University of Alberta

A Framework for Enforcing Privacy in Mining Frequent Patterns

by

Stanley R. M. Oliveira
Osmar R. Zaïane

Technical Report TR 02-13
June 2002

DEPARTMENT OF COMPUTING SCIENCE
University of Alberta
Edmonton, Alberta, Canada

A Framework for Enforcing Privacy in Mining Frequent Patterns

Stanley R. M. Oliveira[1,2]
oliveira@cs.ualberta.ca

Osmar R. Zaïane[1]
zaiane@cs.ualberta.ca

[1]Department of Computing Science, University of Alberta, Canada
[2]Embrapa Information Technology, Campinas, São Paulo, Brazil

Abstract

Discovering hidden patterns from large amounts of data plays an important role in marketing, business, medical analysis, and other applications where these patterns are paramount for strategic decision making. However, recent research has shown that some discovered patterns can pose a threat to security and privacy. One alternative to address the privacy requirements in mining hidden patterns is to look for a balance between hiding restrictive patterns and disclosing non-restrictive ones. In this paper, we propose a new framework for enforcing privacy in mining frequent itemsets. We combine, in a single framework, techniques for efficiently hiding restrictive patterns: a transaction retrieval engine relying on an inverted file and Boolean queries; and a set of algorithms to "sanitize" a database. In addition, we introduce mining performance measures for frequent itemsets that quantify the fraction of mining patterns which are preserved after sanitizing a database. We also report the results of a performance evaluation of our research prototype and an analysis of the results.

1 Introduction

Data mining techniques have emerged as a means of extracting hidden or previously unknown information from large repositories of data. One of the most studied problems in data mining is the process of discovering frequent itemsets and, consequently, association rules [2, 3, 22]. The discovery of interesting frequent patterns among huge amounts of transactional records can be very effective in revealing actionable knowledge that leads to strategic decisions.

Despite its benefits in various areas, data mining can also pose a threat to privacy and information security if not done or used properly. Recent advances in data mining and machine learning algorithms have introduced new problems in database security [11, 5, 14, 21], i.e., these techniques are now moving to other domains where privacy issues are very significant. The basic problem is that from non-sensitive data, one is able to infer sensitive information, including personal information, facts, or even patterns that are not supposed to be disclosed.

There is no exact solution that resolves this privacy problem in data mining. An approximate solution could be sufficient depending on the application since the level of privacy can be interpreted in different contexts. Clifton et al. [12] argued that the goal is to achieve a solution with bounded error, or difference from the data house solution. Indeed, privacy issues in data mining cannot simply be addressed by restricting data collection or even by restricting the use of information technology. An appropriate balance between a need for privacy and knowledge discovery should be found [7].

Clifton and Marks [13] enumerated various scenarios in which mining applications require preventing the disclosure of information, including market basket analysis. Let us consider, for example, a situation in which one supplier offers products at a reduced price to some consumers and, in turn, this supplier receives permission to access the database of the consumers' customer purchases. The threat becomes real whenever the supplier is allowed to derive highly sensitive knowledge from unclassified data that is not even known to the database owners (consumers). In this case, the consumers benefit from reduced prices, whereas the supplier is provided with enough information to predict inventory needs and negotiate other products to obtain a better deal for his consumers. This implies that the competitors of this supplier start losing business.

Notice that the simplistic solution to enforce privacy in data mining is to implement a filter after the mining phase to weed out/hide the restricted discovered patterns. However, in the context of our research, the users are provided with the data and not the patterns, and are free to use their own tools, and thus the restriction for privacy has to be applied before the mining phase on the data itself. The process of removing or altering transactions for the purpose of hiding restricted patterns from within the data is also refered to as "data sanitization" in [5]. As we shall see, in our techniques, we do not add itemsets in transactions but only strategically and selectively remove individual items from sensitive transactions preventing the disclosure of some patterns while preserving as much of the original information as possible for other applications.

In this paper, we propose a new framework for enforcing privacy in mining frequent patterns that combines techniques for efficiently hiding restrictive rules. The framework is composed of a transaction retrieval engine relying on an inverted file and Boolean queries for retrieving transaction IDs from a database, and a set of sanitizing algorithms. One major novelty with our approach is that we take into account the impact of our sanitization not only on hiding the patterns that should be hidden but also on hiding legitimate patterns that should not be hidden. Thus, our framework tries to find a balance between privacy and disclosure of information by attempting to minimize the impact on the sanitized transactions. Other approaches presented in the literature focus on the hiding of restrictive patterns but do not study the effect of their sanitization on accidentally concealing legitimate patterns or even generating artifact patterns.

