database and it has confidence 100% since all the transactions that contain YZ also contain X.

Table 2.3. Rules obtained from the sample database

Rules	Confidence	Support
Y => X	100%	66%
Y => Z	75%	50%
Z => X	100%	66%
Z =>Y	75%	50%
Y => XZ	75%	50%
Z => XY	75%	50%
XY => Z	75%	50%
XZ => Y	75%	50%
YZ => X	100%	50%

In our proposed work**,** we try to hide the association rules over streaming data, so general issues of stream association rule mining will be discussed in the next subsection.

2.1.2. Stream Association Rule Mining (SARM)

A data stream is an ordered sequence of items that arrives in timely order. Different from data in traditional static databases, data streams are continuous, unbounded, usually come with high speed and have a data distribution that often changes with time [12]. As the number of applications on mining data streams grows rapidly, there is an increasing need to perform association rule mining on stream data [16].

One example application of data stream association rule mining is to estimate missing data in sensor networks [13]. Another example is to predict frequency estimation of Internet packet streams [7]. In the MAIDS project [3], this technique is used to find alarming incidents from data streams. Association rule mining can also be applied to monitor manufacturing flows [17] to predict failure or generate reports based on web log streams, and so on.

Due to the characteristics of stream data, there are some inherent challenges for stream data association rule mining [16]. First, 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 streams, 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. Furthermore, there is not enough space to store all the stream data for online processing. Therefore, a one scan of data and compact memory usage of the association rule mining technique are necessary. Second, the mining method of data streams needs to adapt to their changing data distribution. 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 coming rate, otherwise data approximation techniques, such as sampling and load shedding, need to be applied which will decrease the accuracy of the mining results. Fourth, due to the continuous, high speed, and changing data distribution characteristics, the analysis results of data streams often keep changing as well. Therefore, mining of data streams should be an incremental process to keep up with the highly update rate, i.e. new iterations of mining results are built based on old mining results so that the results will not have to be recalculated each time a user's request is received. Fifth, owing to the unlimited amount of stream data and limited system resources, such as memory space and CPU power, a mining mechanism that adapts itself to available resources is needed; otherwise, the accuracy of the mining results will be decreased.

Since we use association rule hiding in our proposed work, in the next subsection, we will introduce privacy preserving data mining and association rule hiding.

2.2. Privacy Preserving Data Mining

Recent advances in data collection, data dissemination and related technologies have inaugurated a new era of research where existing data mining algorithms should be reconsidered from a different point of view, this of privacy preservation. It is well documented that this new without limits explosion of new information through the Internet and other media, has reached to a point where threats against the privacy are very common on a daily basis and they deserve serious thinking [23].

Privacy preserving data mining, is a novel research direction in data mining and statistical databases, where data mining algorithms are analyzed for the side-effects they incur in data privacy. The main consideration in privacy preserving data mining is two

fold. First, sensitive raw data like identifiers, names, addresses and the like should be modified or trimmed out from the original database, in order for the recipient of the data not to be able to compromise another person's privacy. Second, sensitive knowledge which can be mined from a database by using data mining algorithms should also be excluded, because such knowledge can equally well compromise data privacy, as we will indicate. The main objective in privacy preserving data mining is to develop algorithms for modifying the original data in some way, so that the private data and private knowledge remain private even after the mining process. The problem that arises when confidential information can be derived from released data by unauthorized users is also commonly called the "database inference" problem.

There are many approaches which have been adopted for privacy preserving data mining. [23] classifies them based on the following dimensions: data distribution, data modification, and data mining algorithm, data or rule hiding, privacy preservation. Since we used association rule hiding in our work, we will introduce association rule hiding and its usage in real life in the next subsections.

2.2.1. Association Rule Hiding (ARH)

Association rule mining, which was introduced in Section 2.1.1, scans the database of transactions and calculate the support and confidence of the candidate rules to determine if they are significant or not. A rule is significant if its support and confidence is higher than the user specified minimum support and minimum confidence threshold. In this way, algorithms do not retrieve all the association rules that may be derivable from a database, but only a very small subset that satisfies the minimum support and minimum confidence requirements set by the users. An association rule-mining algorithm works as follows. It finds all the sets of items that appear frequently enough to be considered relevant and then it derives from them the association rules that are strong enough to be considered interesting. Preventing some of those rules, that are referred to as "sensitive rules", from being disclosed is aimed. The problem can be stated as follows:

Given a database D, a set R of relevant rules that are mined from D and a subset R_H of R, how can we transform D into a database D_0 *in* such a way that the rules in R can still be mined, except for the rules in R_H? [9].

In [8] the authors demonstrate that solving this problem by reducing the support of the large itemsets via removing items from transactions (also referred to as "sanitization" problem) is an NP-hard problem. Thus, we should look for a transformation of D (the source database) in D_0 (the released database) that *maximizes* the number of rules in R - R_H that can still be mined.

To hide a set R_H of rules (i.e., prevent them from being discovered by association rule mining algorithms), there are two main approaches that can be adopted: First, we can prevent the rules in R_H from being generated, by hiding the frequent sets from which they are derived. Second, we can reduce the confidence of the sensitive rules, by bringing it below a user-specified threshold (*min conf*).

In this thesis, we try to hide sensitive association rules by decreasing the support of each rule. We do association rule hiding for both raw data and XML data. Detailed information about XML is given in Appendix A.

2.2.2. ARH in e-commerce

Association rule mining techniques have been widely used in various applications such as marketing, modern business, medical analysis and many other applications. In e-commerce for instance, a modern company can understand the behavior of its customers, support decision making and gain an overall significant benefit over its rivals using association rule mining. Thus, some association rule mining algorithms have been developed especially for handling transactional data in e-commerce.

In spite of its benefits in all of these applications, association rule mining can also have a threat to privacy and information security if not done or used properly. There are a number of realistic scenarios in which privacy and security issues in association rule mining arise. We describe three challenging e-commerce scenarios as follows:

Scenario 1: Firstly, consider a supermarket and two drink suppliers X and Y. If the transaction database of the supermarket is released, X (or Y) can mine the association rules related to his/her drinks and apply the rules to the sales promotion and the goods supply. As a result, a supplier is willing to exchange a lower price of goods for the database with the supermarket. From this aspect, it is good for the supermarket to release the database. However, the conclusion can be opposite if a supplier uses the mining methods in a different way. For instance, if X finds the association rules related to Y's drinks, saying that most customers who buy diapers also buy Y's drinks, he/she can run a coupon that gives a 10 percent discount when buying X's drinks together with diapers. Gradually, the amount of sales on Y's beers is down and Y cannot give a low price to the supermarket as before. Finally, X gets the control of the drink market and is unwilling to give a low price to the supermarket as before. From this aspect, releasing the database is bad for the supermarket. Therefore, for the supermarket, an effective way to release the database with sensitive rules hidden is required.

Scenario 2: Secondly, suppose that two or more companies have huge dataset records of their customers' buying activities. These companies decide to cooperatively conduct association rule mining on their datasets for their mutual benefit since this cooperation supplies them an advantage over other competitors. However, some of these companies may not want to share some strategic patterns hidden within their own data (also called sensitive association rules) with the other parties. They would like to transform their data in such a way that these sensitive association rules cannot be discovered.

Scenario 3: Lastly, suppose we have a server and many clients in which each client has a set of sold items (e.g. books, clothes, movies, etc). The clients want the server to collect statistical information about associations among items in order to provide recommendations to the clients. On the other hand, the clients do not want the server to know some restrictive association rules. In this context, the clients represent companies and the server is a recommendation system for an e-commerce application, for example, fruit of the client's collaboration. In the absence of rating, which is used in collaborative filtering for automatic recommendation building, association rules can be effectively used to build models for on-line recommendation. When a client sends its frequent itemsets or

association rules to the server, it sanitizes some sensitive itemsets according to some specific policies. The server then gathers statistical information from the sanitized itemsets and recovers from them the actual associations.

In all of these scenarios we should ask ourselves the following question: “How can we get rational data mining results that will allow for correct decision making while preventing the disclosure of sensitive information?” In other words, “Is it possible for us to benefit from the collaborations in the scenarios above by sharing our data while still preserving some sensitive association rules?” In our proposed system, we tried to answer these questions. As generating large-scale real-time data has never been easier by e-commerce, we developed our system for the collaborations at which data is sent in a streaming fashion.

2.3. SARM Techniques and Algorithms

In this section, we try to give detailed information about the techniques and algorithms for both stream association rule mining and association rule hiding as we combine these two in our work.

2.3.1. Model of Data Processing

Model of data processing is an important issue in stream association rule mining. According to [28], there are three stream data processing models, Landmark, Damped and Sliding Windows.

The Landmark model mines all frequent itemsets over the entire history of stream data from a specific time point called landmark to the present. If people are interested only in the most recent information of the data streams, this model is not suitable. For example, in the stock monitoring systems, current and real time information and results will be more meaningful to the end users.

The Damped model, also called the Time-Fading model, mines frequent itemsets in stream data in which each transaction has a weight and this weight decreases with age. So,

older transactions contribute less weight toward itemset frequencies. In this model, different weights for new and old transactions are considered. Damped model can be useful and suitable for applications in which old data has an effect on the mining results, but the effect decreases as time goes on.

The Sliding Windows model finds and maintains frequent itemsets in sliding windows. When the data flows in, only part of the data streams within the sliding window are stored and processed at the time. The size of the sliding window may change according to applications and system resources. The mining result of the sliding window method completely depends on recently generated transactions in the range of the window. All the transactions in the window need to be maintained in order to remove their effects on the current mining results when they are out of range of the sliding window.

In current research on data streams mining, all these three models have been used. Deciding which kind of data process models to use largely depends on application needs. We can convert a Landmark model based algorithm to that using the Damped model by adding a decay function on the upcoming data streams. We can also convert it to that using Sliding Windows by keeping track of and processing data within a specified sliding window.

In our proposed system, since we are interested in hiding the sensitive itemsets and sending the hidden database to the other side, our data processing model finds and maintains sensitive itemsets until we send the data to the receiver. In other words; the algorithm we developed for finding sensitive itemsets is similar to Landmark model.

2.3.2. Data Structure

In stream association rule mining, an efficient and compact data structure is needed to store, update and retrieve the collected information. This is due to bounded memory size and huge amounts of data streams coming continuously. Failure in developing such a data structure will largely decrease the efficiency of the mining algorithm because, even if we store the information in disks, the additional I/O operations will increase the processing time. The data structure needs to be incrementally maintained since it is not possible to

rescan the entire input due to the huge amount of data and requirement of rapid online querying speed. A proper data structure is a crucial part of an efficient algorithm since it is directly associated with the way we handle newly arrived information and update old stored information. A small and compact data structure which is efficient in inserting, retrieving and updating information is most favorable when developing an algorithm to mine association rules for stream data [16].

In [21], a lattice data structure is used to store itemsets, approximate frequencies of itemsets, and maximum possible errors in the approximate frequencies. In [11], a FP-tree is constructed to store items, support information and node links. In [19], the authors employ a prefix tree data structure to store item ids and their support values, block ids, head and node links pointing to the root or a certain node. In our work, we also develop a prefix tree data structure for finding sensitive itemsets before hiding them.

2.3.3. Techniques and Algorithms

Traditional association rule mining algorithms are developed to work on static data and, thus, can not be applied directly to mine association rules in stream data. The first recognized frequent itemsets mining algorithm for traditional databases is Apriori [1]. After that, many other algorithms based on the ideas of Apriori were developed for performance improvement [2], [14]. Apriori-based algorithms require multiple scans of the original database, which leads to high CPU and I/O costs. Therefore, they are not suitable for a data stream environment, in which data can be scanned only once. Another category of association rule mining algorithms for traditional databases proposed by [15] are those using a frequent pattern tree (FPtree) data structure and an FP-growth algorithm which allows mining of frequent itemsets without generating candidate itemsets. Compared with Apriori-based algorithms, it achieves higher performance by avoiding iterative candidate generations. However, it still can not be used to mine association rules in data streams since the construction of FP-tree requires two scans of data.

In stream association rule mining, to choose the right type of mining algorithms is a fundamental issue. Association rules can be found in two steps: First we find large itemsets

(support is ≥ user specified support) for a given threshold support, then we generate desired association rules for a given confidence.

There exist a number of techniques for finding frequent itemsets in data streams. Based on the result sets produced, we can categorize data stream mining algorithms as exact algorithms or approximate algorithms.

In exact algorithms, the result sets consist of all of the itemsets the support values of which are greater than or equal to the threshold support. In [18] and [26], the authors use the exact algorithms to generate the result frequent itemsets. Although it is important for many applications to know the exact answers of the mining results; additional cost is needed to generate the accurate result set when the processing data is huge and continuous. The technique proposed in [18] takes two scans to generate the exact result set, and in [26], the algorithm generated can only mine short itemsets, which cannot be applied to large itemsets. Another option to get the exact mining results with relatively small memory usage is to store and maintain only special frequent itemsets, such as closed or maximal frequent itemsets, in memory. In [6] and [22], the authors proposed algorithms to maintain only closed frequent itemsets and maximal frequent itemsets over a sliding window and landmark processing model, respectively. In both of these cases, how we can get all the information to further generate association rules based on these special itemsets is an additional issue that needs to be considered.

Approximate algorithms generate approximate result sets with or without an error guarantee. Approximate mining frequent patterns with a probabilistic guarantee can take two possible approaches: false positive oriented and false negative oriented. The former includes some infrequent patterns in the result sets, whereas the latter misses some frequent patterns [27].

From the above discussions, we can see that when designing a stream data association rule mining algorithm, we need to answer a number of questions: which kind of algorithm we should use to perform association rule mining in data streams; exact or approximate? Can we guarantee its error if it is an approximate algorithm? How can we reduce and guarantee the error? Is data processed within one pass? What is the tradeoff

between accuracy and processing speed? Can we handle a large amount of data by this algorithm? Up to how many frequent itemsets can this algorithm mine?

In the current works published in this area [18], [26], [6] and [22] proposed exact algorithms, while [19], [27], [4], [21], [5] and [11] proposed approximate algorithms. Among them [27] uses the false negative method to mine association rules, while the other approximate algorithms use the false positive method.

In this work, as we have to keep data until we send (after hiding the sensitive association rules), similar to [19] we have employed a prefix tree data structure to store the data. That is to say, we have used an approximate algorithm for proposed system.

2.4. ARH Techniques and Algorithms

To hide a set R_H of rules, [9] propose five strategies to hide rules using both the approaches. Since we use one of these strategies to hide sensitive rules over data streams in our work, it will be beneficial to give some detailed information about these strategies.

Definition: Given a transaction *t* and an itemset *S*, we say that *t fully supports S* if the values of the items of *S* in *t.values of items* are all 1; *t* is said to *partially support S* if the values of the items of *S* in *t.values of items* are not all 1's. For example, if *S* = {*A; B; C}* = [1110] and p =< *T*1*;* [1010]*;* 2 >, q =< *T*2*;* [1110]*;* 3 > then we would say that q fully supports S while p partially supports S [9].

The hiding strategies that they propose heavily depend on finding transactions that fully or partially support the generating itemsets of a rule. The reason for this is that if we want to hide a rule, we need to change the support of some part of the rule (i.e., decrease the support of the generating itemset). Another issue is that the changes in the database introduced by the hiding process should be limited, in such a way that the information loss incurred by the process is minimal. According to this, they try to apply minimal changes in the database at every step of the hiding algorithms that they propose.

The decrease in the support of an itemset S can be done by selecting a transaction *t*, that supports S and by setting to 0 at least one of the non-zero values of *t.values of items* that represent items in S. Also the increase in the support of an itemset S can be accomplished by selecting a transaction *t* that partially supports it and setting to 1 the values of all the items of S in *t.values of items*.

In order to be able to identify some viable ways for reducing either the support or the confidence of a rule, we need to analyze the formulas that we have already presented for the confidence and the support. Both the confidence and the support are expressed as ratios of supports of itemsets that support the two parts of a rule or its generating itemset. In this way, if we want to lower the value of a ratio, we can decrease the numerator while keeping the denominator fixed or increase the denominator while keeping the numerator fixed.

First we consider which one of the above options can be adopted for decreasing the support of a rule X=>Y. Suppose that the support of this rule is (XUY / N) * 100 where N is the number of transactions in *D*. Since N is constant, this means that we can only change the numerator. For this reason we can only adopt option (a) from above, which is to decrease the numerator (support) of the generating itemset of the rule.

Following we need to investigate which of the three options can be adopted for decreasing the confidence of a rule X=>Y. The confidence of this rule is written as is (XUY / X) * 100. In order to see whether any of the options that were mentioned previously can be applied to this case or not, we need to investigate if any of the pairs of the actions stated in each option are conflicting, based on the relationship between the numerator and the denominator in the confidence ratio. Option (a) implies that we need to decrease the numerator (which is the support) of the generating itemset of the rule, while the support of the itemset in the left hand side of the rule remains fixed. In order to do that, we can decrease the support of the generating itemset of the entire rule by modifying the transactions that support this itemset, making sure that we hide items from the consequent or the right hand side of the rule. This will decrease the support of the generating itemset of the rule, while it will leave unchanged the support of the left hand side or else the denominator.

Option (b) implies that we need to increase the denominator (which is the support of the itemset in the antecedent) of the rule, while the support of the generating itemset of the rule remains fixed. This option is also applicable, since we can increase the support of the rule antecedent while keeping the support of the generating itemset fixed by modifying the transactions that partially support the itemset in the antecedent of the rule but do not fully support the itemset in the consequent [9].

The strategies that [9] developed, based on the observations below, can be summarized as follows: Given a rule X=>Y on a database *D*, the support of the rule in *D* expresses the probability to find transactions containing all the items in XUY. The confidence of X=>Y is, instead, the probability to find transactions containing all the items in XUY (e.g. supporting XUY) once we know that they contain X. This suggests another way to express the confidence of a rule, based on support: Conf(X=>Y) = (Supp (XUY) / Supp (X)) * 100, where Supp (XUY) is the support of the rule and Supp(X) is the support of the antecedent of the rule.

We can decrease the confidence of the rule in two ways: We increase the support of the rule antecedent X through transactions that partially support it or we decrease the support of the rule consequent Y in transactions that support both X and Y. Also we can decrease the support of the rule by decreasing the support of either the rule antecedent X, or the rule consequent Y, through transactions that fully support the rule.

In proposed system, we decrease the support of rules to hide sensitive itemsets over data streams.

3. PROPOSED SYSTEM: ARHDS

In this section, we will introduce our proposed system: "Association Rule Hiding over Streaming Data".

3.1. Problem Definition

In ARDHS, we propose a single-pass algorithm for hiding all sensitive itemsets in data streams similar to the landmark windows model when a user-specified minimum support threshold $ms \in (0, 1)$, and a user-defined error threshold $\varepsilon \in (0, ms)$ for data pruning phase are given. For hiding all sensitive itemsets, a data stream can be defined as follows.

Let $I = \{I_1, I_2, I_{3...} I_m\}$ be a set of literals, called *items*. Let *data stream* $DS = B_1, B_2, B_{3...} B_N\ldots$ be an infinite sequence of *blocks*, where each block is associated with a block identifier i, and N is the identifier of the "*latest*" block B_N. Each block B_i consists of a timestamp ts_i, and a set of transactions; that is, $B_i = [ts_i, T_1, T_2, T_{3...} T_k]$, where $k \geq 0$. Hence, the *current length* (*CL*) of the data stream is defined as $CL = |B1|+|B2|+\ldots+|BN|$. A *transaction T* consists of a set of items, such that $T \subseteq I$. Moreover, each transaction is also associated with a unique transaction identifier, called *TID*. A set of items X is also called an *itemset* and an itemset X with k items is denoted as $(x_1, x_2, x_{3...} x_k)$, such that $X \subseteq I$.

Due to the unique characteristics of streaming data, any one-pass algorithms have to sacrifice the correctness of their analysis results by allowing some errors. Hence, the *True support* of an itemset X, denoted by $Tsup(X)$, is the number of transactions seen so far in which that itemsets occurs as a subset, and the *Estimated support* of the itemset X, denoted as $Esup(X)$, is the *estimated true support* of X stored in the summary data structure constructed by the one-scan approaches, where $Esup(X) \leq Tsup(X)$. An itemset X is called a *sensitive* itemset if $Tsup(X) \geq ms*CL$. An itemset is called an *insensitive* itemset if $\varepsilon*CL > Tsup(X)$.

Hence, given a user-defined minimum support threshold $ms \in (0, 1)$, a user-specified error threshold $\varepsilon \in (0, ms)$ and a data stream *DS*, our goal is to develop a single-pass algorithm to hide all sensitive itemsets similar to the landmark windows model of the streaming data using as little main memory as possible.

3.2. Assumptions

In our proposed system ARDHS, we have the four assumptions: First, we assume that arriving items in a transaction or an itemset, are sorted in lexicographic order. Second, the average size of each block of data stream is a constant value *k* for simplicity, that is, each block contains k transactions. Third, we hide only rules that are supported by disjoint large itemsets: If we try to hide overlapping rules, then hiding a rule may have side effects on the other rules to be hidden. This increases the time complexity of our algorithm since hiding a rule may cause an already hidden rule to haunt back. Therefore the algorithms should reconsider previously hidden rules and hide them back if they are no longer hidden. Fourth, we hide one rule at a time: Hiding one rule must be considered as an atomic operation. This is actually related to the third assumption. Since the rules to be hidden are assumed to be disjoint, the items chosen for hiding a rule will also be different for different rules. Therefore, hiding a rule will not have a side effect on the rest of the rules. So, considering the rules one at a time or all together will not make any difference.

3.3. ARDHS Algorithm

Our algorithm for our system ARHDS consists of six steps. After describing each step in detail, we will give a running example to explain our system.

Step 1: *Reading a block of transactions*: In the first step we read the data stream block by block.

Step 2: *Constructing the Potentially Sensitive Itemset Forest (PSIF) and the Temporary Database (TDb) for the block*: PSIF consists of Potentially Sensitive Itemset Trees (PSIT) of item suffixes. Similarly TDb consists of *Temporary Database Trees*

(TDbT) where the root of each tree is the first item in the transaction. Both PSIT and TDbT have the same structure which is as follows:

Each node in the tree consists of four fields: *itemName, support, ChildTrees and parentTree* where *itemName* is the name of the node, *support* records the number of transactions containing the item, *ChildTrees* is a hash table for a faster reach to its children trees and *parentTree* is the parent tree of the item. In addition to these properties, each root node has a Hash Table (HT) for its children nodes. Whereas the key for HT is the children item id, the value of HT consists of two fields: Number of occurrence of children items in the tree and the pointers to these occurrences.