The main contributions of this paper are the following: (1) the design and implementation of the framework; (2) algorithms for cleansing a transactional database; (3) a taxonomy for our algorithms; and (4) performance measures for mining frequent patterns that quantify the fraction of mining patterns which are preserved, under a certain privacy threshold ψ, after sanitizing a database.

This paper is organized as follows. In Section 2, we provide some basic concepts on transactional

databases, frequent patterns and association rules, and the process of sanitizing a database. In addition, the definition of the research problem is given. We present our framework in detail in Section 3. In Section 4, we introduce our transaction sanitizing algorithms. In Section 5, we present the experimental results and discussion. Related work is reviewed in Section 6. Finally, Section 7 presents our conclusions and a discussion of future work.

2 Basic Concepts

In this section, we briefly review some concepts related to transactional databases, frequent patterns, and association rules. In addition, we present the formulation of the research problem addressed in this paper and describe the process of sanitizing a transactional database.

2.1 Transactional Databases

A transactional database is a relation consisting of transactions in which each transaction t is characterized by an ordered pair, defined as $t = \langle TID,\ list\_of\_elements \rangle$, where TID is a unique transaction identifier number and *list_of_items* represents a list of items making up the transactions [10]. For instance, in market basket data, a transactional database is composed of business transactions in which the list of elements represents items purchased in a store.

In general, transactional databases require the transforming of relational data into a single table, and miners then are able to look for patterns in such an engineered table. Based on such a data structure, data mining systems for transactional data can answer queries such as "*identify the sets of items which are frequently sold together*" that a regular data retrieval system cannot.

A transactional database may have additional tables associated with it. Such tables may contain information concerning sales, such as the date of the transaction, the customer's ID number, the ID of the sales person, the branch at which the sale occurred [17], and so on. A sample transactional database, composed of six transactions, is shown in Fig 3 (Section 3.2).

2.2 The Basics of Mining Frequent Patterns and Association Rules

The discovery of the recurrent patterns in large transactional databases has become one of the main topics in data mining. In its simplest form, the task of finding frequent patterns can be viewed as the process of discovering all item sets, i.e., all combinations of items that are found in a sufficient number of examples, given a frequency threshold σ. If the frequency threshold is low, then there may be many frequent patterns in the answer set.

The items in a frequent pattern are Boolean, i.e., items are either present or absent. For this reason, a transactional database may be represented by a sparse matrix in which the rows correspond to transactions and the columns correspond to the items available in one store. If the element (i, j) is 1, this indicates that customer i purchased item j, while 0 indicates that the item j was not purchased.

A data mining system is able to generate thousands or even millions of frequent patterns. For instance, in a dataset with m items, 2^m patterns are possible. So this task is exhaustive since there is

an exponential number of patterns generated from the variables present in the whole data. In market basket applications, for example, the number of frequent patterns tends to be quite large.

In most cases, only a small fraction of the patterns generated would actually be of interest to any miner. The research challenge in discovering frequent patterns is to find useful and valuable ones. Han and Kamber [17] suggested that a pattern is interesting if it satisfies the following conditions: (a) it is easily understood by humans; (b) it is valid on new or test data with some degree of certainty; (c) it is potentially useful; (d) it is novel; and (e) it meets a frequency threshold controlled by the user.

2.3 The Basics of Frequent Patterns and Association Rules

When the frequent patterns are known, finding association rules is simple. Association rules provide a very simple but useful form of rule patterns for data mining. A rule consists of a left-hand side proposition (the antecedent or condition) and a right-hand side (the consequent). Both the left and right-hand side consist of Boolean statements (or propositions). The rules state that if the left-hand side is true, then the right-hand side is also true.

More formally, association rules are defined as follows: Let $I = \{i_1,...,i_n\}$ be a set of literals, called items. Let D be a database of transactions in which each transaction t is an itemset such that $t \subseteq I$. A unique identifier, called *TID*, is associated with each transaction. A transaction t supports X, a set of items in I, if $X \subset t$. An association rule is an implication of the form $X \Rightarrow Y$, where $X \subset I$, $Y \subset I$ and $X \bigcap Y = \emptyset$. Thus, we say that a rule $X \Rightarrow Y$ holds in the database D with *confidence* φ if $\frac{|X \bigcup Y|}{|X|} \geqslant \varphi$ where $|A|$ is the number of occurrences of the set of items A in the set of transactions D. Similarly, we say that a rule $X \Rightarrow Y$ holds in the database D with *support* σ if $\frac{|X \bigcup Y|}{|N|} \geqslant \sigma$ where N is the number of transactions in D.