For XML data, we have the same structure defined above. Additionally, at each leaf node, we store the transaction number for each transaction.

The construction of PSIF and TDb can be described as follows: Let a transaction to be inserted to the system be $T = (x_1, x_2, x_{3...} x_k)$, in the current block. Whereas T is directly inserted into TDb, before it is inserted into PSIF, transaction T is converted into k small transactions; that is, $(x_1, x_2, x_{3...} x_k)$, $(x_2, x_{3...} x_k)\ldots (x_{k-1,} x_k)$ and (x_k).

Step 3: *Pruning the insensitive itemsets from PSIF*: Provided an error threshold $\varepsilon \in (0, ms)$ by the user, PSIF is pruned from the insensitive items. Until we start hiding process, we go back to step 1.

Step 4: *Finding sensitive disjoint itemsets from PSIF to hide*: To hide all sensitive association rules we develop a heuristic as follows: Given a minimum support threshold $ms \in (0, 1)$ provided by the user, first we find all sensitive itemsets, then we sort these sensitive itemsets according to their support. Then, beginning from the sensitive itemset with the highest support, we discover the *sensitive* disjoint itemsets to hide. We hide sensitive rules from TDb with Step 5, and then we come back to Step 4 to check if there still exists any sensitive disjoint itemsets. We follow this strategy until we hide all sensitive association rules from TDb.

Step 5: *Hiding sensitive disjoint itemsets and updating TDb*: After determining the sensitive disjoint itemsets, we hide them by developing one of the strategies purposed by [9]. In their work, they present five different algorithms for association rule hiding. Here, we use the fastest of these algorithms since we are trying to hide association rules over data streams. Also with this algorithm no new rule is introduced. The algorithm is as follows:

To hide sensitive rules, we decrease the support of their generating itemsets until the support is below the minimum support threshold. If there are more than one large itemsets to hide, we first sort the large itemsets with respect to their size and support. Formally, let the next itemset to be hidden be *Z*. We hide *Z* from Database D, by removing the items in *Z*, from the transactions in T_Z, in round robin fashion. We start with a random order of items in *Z* and a random order of transactions in T_Z. Assume that the order of items in *Z* be $i_0, i_1 \ldots i_{n-1}$ and let the order of transactions in T_Z be $t_0, t_1 \ldots t_{n-1}$ .In step 0 of the algorithm, the item i_0 is removed from t_0. At step 1, i_1 is removed from t_1, and in general, at step *k*, item $i_{(k \bmod n)}$ is removed from transaction t_k. The execution stops after the support of the current itemset, to be hidden, goes below the minimum support threshold.

Here we develop this algorithm to hide all sensitive itemsets. Our contribution is that when we update the database D, we also update our PSIF structure to go back to Step 4 to check if there still sensitive itemsets exist.

An overview of this algorithm is depicted in Figure 3.1.

```
INPUT: a set L of large itemsets, the set L_H of large itemsets to hide, the
database D and the minimum support threshold ms ∈ (0, 1).
OUTPUT: the database D modified by the deletion of the large itemsets in
L_H
Begin
1. Sort L_H in descending order of size and support of the large itemsets
foreach Z in L_H
{
    2. Sort the transactions in T_Z
       in ascending order of transaction size
    3. N iterations = |T_Z| - max_supp *|D|
     For k = 1 to N_iterations do
    {
        4. Remove the maximal support item
           of Z from the next transaction in T_Z
        5. Update the database, D
        6. Update PSIF
    }
}
End
```

Figure 3.1. Pseudocode for Hiding Sensitive Itemsets

The intuition behind the idea of hiding in round robin fashion is fairness. This way, no item is over-killed and the chance of having a smaller number of side effects is higher than choosing an item at random and always trying to hide it.

Step 6: *Sending the updated TDb (the block)*: After hiding all sensitive items from TDb, the hidden TDb is sent to the receiver.

Now let us examine two simple examples to explain our proposed system. In the first example we will inspect our algorithm step by step for raw data. In the second example we will give only the structure of TDb for XML data, since others steps are same.

Example 1: Let us suppose that the data stream consists of two blocks each having five transactions. Let the first block B_1 of the data stream be (*acdef*), (*df*), (*abe*), (*acdf*),

(*cef*), the second block B_2 be (*bef*), (*bdg*), (*def*), (*bg*), (*ceg*). Let ε = 25% be the error threshold for pruning and ms = 30% be the minimum support for hiding where *a*, *b*, *c*, *d*, *e*, *f,g* are items in the stream.

Step 1: We read the first block B_1 from the data stream.

Step 2: 1) *First transaction acdef*: ARHDS reads the first transaction *acdef,* inserts item-suffix transactions *acdef*, *cdef*, *def*, *ef, f* into PSIF and *acdef* into TDb. Results are shown in Figure 3.2. Here, item name and support of each PSIT (Potentially Sensitive Itemset Trees) are presented. Also for each PSIT, their HT (Hash Table) for its children nodes is shown. In the following steps, we omit the *pointers* to the occurrences of each children node for concise representation.

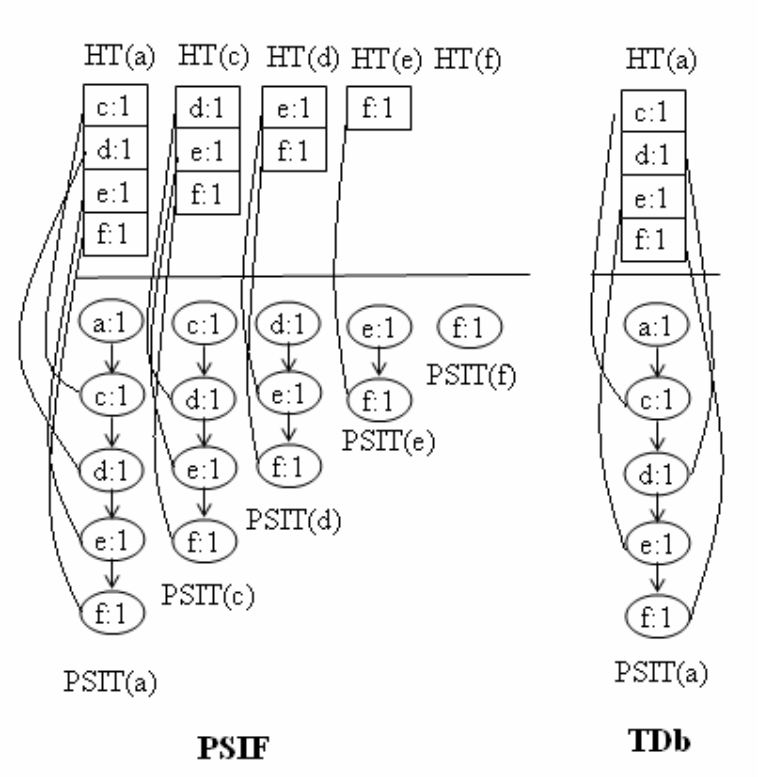

Figure 3.2. PSIF and TDb after inserting the first transaction *acdef*

2) *Second transaction df*: ARHDS reads the second transaction *df,* inserts item-suffix transactions *df*, *f* into PSIF and *df* into TDb. Results are shown in Figure 3.3.

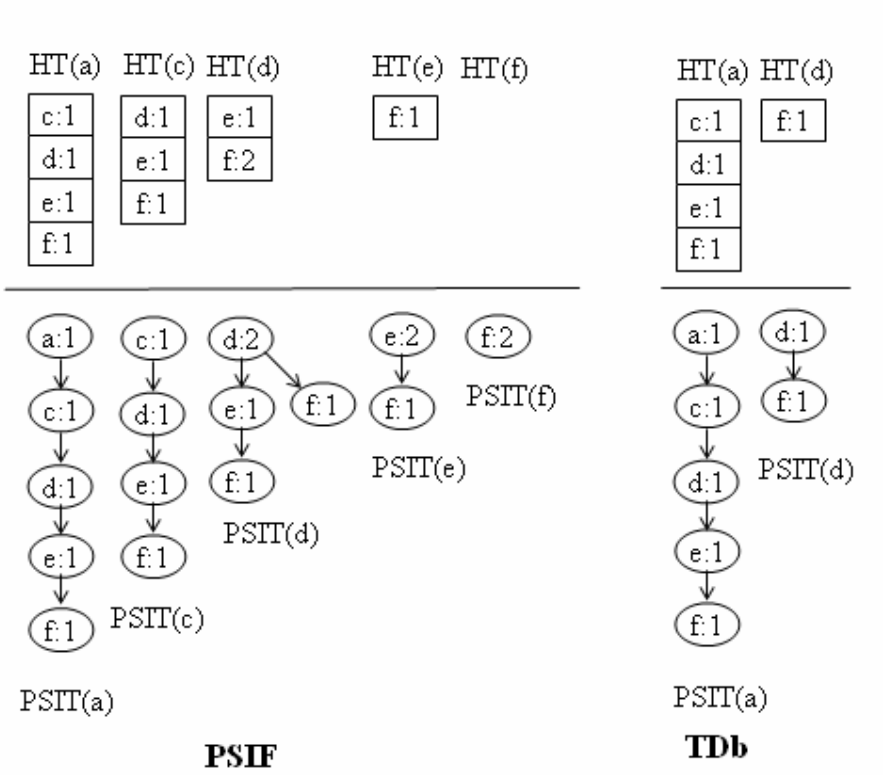

Figure 3.3. PSIF and TDb after inserting the second transaction *df*

3) *Third transaction abe*: ARHDS reads the third transaction *abe,* inserts item-suffix transactions *abe, be, e* into PSIF and *abe* into TDb. Results are shown in Figure 3.4.

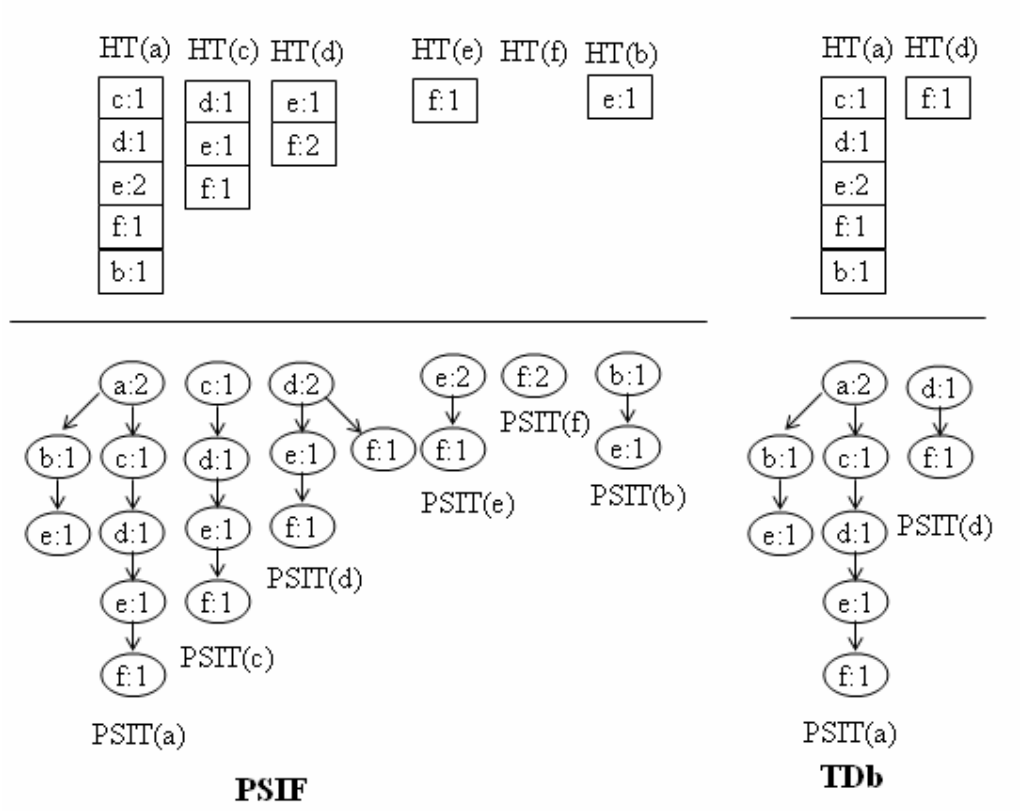

Figure 3.4. PSIF and TDb after inserting the third transaction *abe*

4) *Fourth transaction acdf*: ARHDS reads the fourth transaction *acdf,* inserts item-suffix transactions *acdf, cdf, df, f* into PSIF and *acdf* into TDb. Results are shown in Figure 3.5.

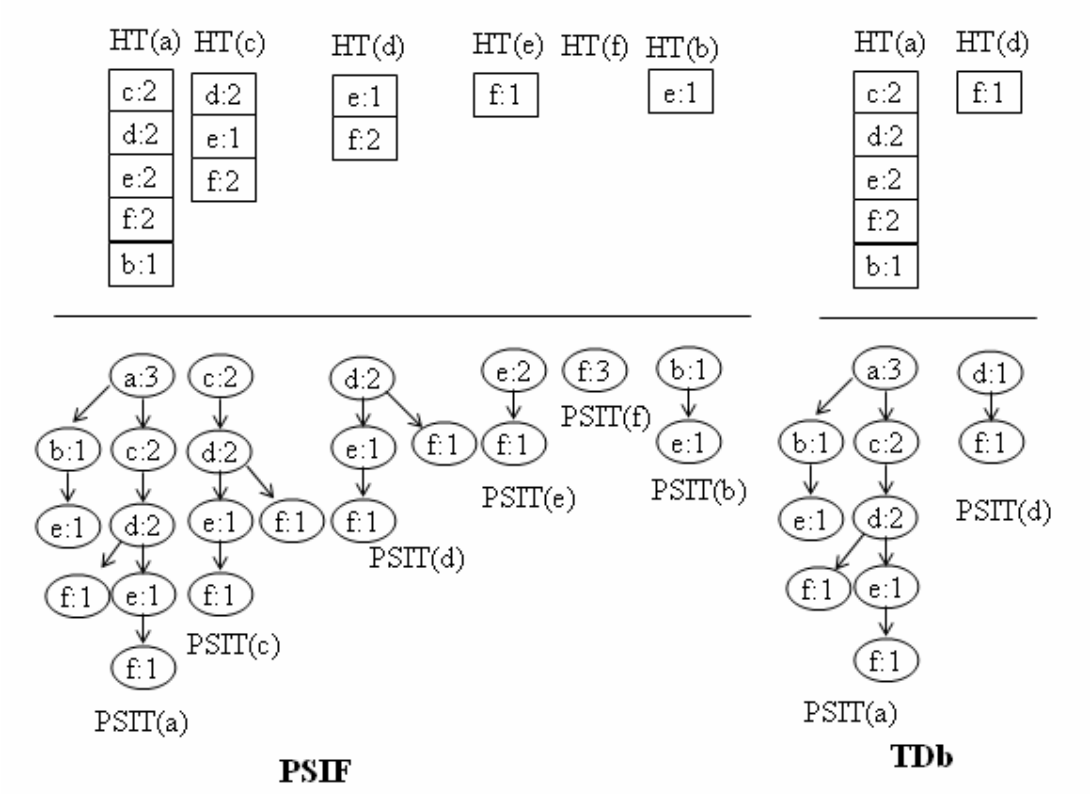

Figure 3.5. PSIF and TDb after inserting the fourth transaction *acdf*

5) *Fifth transaction cef*: ARHDS reads the fifth transaction *cef,* inserts item-suffix transactions *cef, ef, f* into PSIF and *cef* into TDb. Results are shown in Figure 3.6.

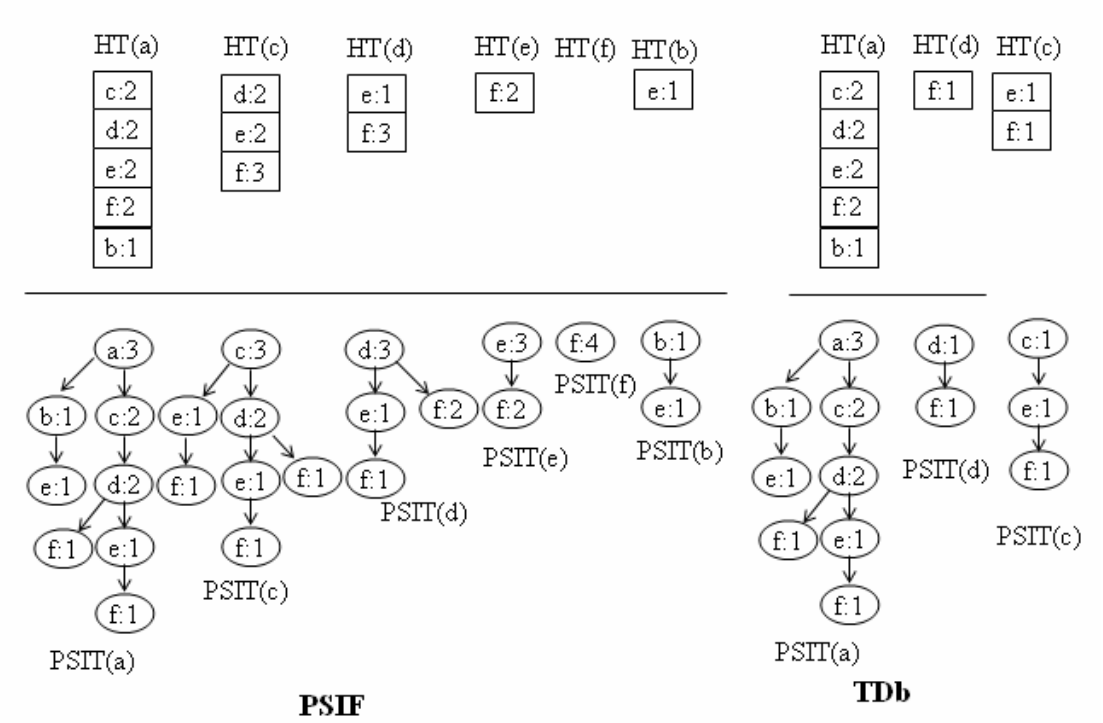

Figure 3.6. PSIF and TDb after inserting the fifth transaction *cef*

Step 3: After processing the first block B_1, ARHDS prunes insensitive itemsets from the current PSIF. At this time, ARHDS deletes the PSIT(b) and its corresponding HT(b), and prunes the entry *b* from all other PSIT 's because item *b* is an insensitive item; that is, $Esup(b) < \varepsilon * CL$ ($1 < 0.25 * 5$). The resulting PSIF is shown in Figure 3.7.

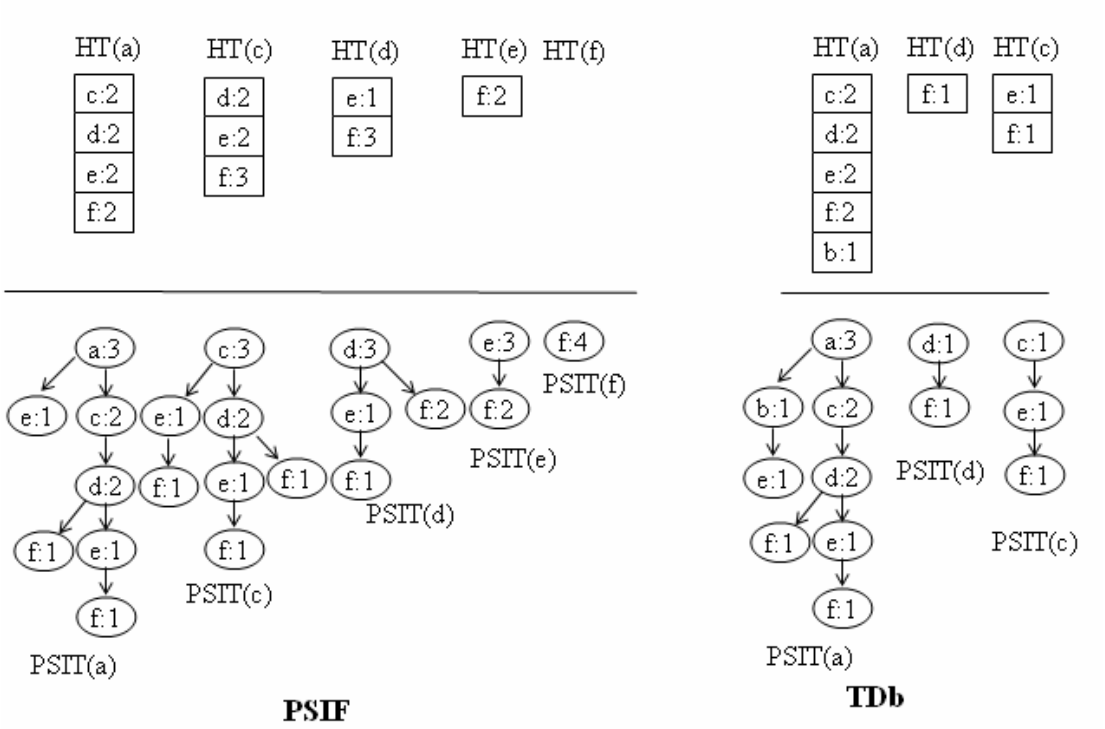

Figure 3.7. PSIF and TDb after pruning insensitive item b

After pruning the insensitive items from PSIF, we read the second block B_2 for *constructing the PSIF*. The construction process of PSIF with respect to block B_2 is the same as Step 1 and Step2. The result is shown in Figure 3.8. Later, we go to the Step 4 as we start hiding sensitive itemsets.