As stated before, association rule mining algorithms rely on support and confidence and mainly have two major phases: (1) based on a support σ set by the user, frequent itemsets are determined through consecutive scans of the database; (2) strong association rules are derived from the frequent item sets and constrained by a minimum confidence φ also set by the user. Since the main challenge is the discovery of the frequent itemsets, we consider only this second phase in our analysis.

One well-known algorithm for association rules is Apriori [2, 3]. This algorithm constructs a candidate set of large itemsets, counts the number of occurrences of each candidate itemset, and then determines large itemsets based on a predetermined minimum support. The main idea behind the apriori algorithm is the a priori property, where every subset of a frequent itemset must also be a frequent itemset. Other algorithms for finding association rules involve some variations to reduce the number of data scans required, such as hashing and transaction reduction, or even partitioning the data and sampling the data. Examples of such algorithms include the hash-based algorithm for association rules introduced by Park et al. [20], the sampling algorithm for association rules proposed by Toivonen [23], and the binary tree algorithm proposed by Gyenesei [16].

2.4 Privacy Preservation: Problem Definition

In this paper we focus on the discovery of frequent patterns. Unlike association rules that rely on support and confidence, frequent patterns satisfy a support threshold σ only.

In this work, our goal is to hide a group of frequent patterns which contain highly sensitive knowledge. We refer to these frequent patterns as restrictive patterns, and we define them as follows:

Definition 1 (Restrictive Patterns) *Let D be a transactional database, P be a set of all frequent patterns that can be mined from D, and $Rules_H$ be a set of decision support rules that need to be hidden according to some security policies. A set of patterns, denoted by R_P, is said to be restrictive if $R_P \subset P$ and if and only if R_P would derive the set $Rules_H$. $\tilde{}R_P$ is the set of non-restrictive patterns such that $\tilde{}R_P \cup R_P = P$.*

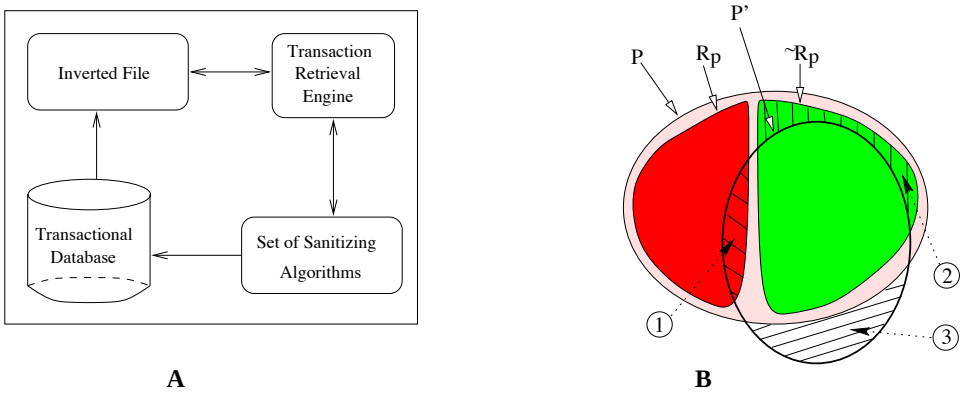

Figure 1: (A): Privacy Preservation Framework. (B):Visual representation of restrictive and non restrictive patterns and the patterns effectively discovered after transaction sanitization

Figure 1B illustrates the relationship between the set P of all patterns in the databse D, the restrictive and non-restrictive patterns, as well as the set P' of patterns discovered from the sanitized database D'. 1, 2, and 3 are potential problems that represent the restrictive patterns that are failed to be hidden, the legitimate patterns accidentally missed, and the artificial patterns created by the sanitization process. These are explained in Section 5 in the discussion of how to measure the effectiveness of our algorithms.

A group of restrictive patterns is mined from a database D based on a special group of transactions. We refer to these transactions as sensitive transactions and define them as follows.

Definition 2 (Sensitive Transactions) *Let T be a set of all transactions in a transactional database D and R_P be a set of restrictive patterns mined from D. A set of transactions is said to be sensitive, denoted by S_T, if $S_T \subset T$ and if and only if all restrictive patterns can be mined from S_T and only from S_T.*

One simple and effective way to hide some restrictive patterns is to decrease their support in a given database. This procedure of altering the transactions is called the sanitization process [5]. To

do so, a small number of transactions have to be modified by deleting one or more items from them or even changing items in transactions such as in [5]. On one hand, this approach slightly modifies some data, but this is perfectly acceptable in some real applications. On the other hand, such an approach must hold the following restrictions: (1) the impact on the data in D has to be minimal and (2) an appropriate balance between a need for privacy and knowledge discovery must be guaranteed.

The specific problem addressed in this paper can be stated as follows: If D is the source database of transactions and P is a set of relevant association patterns that could be mined from D, the goal is to transform D into a database D' so that the most frequent patterns in P can still be mined from D' while others will be hidden. In this case, D' becomes the released database.

2.5 The Sanitization Process

As mentioned in the previous section, the goal of the sanitization process is to hide some restrictive patterns that contain highly sensitive knowledge. Figure 2 illustrates the different steps in this process. In the first step, the set P of all patterns from D is identified. The second step distinguishes restricted patterns R_p from the non-restrictive patterns $\tilde{}R_P$ by applying some security policies. It should be noted that what constitutes restrictive patterns depends on the application and the importance of these patterns in a decision process. In Step 3, sensitive transactions are identified within D. In our approach, we use a very efficient retrieval mechanism called the transaction retrieval engine (discussed in Section 4.2) to speed up the process of finding the sensitive transactions. Finally, Step 4 is dedicated to the alteration of these sensitive transactions to produce the sanitized database D'. In our framework, the process of modifying such transactions satisfies a privacy threshold ψ controlled by the user. This threshold basically expresses how relaxed the privacy preserving mechanisms should be. When $\psi = 0\%$, no restrictive patterns are allowed to be discovered. When $\psi = 100\%$, there are no restrictions on the restrictive patterns. The advantage of having this threshold is that it enables a compromise to be found between hiding patterns while missing legitimate ones and finding all legitimate patterns but uncovering restrictive ones. This is indeed the balance between privacy and the disclosure of information.

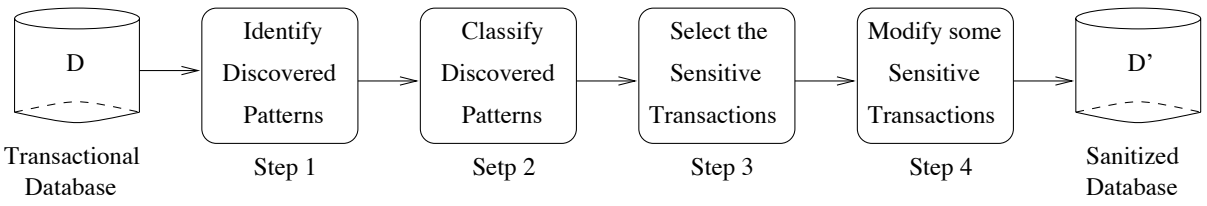

Figure 2: The Sanitization Process

3 The Framework for Privacy Preservation

As depicted in Figure 1(A), our framework encompasses a transactional database (modeled into a document database), an inverted file, a set of sanitizing algorithms used for hiding restrictive patterns from the database, and a transaction retrieval engine for fast retrieval of transactions.

3.1 Mapping a Transactional Database into a Text Document Database

In our approach, we build an index schema to speed up the sanitizing process of a transactional database. To accomplish this, we map a transactional database into a text document database.

A text database is a file that contains word descriptions for objects. A text database is considered to be composed of two fundamental units: the *document* and the *term*. A *document* can be a traditional document, such as a book, product specifications, summary reports, or bug reports. Other text documents include chapters, sections, paragraphs, or even e-mail messages and Web pages. On the other hand, a *term* can be a word, word-pair, or phrase within a document, such as the word *computer* or word-pair *computer science*.

For our purpose, we define a text database as a collection of transactions in which each transaction is a document and terms are represented by items. This model preserves all the information and provides the basis for our indexing, borrowing from the information retrieval domain.

3.2 The Inverted File Index

Database design and configuration become increasingly important as a database's size and number of concurrent users grow. An application that begins small might grow significantly over time, and its database must grow as well. For instance, in a transactional database, there are thousands or even millions of transactions. For this reason, the design of an index schema will be very important to deal with emerging performance problems.

Considering that in our approach we map a transactional database into a text database, we need to look for an index schema to deal with a text database. One very efficient strategy for indexing a text database is an inverted file [6]. An inverted file, a structure comprising the *vocabulary* and the *occurrences*, is a word-oriented mechanism for indexing a text collection with the purpose of speeding up the searching task. The vocabulary is the set of all different words in the text, whereas the occurrences represent a data structure which assigns to each text word the list of documents in which such an word is presented.

In our framework, the inverted file's vocabulary is composed of all the different items in the transactional database, and for each item there is a corresponding list of transaction IDs in which the item is present. Figure 3 shows an example of an inverted file corresponding to the sample transactional database shown in the figure.