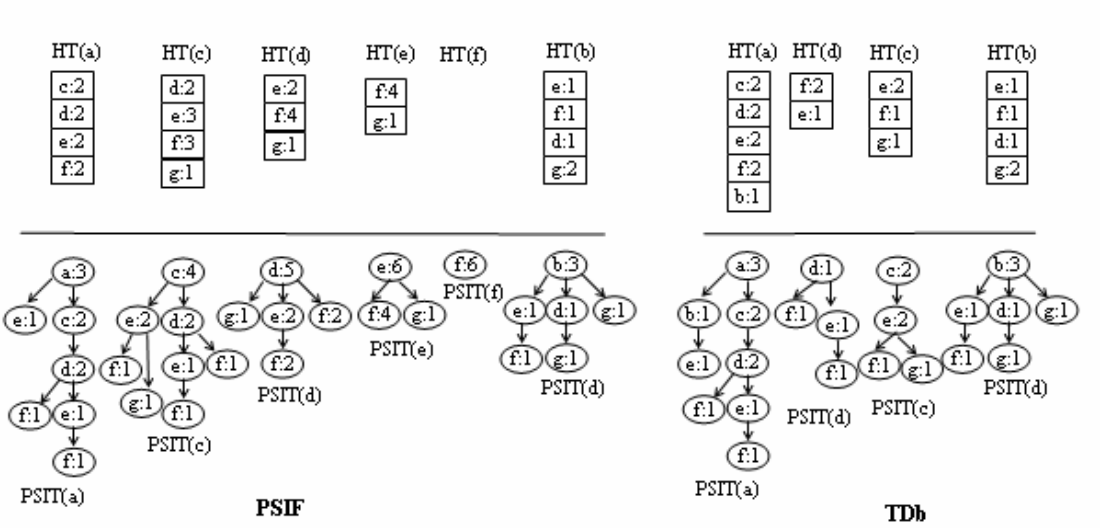

Figure 3.8. PSIF and TDb after processing the second block B_2

Step 4: After processing the second block B_2, first we find all sensitive itemsets with respect to the minimum support threshold *ms =0.30.* Then we sort the sensitive itemsets according to their support. Table 3.1 shows all sensitive itemsets, sorted by their support. Then, beginning from the sensitive itemset with the highest support, we discover the sensitive disjoint itemsets to hide. Table 3.2 shows the obtained sensitive disjoint itemsets.

Table 3.1. All sensitive itemsets sorted by their support

Itemset	**Support**
df	40%
ef	40%
ce	30%
cf	30%

Table 3.2. Disjoint sensitive itemsets

Itemset	**Support**
df	40%
ce	30%

Step 5: In this step, we do hiding sensitive itemsets by the hiding algorithm we purposed. Table 3.3 shows the transactions to change and items to be hidden and Table 3.4 shows the updated database.

Table 3.3. Transactions to change and items to be hidden.

Transactions	**Items**
acdef	d
acdf	f
acef	c

Table 3.4. Original and the hidden database

Original DB	**Hidden DB**
acdef	aef
df	df
abe	abe
acdf	acd
cef	cef
bef	bef
bdg	bdg
def	def
bg	bg
ceg	ceg

After completing first iteration we go back to Step 4 to check if there still exists any sensitive itemsets. Table 3.5 shows all sensitive itemsets and Table 3.6 shows disjoint sensitive itemsets discovered by the second iteration.

Table 3.5. All sensitive itemsets discovered by the second iteration

Itemset	**Support**
ef	40%

Table 3.6. Disjoint sensitive itemsets obtained by the second iteration

Itemset	**Support**
ef	40%

Later, we come back to Step 5 to hide the disjoint sensitive itemsets we discovered by the second iteration. Table 3.7 shows the transactions to change and items to be hidden.

Table 3.7. Transactions to change and items to be hidden at second iteration

Transactions	**Items**
aef	e
def	f

Then, hide the sensitive itemsets from the database. Table 3.8 shows the updated database summary.

Table 3.8. Original and the hidden database

Original DB	**DB after 1st iteration**	**DB after 2nd iteration**
acdef	aef	af
df	df	df
abe	abe	abe
acdf	acd	acd
cef	cef	cef
bef	bef	bef
bdg	bdg	bdg
def	def	de
bg	bg	bg
ceg	ceg	ceg

After the second iteration, again we go back to Step 4 to check if there still exists any sensitive itemsets. As we find no sensitive itemsets, we continue with Step 6. Figure 3.9 shows the hidden TDb.

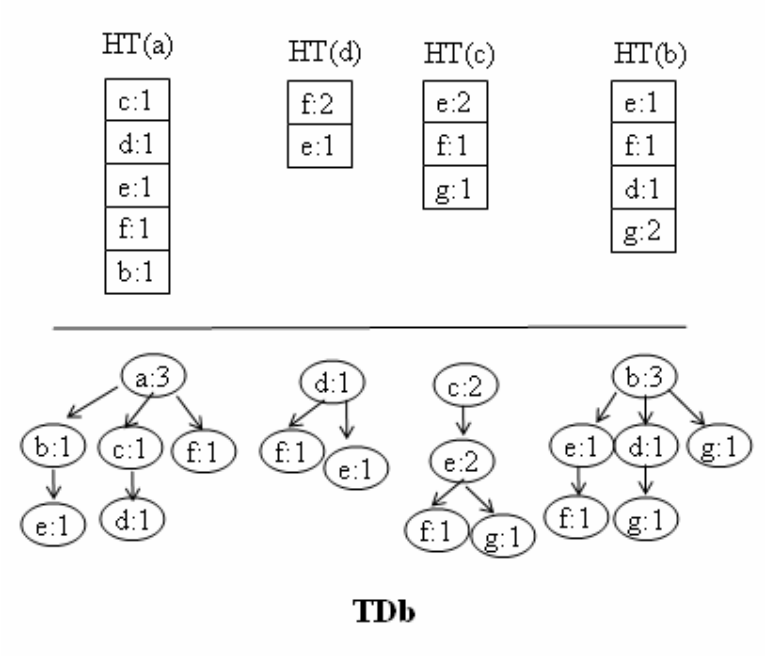

Figure 3.9. Hidden TDb

Step 6: Lastly we send the hidden database to the receiver.

Example 2: Let us suppose that the data stream is the same as in Example 1, but the stream arrives in XML format. Figure 3.10 shows the XML stream for the first block B_1.

Figure 3.10. XML stream for the first block B_1

As we explained earlier, all the steps of our algorithm are same for XML stream. The only difference is that at each leaf node, we store the transaction number for each transaction. Figure 3.11 shows the TDb after processing the first block B_1, Figure 3.12 shows the TDb after processing the second block B_2 and Figure 3.13 show the hidden TDb for XML stream. While hiding an item from a transaction, we make necessary operations for updating the place where we store the transaction number. When we send the hidden data to the receiver; we use these transaction numbers to merge the data on which we do association rule hiding with its related part.

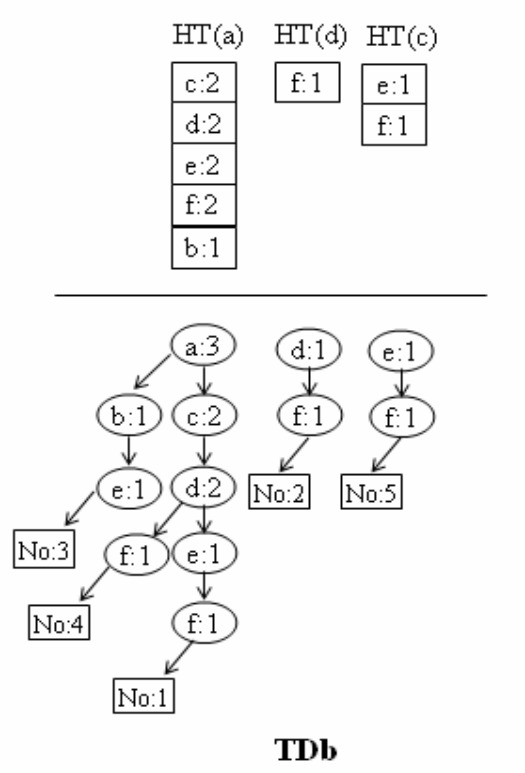

Figure 3.11. TDb for XML data after processing the first block B_1

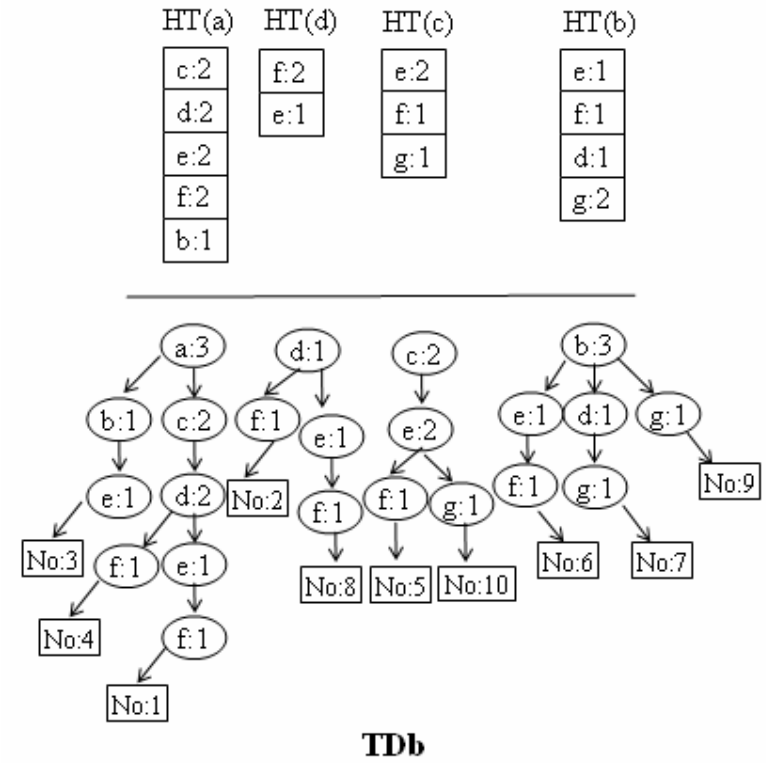

Figure 3.12. TDb for XML data after processing the first block B_2

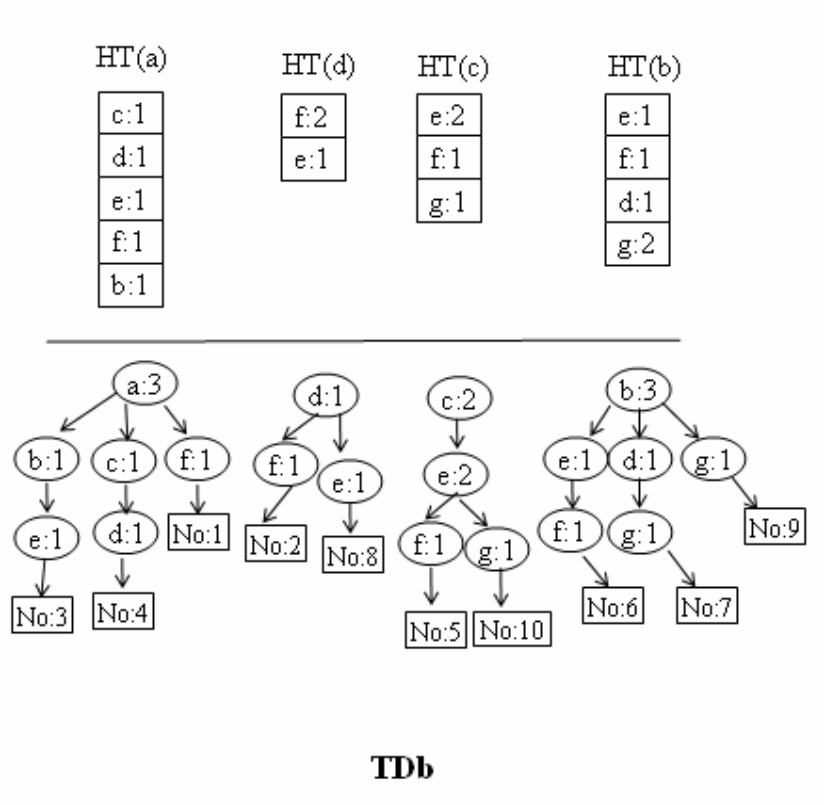

Figure 3.13. Hidden TDb for XML data

4. ANALYSIS AND EXPERIMENTAL RESULTS

4.1. Overview of Results

We have tried to assess the performance of our introduced system ARHDS, examine its results with different parameter settings. We have also examined the scalability of the algorithm, which is one of the powerful capabilities of our algorithm, by running it for both raw data and XML data with different parameters.