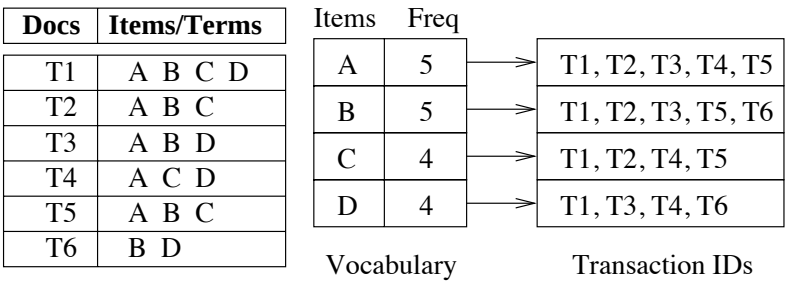

Figure 3: An example of transactions modeled by documents and the corresponding inverted file

As can be seen, this schema saves space because all the occurrences of the same item in a transaction are referenced only once. In addition, the pointers may be smaller because there are fewer transactions than item positions. For single-item queries, it is not necessary to access the text because when a item appears in a transaction, the algorithm retrieves the entire transaction.

We implemented the vocabulary based on a perfect hash table [15], with no collision, insertion, or deletion. For a given item, one access suffices to find the list of all transaction IDs that contain the item. The occurrences with transaction IDs are created and simultaneously sorted in ascending order of transaction IDs. Thus, to search for a transaction ID of a particular item, we use a binary search in which, in the worst case, the access time is $O(logN)$, where N is the number of transaction IDs in the occurrences.

The sequence of steps that we need in order to build the inverted file proposed in our framework, scanning the transactional database only once, are the following: (1) building the vocabulary composed of all different items and computing the frequency of each item as we read each transaction and (2) building and/or updating the occurrence list for each different item in one transaction.

3.3 The Transaction Retrieval Engine

To search for sensitive transactions in the transactional database, it is necessary to access, manipulate, and query transaction IDs. The transaction retrieval engine performs these tasks. It accepts requests for transactions from a sanitizing algorithm, determines how these requests can be filled (consulting the inverted file), processes the queries using a query language based on the Boolean model, and returns the results to the sanitizing algorithm. The process of searching for sensitive transactions through the transactional database works on the inverted file. In general, this process follows three steps: (1) *Vocabulary search*: each restrictive pattern is split into single items. Isolated items are transformed into basic queries to the inverted index; (2) *Retrieval of transactions*: The lists of all transaction IDs of transactions containing each individual item respectively are retrieved; and (3) *Intersections of transaction lists*: The lists of transactions of all individual items in each restrictive pattern are intersected using a conjunctive Boolean operator on the query tree to find the sensitive transactions containing a given restrictive pattern.

The transaction retrieval engine designed for our framework relies on the Boolean model. The selection of such a model was influenced by the following reasons: First, the Boolean model is a simple information retrieval model based on set theory and Boolean algebra. However, because we use only AND operations for retrieving sensitive transactions, our transaction retrieval engine is confined to this Boolean operation. Second, Boolean expressions have a precise semantic since the index terms are present or absent in a transaction. Third, the Boolean operation AND is very simple, intuitive, and easy to implement.

To illustrate how our transaction retrieval engine works, Figure 4 shows the search for the restrictive pattern ACD in the sample database depicted in Figure 3. As illustrated, a Boolean query has a syntax composed of items (i.e., basic queries) represented by a set of transactions as well as the Boolean AND operator which works on their operands (which are a set of transactions) and delivers a set of sensitive

transactions. Since the scheme is in general compositional (i.e., operators can be composed over the results of other operators), a query syntax tree is naturally defined in which the leaves correspond to the basic queries and the internal nodes to the operators.

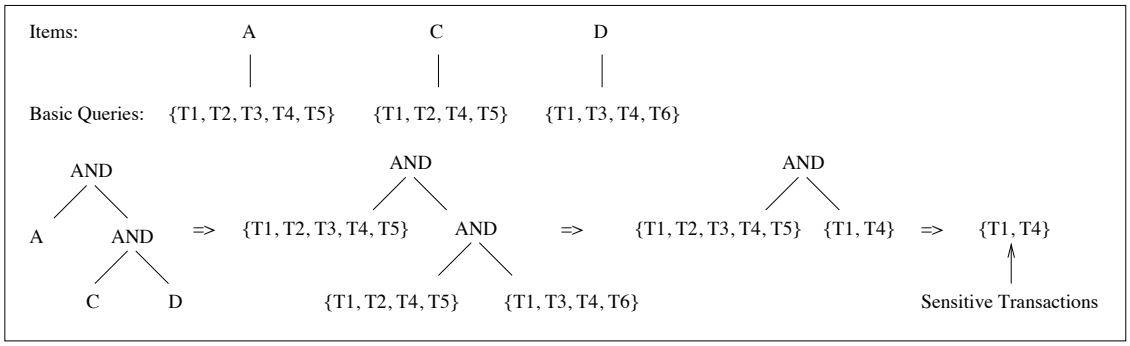

Figure 4: Processing the internal nodes of the query syntax tree

4 The Sanitization Algorithms

In our taxonomy for transactional database sanitizing algorithms for the purpose of information hiding, we distinguish the algorithms that solely remove information from the algorithms that modify existing information. The first algorithms only reduce the support of some items, while the second may increase the support of some items. We believe that the algorithms that remove information have a smaller impact on the database since they do not generate artifacts such as illegal patterns that would not exist had the sanitizing not happened. Among the approaches that remove information only, we distinguish the pattern restriction-based approaches that remove complete restrictive patterns from the sensitive transactions and the item restriction-based approaches that selectively remove some items from sensitive transactions, as can be seen in Figure 5. The pattern restrictive-based approaches have a bigger impact on the database as more legal patterns may end-up hidden along with the restricted patterns.

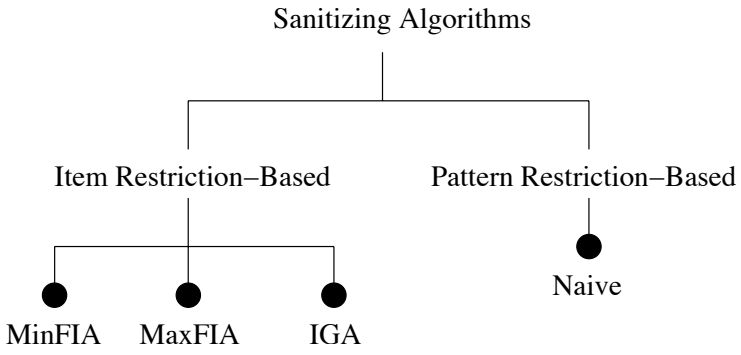

Figure 5: A Taxonomy of Sanitizing Algorithms

In this section, we present the idea behind our hiding strategies. For each algorithm, the inputs are a transactional database D, a set of restrictive patterns R_P, and a privacy threshold ψ, while the

output is the sanitized database D'.

To sanitize a database, each sanitizing algorithm requires an additional scan over the original database D in order to alter some sensitive transactions while keeping the other transactions intact. An initial scan is necessary to build the inverted index.

In most cases, a sensitive transaction contains more than one restrictive pattern. We refer to these transactions as *conflicting transactions* since modifying one of them causes an impact on other restrictive patterns or even on non-restrictive ones. The *degree of conflict* of a sensitive transaction is defined as the number of restrictive patterns that can be mined from the sensitive transaction.

To illustrate the presented concepts, let us consider the sample transactional database in Figure 3. Suppose that we have a set of restrictive patterns $R_P = \{\text{ABD, ACD}\}$. This example yields the following results. The sensitive transactions S_T containing the restrictive patterns are {T1, T3, T4}. The degrees of conflict for the transactions T1, T3 and T4 are 2, 1 and 1 respectively. Thus, the only conflicting transaction is T1, which covers both restrictive patterns at the same time. An important observation here is that any pattern that contains a restrictive pattern is also a restrictive pattern. Hence, if ABD is a restricted pattern but not ACD as above, the pattern ABCD will also be restrictive since it contains ABD. This is because if ABCD is discovered as a frequent pattern, it is straight forward to conclude that ABD is also frequent, which should not be disclosed.

All our item restriction-based and pattern restriction-based algorithms have essentially four major steps:

1. Identify sensitive transactions for each restrictive pattern after building an inverted index where the vocabulary is the set of all items in the database and the occurences are the transaction IDs. In our algorithms, we also generated an inverted index where the vocabulary is the set of restrictive patterns and the occurences are the IDs of sensitive transactions.

2. For each restrictive pattern, identify a candidate item that should be eliminated from the sensitive transactions. This candidate item is called the *victim item*.

3. Based on the privacy threshold ψ, calculate for each restrictive pattern the number of sensitive transactions that should be sanitized.

4. Based on the number found in step 3, identify for each restrictive pattern the sensitive transactions that have to be sanitized and remove the victim item from them.

Most of our sanitizing algorithms mainly differ in step 2 in the way they identify a victim item to remove from the sensitive transactions for each restrictive pattern, and in step 4 where the sensitive transactions to be sanitized are selected. Steps 1 and 3 remain essentially the same for all approaches.

4.1 The Naïve Algorithm

The main idea behind the Naïve Algorithm, denoted by NA, is to select all items in a given restrictive pattern as victims. The rationale behind this selection is that by removing from the sensitive transactions the items of a restrictive pattern, such a pattern will be hidden. If a sensitive transaction contains

exactly the same items as a restrictive pattern, the Naïve Algorithm removes all items of this transaction except for the item with the highest frequency in the database. Because one item must be kept, the number of transactions is not modified.