The most important issue in ARHDS is the execution time since we must hide the sensitive rules of the data and send the hidden database to the receiver in a very short time. We have evaluated the time we consume to finish different steps of ARHDS such as creating PSIF (Potentially Sensitive Itemset Forest) and TDb (Temporary Database), hiding the sensitive rules and writing the output data to the file (sending the hidden database to the receiver).

Later, we tried to analyze the performance of ARHDS under different parameter settings. We have performed some tests under different values of parameters such as user-specified minimum support threshold $ms \in (0, 1)$, a user-defined error threshold $\varepsilon \in (0, ms)$ and the block size for the synthetic data. We presented analysis of the results by the help of graphics and plots.

Finally we run ARHDS for XML data we generated and made a comparison of our proposed algorithm on XML data and on raw data according to their execution time. Also as we improved one of the algorithms developed by [9] for the sensitive rule hiding part of our system, we attempted to compare hiding time of our algorithm with the algorithm of [9].

4.2. Overview of Synthetic Data

To run our proposed system ARHDS, we have generated some synthetic datasets via the IBM's data generator [2]. For clarity we name each dataset in the form of TxxIxxDxx

where T, I and D mean the average transaction length, the average length of maximum pattern, and the total number of transactions, respectively. For evaluating our work, we have used four datasets: T7I4D200K, T5I4D10K, T5I4D50K, and T5I4D100K. We made use of T7I4D200K for the performance evaluation of the proposed system. We used other datasets for comparison of hiding time of ARHDS with one of the algorithms developed by [23]. The parameter setting for generating the synthetic dataset used in the experiments are shown in Table 4.1.

Table 4.1. Parameter setting for generating the synthetic dataset

Parameter	**Dataset I**	**Dataset II-III-IV**
D: number of transactions	200K(200000)	10K-50K-100K
T: average number of items per transaction	7	5
I: average length of maximal pattern	4	4
number of distinct items	1000	50

4.3. Overview of XML Data

To evaluate our ARHDS on XML data, we have generated synthetic XML data by using the synthetic data generated via the IBM's data generator [2]. As we try to hide association rules over XML streaming data by the user guided template, that is; since we are interested only in some parts of the XML stream to hide the association rules, we have used the synthetic data which we generated before as the essential part of the synthetic XML data on which we will do association rule hiding. The parameter setting is the same as Table 4.1 for the XML data. To explain the generation of our synthetic XML data clearer, let's give an example:

Suppose our XML data will consist of purchase summaries obtained from a market chain and suppose the user is only interested in the product ids for association rule hiding. Assume that the first transaction in the dataset which we generated by IBM's data generator is the "1 5 8 9 46". Figure 4.1 shows an example of our synthetic XML data for this transaction.

```
<?xml version="1.0" encoding="utf-8"?>
<PurchaseCollection>
  <Purchase>
    <No>1</No>
    <Date>Date of 1</Date>
    <Place>Place of 1</Place>
    <Company>Company of 1</Company>
    <Type>Type of 1</Type>
    <Cost>Cost of 1</Cost>
    <Products>
      <ProductID>1</ProductID>
      <ProductID>5</ProductID>
      <ProductID>8</ProductID>
      <ProductID>9</ProductID>
      <ProductID>46</ProductID>
    </Products>
  </Purchase>
 </PurchaseCollection>
```

Figure 4.1. XML for the transaction “1 5 8 9 46”

4.4. The Experimental Setup

We all made our experiments on a PC with AMD Athlon(TM) 1.8GHz CPU, 1GB main memory and Microsoft XP Professional. All the implementation for proposed algorithm is written in Microsoft Visual C# 2.0 as the application development environment Microsoft Visual Studio 2005 is used. We ran our code in VS 2005 environment as well.

To run our implementation with a short execution time, we have made use of Dictionary (implemented as a generic hash table) and List (generic equivalent of the ArrayList class) classes of the Generic collection in c# 2.0.

We had spent some effort for generating the synthetic data to run our experiments. We had used the open source [24] to generate data. The code was written in Linux environment so to run it on Windows we had used a free Linux-like environment for Windows [25].

For generating the XML data, again we have used Microsoft Visual C# 2.0 utilities. We generated XML data by using the raw data which we have generated by IBM data generator.

4.5. Performance Evaluation of ARHDS

The choice of parameters is an important issue for the application of any of the algorithm.

Firstly, as the most important factor in association rule hiding over data streams is the *time of execution* to send the hidden data to the receiver, we have examined the execution time of ARHDS. We have evaluated the time for creating PSIF (Potentially Sensitive Itemset Forest) and TDb (Temporary Database), hiding the sensitive rules and writing the output data to the file (sending the hidden database to the receiver).

Afterwards, we tried to analyze the performance of ARHDS under different parameter settings. We have run our proposed system under different values for parameters which are: a user-specified minimum support threshold $ms \in (0, 1)$, a user-defined error threshold $\varepsilon \in (0, ms)$ and the block size for the synthetic data.

Finally we have made a comparison of ARHDS on XML data and on raw data we have generated according to their execution time.

4.6. Performance results

Test 1: The first test is performed to examine the time executed by our algorithm at each step. With $ms = 0.0025$ and $\varepsilon = 0.0005$, we run ARHDS on T7I4D200K data. The data is broken into blocks with size 25K for simulating the continuous characteristics of streaming data. Hence there are 8 blocks in this test. Figure 4.2 shows the execution time for creating PSIF and TDb before we make association rule hiding.

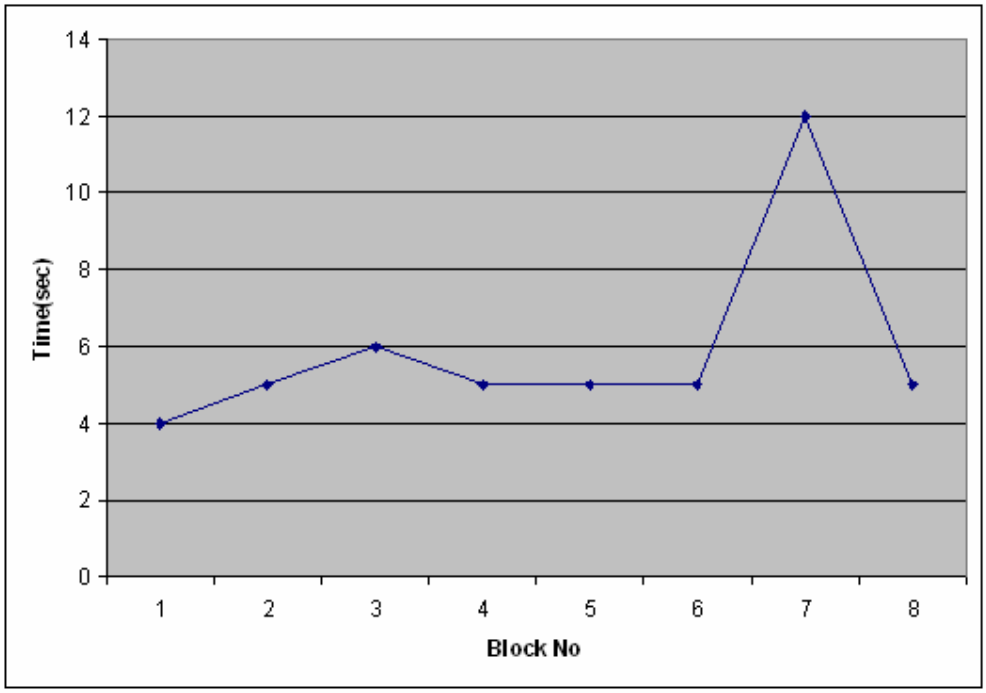

Figure 4.2. Execution time for creating PSIF and Tdb

Note that after arrival of each block, we make pruning operation for PSIF. We expect that the time needed for inserting new transactions into both PSIF and TDb will increase with the incoming blocks. From the figure above we can see some points at where execution time decreases. This may be due to the characteristic of the data we generated. Since there is no sharp increase in execution time (seconds) we can say that our system creates the data structure for ARHDS efficiently.

Table 4.2 shows number of all sensitive itemsets, number of disjoint sensitive itemsets chosen after sorting all sensitive itemsets with respect to their support and the number of transactions changed during the hiding process. As expected, all of them decrease by the next iteration. Also note that by number of transactions changed we actually mean that the number of attempts to change a transaction as we can change the same transaction for many times.

Table 4.1. Number of all and disjoint sensitive itemsets and transactions changed

	# all sensitive itemsets	**# disjoint sensitive itemsets**	**# transactions changed**
Iteration 1	20	7	849
Iteration 2	15	5	449
Iteration 3	8	4	192
Iteration 4	5	1	57
Iteration 5	4	1	53
Iteration 6	3	1	40
Iteration 7	2	1	2
Iteration 8	1	1	1

Table 4.3. shows the execution time of proposed system at each step. These results can change with respect to the parameter settings, that is to say; hiding time can exceed constructing PSIF and TDb time.

Table 4.2 Execution time of ARDHS at each step

	Execution Time
Constructing PSIF and TDb	47 sec
Hiding disjoint sensitive itemsets	15 sec
Writing hidden database to a file	4 sec
	Total Time: 66 sec

Test 2: To evaluate the scalability of our approach, we performed our second test on the user-defined minimum error threshold. By keeping other variables constant (*ms* = 0.0025, block size=25K, # of blocks = 8) we play with the minimum error threshold ε. Figure 4.3 shows execution times for different minimum error threshold. As there is no sharp increase or decrease in the graph, we can say that our approach is stable.