Selecting the sensitive transactions to sanitize is based simply on their degree of conflict. Given the number of sensitive transactions to alter, based on ψ, this approach selects for each restrictive pattern the transactions with the smallest degree of conflict. The rationale is, as above, to minimize the impact of the sanitization on the discovery of the legitimate patterns.

The sketch of the Naïve Algorithm is presented below:

Naive_Algorithm
Input: D, R_P, ψ
Output: D'
Step 1. For each restrictive pattern $rp_i \in R_P$ do
1. $T[rp_i] \leftarrow$Find_Sensitive_Transactions(rp_i, D);
Step 2. For each restrictive pattern $rp_i \in R_P$ do
1. $Victims_{rp_i} \leftarrow \forall\ item_k$ such that $item_k \in rp_i$
Step 3. For each restrictive pattern $rp_i \in R_P$ do
1. $NumbTrans_{rp_i} \leftarrow |T[rp_i]| \times (1 - \psi)$ // $|T[rp_i]|$ is the number of sensitive transactions for rp_i
Step 4. $D' \leftarrow D$
For each restrictive pattern $rp_i \in R_P$ do
1. Sort_Transactions($T[rp_i]$); //in ascending order of degree of conflict
2. $TransToSanitize \leftarrow$ Select first $NumbTrans_{rp_i}$ transactions from $T[rp_i]$
3. in D' foreach transaction $t \in TransToSanitize$ do
3.1. $t \leftarrow (t - Victims_{rp_i})$
End

The four steps in this algorithm correspond to the four steps described above for all pattern restriction-based algorithms. The first step builds an inverted index of the item in D in one scan of the database. As illustrated in the example in Figure 3, the support of each item in the database is also calculated during this scan and attached to the respective items in the inverted index. This support of the items is used in step 2 to identify the victim items $Victims_{rp_i}$ for each restrictive pattern. For the Naïve algorithm, all items in a given restrictive pattern are selected. Line 1 in step 3 shows that ψ is used to compute the number $NumbTrans_{rp_i}$ of transactions to sanitize. This means that the threshold ψ is actually a measure of the impact of the sanitization rather than a direct measure of the restricted patterns to hide or disclose. Indirectly, ψ does have an influence on the hiding or disclosure of restricted patterns. There is actually only one scan of the database in the implementation of step 4. Transactions that do not need sanitization are directly copied from D to D', while the others are sanitized before being copied to D'. In our implementation, the sensitive transactions to be cleansed are first marked before the database scan for copying. The selection of the sensitive transactions to sanitize, $TransToSanitize$, is based on their degree of conflict, hence the sorting in line 1 of step 4. When a transaction is selected for sanitization, the victim items are removed from it (line 3.1 in step 4).

Theorem 1 *The running time of the Naïve Algorithm is $O(n_1 \times NlogN)$, where n_1 is the number of*

restrictive patterns in D and N is the number of transactions in D.

Proof. Let D be the source database, N the number of transactions in D, n_1 the number of restrictive patterns in D, n_2 the maximum number of items in a restrictive pattern, and let n_3 the maximum number of items in one transaction. Assume that the inverted file is stored in memory in which the vocabulary is a hash table containing all distinct items in D, their frequencies, and that for each item, there is a corresponding list of transaction IDs.

In step 1, for each restrictive pattern, the algorithm searches for the transaction IDs corresponding to each item and makes an AND operation on many sorted lists, taking 2 lists at a time. For the first AND operation, the algorithm takes $O(N)$, assuming that each list contains all N transaction IDs, in the worst case. The resulting list has to be AND-performed with the third item's list, and this process continues until the last item's list has been considered. This entire process encompasses $n_2 - 1$ AND operations, so that the running time for each pattern takes $O((n_2 - 1) \times N)$. Because step 1 is executed n_1 times, the complexity of this loop is $O(n_1 \times (n_2 - 1) \times N)$. Considering that $n_2 \ll n_1$ and $n_2 \ll N$, the running time for step 1 can be simplified to $O(n_1 \times N)$.

Steps 2 is also executed n_1 times. For each restrictive pattern rp_i, all items in rp_i are selected as victim items, so that this operation takes $O(n_2)$. Thus, step 2 takes $O(n_1 \times n_2)$.

Step 3 contains a straightforward computation and takes $O(1)$ per restrictive pattern since the algorithm simply selects the number of sensitive transactions to be touched. Because this step is executed n_1 times, the running time for step 3 takes $O(n_1)$.