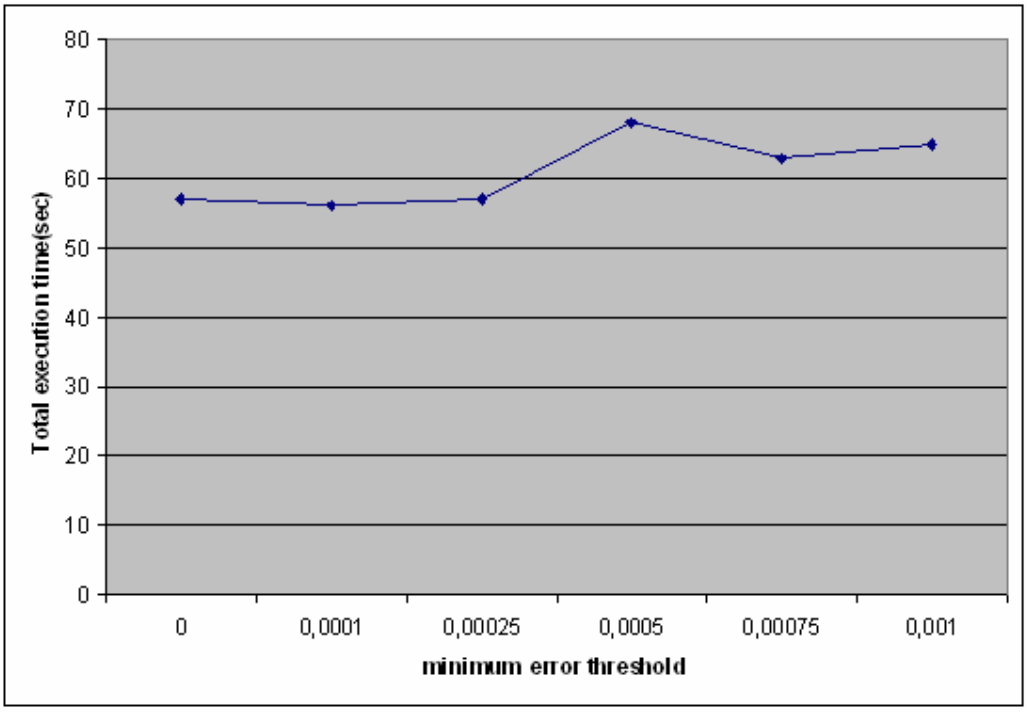

Figure 4.3. Execution time vs. minimum error threshold

Test 3: As a third test, to examine the execution time of hiding the association rules for the disjoint sensitive itemsets, we run ARHDS with different minimum support threshold *ms* by keeping other variables constant (ε = 0.0005, block size=25K, # of blocks = 8). Figure 4.4 shows execution time for hiding disjoint sensitive itemsets changing with different minimum support threshold values. Note that in addition to time required for hiding process, execution time includes time spent for finding all sensitive itemsets and getting the disjoint sensitive itemsets. Figure 4.4 shows that our heuristic to hide all sensitive itemsets from the database gives efficient results.

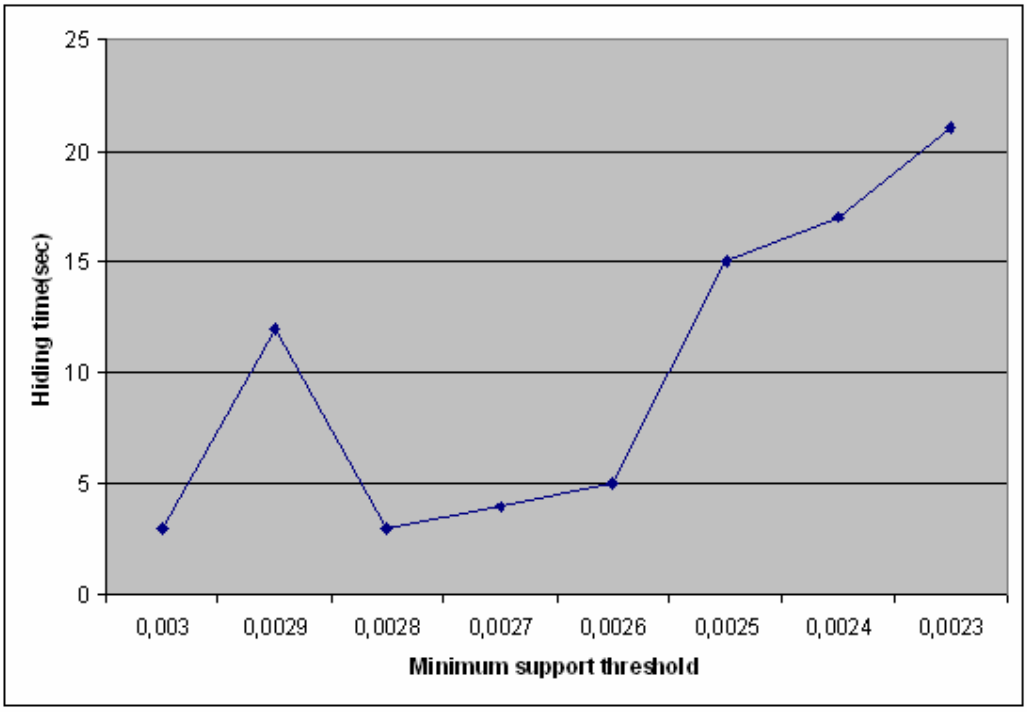

Figure 4.4. Execution time for hiding vs. minimum support threshold

Table 4.4 shows the hiding time and number of transactions changed for different minimum support threshold values.

Table 4.3. Hiding time and number of transactions changed

ms	**execution time**	**# transactions changed**
0,0030	3	620
0,0029	12	740
0,0028	3	899
0,0027	4	1098
0,0026	5	1344
0,0025	15	1642
0,0024	17	1968
0,0023	21	2487

We see that if the minimum support threshold value decreases, the number of transactions changed increases but hiding time does not always increase. This is due to the

heuristic we developed for hiding sensitive rules. Also we can conclude that, we should choose our minimum support threshold carefully to run ARHDS in short execution time.

Test 4: Our fourth test is performed to examine the time executed by our algorithm with different size of blocks, again by keeping other variables constant (*ms* = 0.0025 and ε = 0.0005). Figure 4.5 shows the total execution time of ARHDS with changing block size for the data T7I4D200K. Having highest execution time with the block size 10K can be due to the highest number of pruning or the characteristics of the data we tested. From the figure we can conclude that system is stable under different block sizes.

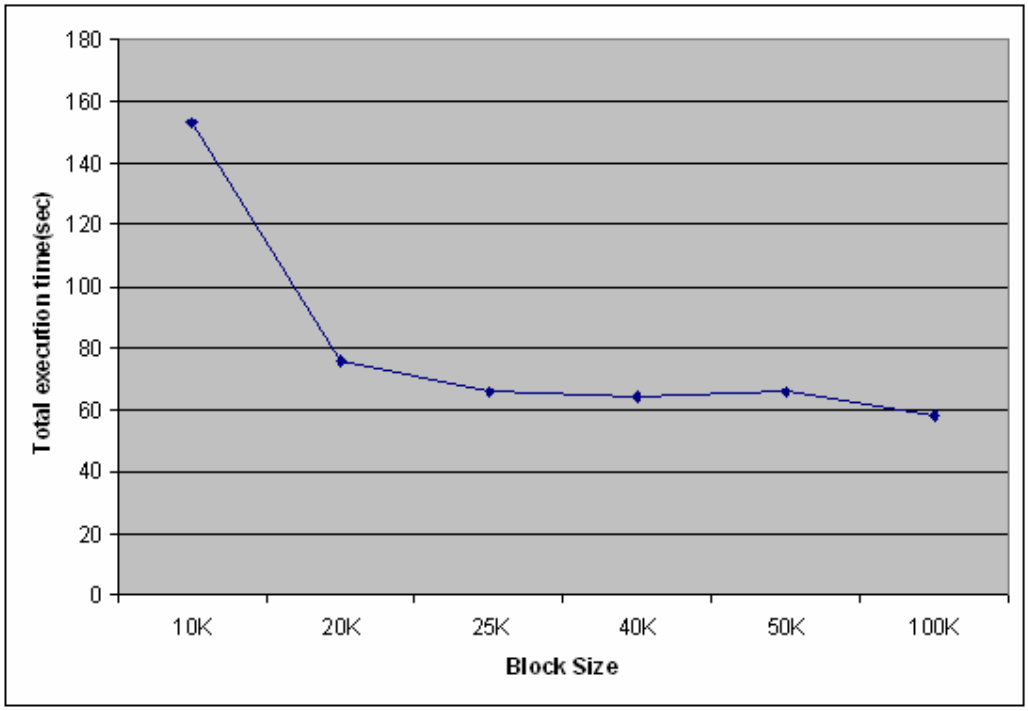

Figure 4.5. Total execution time for hiding vs. different block size

4.6.1. Comparison with XML data

Our fifth test is performed to compare the execution time of our algorithm for raw data and the XML data we generated. For comparison, we used same parameters; ms = *0.0025,* ε = 0.0005, and the block size = 25K. Also as we suppose that the user provided xml template guides us to do association rule hiding on the part of the XML data which is same as our raw data, that is; T7I4D200K. Figure 4.6 shows the comparison of execution

time calculated to create PSIF and TDb for raw data and XML data before we make association rule hiding. From the figure, we see that execution time grows sharply at the last block. This is due to the fact that we also store the other part of XML data which we do not use for association rule hiding but do store to send back to the receiver (File sizes for raw data and XML data are 6MB and 120MB respectively).

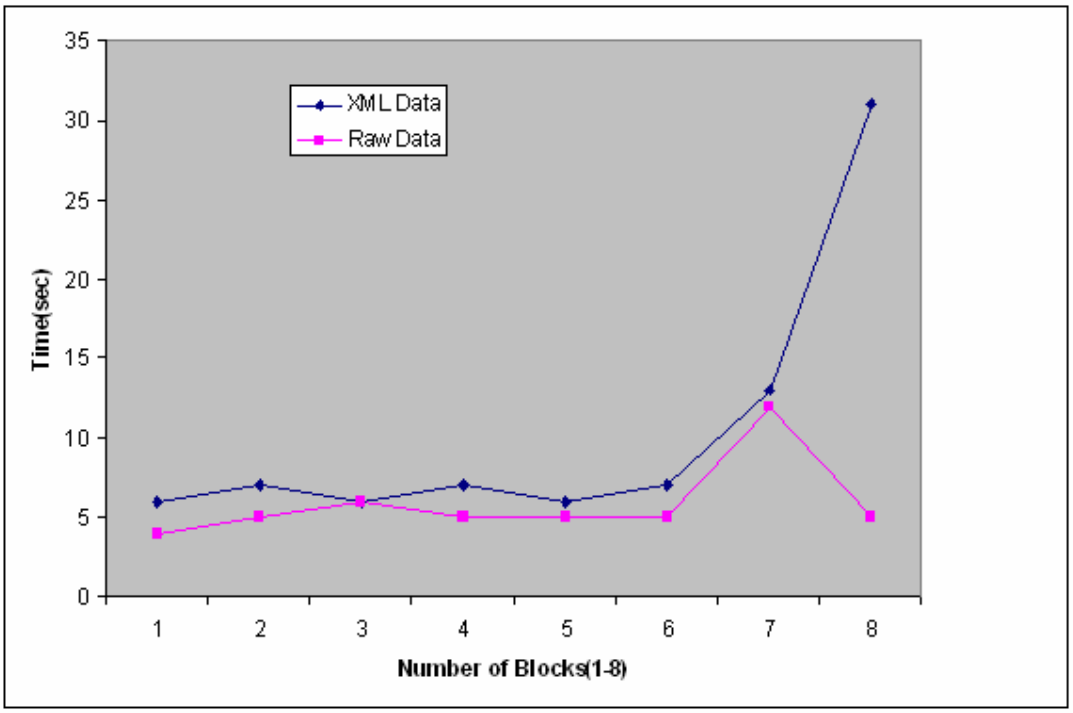

Figure 4.6. Comparison of execution times to create PSIF and Tdb

Table 4.5 shows the hiding time and time of sending the hidden DB to the receiver for raw data and the XML data. As expected there is not a sharp difference in hiding time but it is important to note the difference between the times for sending hidden Db to the receiver.

Table 4.4. Hiding time and time of sending hidden DB to receiver

	hiding (sec)	**sending hidden DB (sec)**
Raw Data	15	4
XML Data	17	20

4.6.2. Comparison of hiding time

Since we improve one of the algorithms developed by [9] for the sensitive rule hiding part of our system, we attempted to compare hiding time of our algorithm with the algorithm of [9].

In their work, [9] run the algorithm they developed just to hide 5 or 10 rules they have chosen. In our system however, we try to hide all sensitive rules above a user specified minimum support threshold from our database. From this point of view, a correct comparison of our algorithm with [9] is not possible. However, we run ARHDS with different minimum support thresholds and, similar to [9], with different size of the databases to get some idea about the hiding time of our system. Table 4.6 and Table 4.7 show the results for [9] and our system ARHDS respectively.

Table 4.5. Results for the algorithm developed by [9]

	10K		**50K**		**100K**	
# rules	5	10	5	10	5	10
#transactions changed	1450	3170	3770	6310	1170	4550
hiding time(sec)	2	3	11	14	23	28,5

Table 4.6. Results for proposed system ARHDS

	10K			**50K**			**100K**		
min support threshold	0.6	0.5	0.4	0.7	0.65	0.6	0.9	0.8	0.7
# rules	9	23	41	6	8	12	4	6	6
#transactions changed	1236	2263	4457	5144	6515	8369	1591	5481	10315
hiding time(sec)	1	10	20	9	12	18	2	13	32

From the tables above, we can conclude that hiding time increases with the increasing number of transactions changed. Although ARHDS's hiding time is higher than [9] for 10K in spite of having smaller number of transactions changed (10 sec-2263 transactions vs. 3 sec-3170 transactions) ARHDS outperforms [9] for 50K and 100K (12 sec-6515 transactions vs. 14 sec-6310 transactions and 13 sec-5481 transactions vs. 23 sec-1170 transactions).

Here we should note that we can not compare two systems by just looking at the number of rules hidden at this case because both systems does not hide the same rules. Hiding more rules with smaller supports may cause a wrong comparison. Also by just looking at the tables above we can not say ARHDS outperforms [9] or vice versa as we choose the transactions to be hidden randomly and the supports of the itemsets fluctuate. An exact comparison can be made by running both systems on the same set of rules for the same database.

5. DISCUSSION AND CONCLUSION

In this work we have introduced and analyzed a new system we named ARHDS for hiding association rule over data streams. ARHDS provides a fast and efficient approach for hiding sensitive knowledge over data streams.

Mining data streams is an interesting and challenging research field. The characteristic of data streams make the traditional mining algorithm unable to be applied for data streams. Also, due to the fact that recent advances in data mining algorithms have increased the disclosure risks that one may encounter when releasing data to outside parties, association rule hiding is another interesting challenging research field. In this work we tried to merge these two challenging research areas.

Our algorithm mainly consists of two parts. First part is creating our data structure to discover sensitive itemsets to hide and the second part is hiding these itemsets from the data we stored. For the first part we use a prefix tree structure and for the second part we hide sensitive itemsets by decreasing their supports.

Since we introduce a new system by combining stream association rule mining with association rule hiding for traditional databases, we did not have the chance to compare our system with another. However, we made various experiments. We ran ARHDS with different parameter settings to examine the scalability and stability of our algorithm. We also tested our proposed system on both raw data and the XML data. We found that our algorithm is fast and efficient.

Our study reveals that in association rule hiding over data streams, both the algorithms to find the sensitive rules and to hide these rules are important to have a fast system. Also implementation issues are important while combining these two algorithms to have an efficient system. It remains a future work to make a more efficient implementation for our proposed system. Also running our algorithm on different datasets remains another future work to improve accuracy and efficiency of our system.

APPENDIX A. XML

In this section we will give information about XML structure.

XML: XML (Extensible Markup Language), which provides a text-based means to describe and apply a tree-based structure to information, is a flexible way to create common information formats and share both the format and the data on the World Wide Web, intranets, and elsewhere. For example, computer makers might agree on a standard or common way to describe the information about a computer product (processor speed, memory size, and so forth) and then describe the product information format with XML. Such a standard way of describing data would enable a user to send an intelligent agent (a program) to each computer maker's Web site, gather data, and then make a valid comparison. XML can be used by any individual or group of individuals or companies that wants to share information in a consistent way. Figure A.1 shows a sample XML file.

```
<?XML VERSION="1.0" STANDALONE="yes" ?>
<STATE STATEID="MN">
  <CITY CITYID="12">
    <NAME>
      Johnson</name>
      <POPULATION>5000</POPULATION>
    </CITY>
  <CITY CITYID="15">
    <NAME>Pineville</NAME>
    <POPULATION>60000</POPULATION>
  </CITY>
  <CITY CITYID="20">
    <NAME>Lake Bell</NAME>
    <POPULATION>20</POPULATION>
  </CITY>
</STATE>
```

Figure A.1. Sample XML file

Some advantages of XML can be listed as:

- XML is human readable.
- XML provides a structure to data so that it is richer in information

- XML can be used as an exchange format to enable users to move their data between similar applications
- XML is easily processed because the structure of the data is simple and standard
- There is a wide range of reusable software available to programmers to handle XML so they don't have to re-invent code
- XML separates the presentation of data from the structure of that data
- A common exchange format
- Many views of data

XSD: An XML Schema Definition (XSD) is an instance of an XML schema written in XML Schema to describe and validate data. An XSD defines a type of XML document in terms of constraints upon what elements and attributes may appear their relationship to each other, what types of data may be in them, and other things. Because XSD is written in XML, there is no need for a parser. XSD defines a richer set of data types such as booleans, numbers, and dates and times, and currencies -- which is invaluable for e-commerce applications**.** In this work, we use xsd to determine which attributes of xml will be hidden.

XSLT: Extensible Stylesheet Language Transformations (XSLT) is an XML-based language used for the transformation of XML documents. XSLT is a specific kind of template processor primarily designed to "transform" XML documents into other XML documents. The original document is not changed; rather, a new document is created based on the content of an existing one. The new document may be serialized (output) by the processor in standard XML syntax or in another format, such as HTML or plain text. XSLT is most often used to convert data between different XML schemas or to convert XML data into HTML or XHTML documents for web pages, or into an intermediate XML format that can be converted to PDF documents.

REFERENCES

1. Agrawal, R., T. Imielinski, and N. Swami, Mining association rules between sets of items in large databases, in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, 207-216, 1993.

2. Agrawal, R. and R. Srikant, Fast algorithms for mining association rules, In *Proc. 20th Int. Conf. Very Large Data Bases, VLDB*, 487-499, 1994.

3. Cai, Y. D., G. Pape, J. Han, M. Welge and L. Auvil, MAIDS: Mining Alarming Incidents from Data Streams, *International Conference on Management of Data*, 2004.

4. Chang, J. H. and W. S. Lee, A Sliding Window Method for Finding Recently Frequent Itemsets over Online Data Streams, *Journal of Information Science and Engineering*, 2004.

5. Charikar, M., K. Chen and M. Farach-Colton, Finding Frequent Items in Data Streams, *Theoretical Computer Science,* 2004.

6. Chi, Y., H. Wang , P. S. Yu and R. Richard, Moment: Maintaining Closed Frequent Itemsets over a Stream Sliding Window, *IEEE Int'l Conf. on Data Mining,* 2004.

7. Demaine, D., A. Lpez-Ortiz and J. I. Munro, Frequency Estimation of Internet Packet Streams with Limited Space, *European Symposium on Algorithms,* 2002.

8. Elmagarmid, A., M. Atallah, E. Bertino, M. Ibrahim and V. Verykios, Disclosure Limitation of Sensitive Rules. *Proceedings of Knowledge and Data Exchange Workshop*, 1999.

9. Elmagarmid, A., V. Verykios, A. K. Bertino, E. Dasseni and Y. Saygin, *Association Rule Hiding*, *IEEE Transactions on Knowledge and Data Engineering*, vol. 16, no. 4, pp. 434-447, 2004.

10. Frawley, W., G. Piatetsky-Shapiro, C. J. Matheus, Knowledge Discovery in Databases: An overview, AAAI/MIT Press, Menlo Park, CA, sid 1-27, 1991.

11. Giannella, C., J. Han, J. Pei, X. Yan and P. S. Yu, Mining Frequent Patterns in Data Streams at Multiple Time Granularities; Data Mining: Next Generation Challenges and Future Directions, AAAI/MIT, 2003.

12. Guha, S., N. Koudas and K. Shi, Data Streams and Histograms, ACM *Symposium on Theory of Computing*, 2001.

13. Halatchev, M. and L. Gruenwald, Estimating Missing Values in Related Sensor Data Streams, *Int'l Conf. on Management of Data*, 2005.

14. Han, J., G. Dong and Y. Yin, Efficient mining of partial periodic patterns in time series database, *IEEE Int'l Conf. on Data Mining*, 1999.

15. Han, J., J. Pei and Y. Yin, Mining Frequent Patterns without Candidate Generation, *Int'l Conf. on Management of Data*, 2000.

16. Jiang, N. and L. Gruenwald, Research Issues in Data Stream Association Rule Mining, *ACM SIGMOD* Record, Vol. 35, No. 1, 2006.

17. Kargupta, H., R. Bhargava, K. Liu, M. Powers, P. Blair, S. Bushra, J. Dull, K. Sarkar, M. Klein, M. Vasa and D. Handy, VEDAS: A Mobile and Distributed Data Stream Mining System for Real-Time Vehicle Monitoring; *SIAM Int'l Conf. on Data Mining*, 2004.

18. Karp, R. M. and S. Shenker, a Simple Algorithm for Finding Frequent Elements in Streams and Bags, *ACM Transactions on Database Systems*, 2003.

19. Li, H. F., S. Y. Lee and M. K. Shan, An Efficient Algorithm for Mining Frequent Itemsets over the Entire History of Data Streams, *Int'l Workshop on Knowledge Discovery in Data Streams*, 2004.

20. Li, H. F., S. Y. Lee and M. K. Shan, Online mining of frequent query trees over XML data streams, *Proceedings of the 15th international conference on World Wide Web*, 2006.

21. Manku, G. S. and R. Motwani, Approximate Frequency Counts over Data Streams, *Int'l Conf. on Very Large Databases*, 2002.

22. Mao, G., X. Wu, C. Liu, X. Zhu, G. Chen, Y. Sun and X. Liu, Online Mining of Maximal Frequent Itemsequences from Data Streams, University of Vermont, Computer Science Technical Report CS-05-07, 2005.

23. Verykios, V. S., E. Bertino, I. N. Fovino, L. P. Provenza, Y. Saygin and Y. Theodoridis, State-of-the-Art in Privacy Preserving Data Mining, *ACM SIGMOD Record*, vol. 3, no. 1, pp. 50-57, 2004.

24. IBM Data Generator, Source code for IBM's synthetic dataset generator http://www.almaden.ibm.com/cs/projects/iis/hdb/Projects/data_mining/datasets/syndata.html.

25. Cygwin, a Linux-like environment for Windows http://www.cygwin.com.

26. Yang, L. and M. Sanver, Mining Short Association Rules with One Database Scan, *Int'l Conf. on Information and Knowledge Engineering*, 2004.

27. Yu, J. X., Z. Chong, H. Lu and A. Zhou, False Positive or False Negative: Mining Frequent Itemsets from High Speed Transactional Data Streams, *Int'l Conf. on Very Large Databases*, 2004.

28. Zhu, Y. and D. Shasha, StatStream: Statistical Monitoring of Thousands of Data Streams in Real Time, *Int'l Conf. on Very Large Databases*, 2002.