Step 4 is the most expensive step of the algorithm. In line 1 of step 4, the algorithm sorts all sensitive transactions of each restrictive pattern by frequency. Considering that in the worst case all transactions in D are sensitive, Line 1 takes $O(n_1 \times NlogN)$ since this operation is executed n_1 times. In Line 2, in the worst case, all transactions in D are sensitive and are stored in the data structure $TransToSanitize$ to be sanitized. So Line 2 takes $O(n_1 \times N)$. In Line 3, the algorithm cleanses at most $n_3 - 1$ items in each sensitive transaction since one item is required to keep the number of transactions in D. Thus, Line 3 takes $O(n_1 \times (n_3 - 1))$. The running time for Step 4 is $O(n_1 \times NlogN + n_1 \times N + n_1 \times (n_3 - 1))$. When N and n_1 are large, $n_1 \times NlogN$ grows faster than $n_1 \times N$ and $n_1 \times (n_3 - 1)$. Thus, the running time for step 4 takes $O(n_1 \times NlogN)$.

The running time of the Naïve algorithm is the sum of running times for each step executed, i.e., $O(n_1 \times N + n_1 \times n_2 + n_1 + n_1 \times NlogN)$. When N is large, $n_1 \times NlogN$ grows faster than $n_1 \times N$ and $NlogN$, $n_1 \times n_2$ and n_1. Thus, the running time of the Naïve Algorithm takes $O(n_1 \times NlogN)$, which is almost linear. □

4.2 The Minimum Frequency Item Algorithm

The main idea behind the Minimum Frequency Item Algorithm, denoted by MinFIA, is to select as a victim item, for a given restrictive pattern, the restrictive pattern item with the smallest support in the database. The rationale behind this selection is that removing the item from the sensitive transactions with the smallest support will have the smallest impact on the database and the legitimate patterns to

be discovered. Again, selecting the sensitive transactions to sanitize is simply based on their degree of conflict. Given the number of sensitive transactions to alter, based on ψ, this approach selects for each restrictive pattern the transactions with the smallest degree of conflict.

The sketch of the Minimum Frequency Item Algorithm is presented below:

Minimum_Frequency_Item_Algorithm
Input: D, R_P, ψ
Output: D'
Step 1. For each restrictive pattern $rp_i \in R_P$ do
1. $T[rp_i] \leftarrow$Find_Sensitive_Transactions(rp_i, D);
Step 2. For each restrictive pattern $rp_i \in R_P$ do
1. $Victim_{rp_i} \leftarrow item_v$ such that $item_v \in rp_i$ and
$\forall\ item_k \in rp_i$ support($item_k$, D) $\geq$ support($item_v$, D)
Step 3. For each restrictive pattern $rp_i \in R_P$ do
1. $NumbTrans_{rp_i} \leftarrow |T[rp_i]| \times (1 - \psi)$ // $|T[rp_i]|$ is the number of sensitive transactions for rp_i
Step 4. $D' \leftarrow D$
For each restrictive pattern $rp_i \in R_P$ do
1. Sort_Transactions($T[rp_i]$); //in ascending order of degree of conflict
2. $TransToSanitize \leftarrow$ Select first $NumbTrans_{rp_i}$ transactions from $T[rp_i]$
3. in D' foreach transaction $t \in TransToSanitize$ do
3.1. $t \leftarrow (t - Victim_{rp_i})$
End

The four steps of this algorithm correspond to those in the Naïve Algorithm. The only difference is that the Minimum Frequency Item Algorithm selects exactly one victim item, as aforementioned.

Theorem 2 *The running time of the Minimum Frequency Item Algorithm is $O(n_1 \times NlogN)$, where n_1 is the number of restrictive patterns in D and N is the number of transactions in D.*

Proof. Let D be the source database, N the number of transactions in D, n_1 the number of restrictive patterns in D, n_2 the maximum number of items in a restrictive pattern, and let n_3 the maximum number of items in one transaction. Assume that the inverted file is stored in memory in which the vocabulary is a hash table containing all distinct items in D and their frequencies, and that for each item, there is a corresponding list of transaction IDs.

From Theorem 1, we know that step 1 takes $O(n_1 \times N)$, step 3 takes $O(n_1 \times N)$, and step 4 takes $O(n_1 \times NlogN)$. So we have to analyze the running time for step 2.

In Steps 2, for each restrictive pattern rp_i, the algorithm sorts the corresponding items by frequency in order to select the one with the least frequency. This procedure takes $O(n_2)$. Because the loop in this step is executed n_1 times, step 2 takes $O(n_1 \times n_2)$.

The running time of the Minimum Frequency Item Algorithm is the sum of running times for each step executed, i.e., $O(n_1 \times N + n_1 \times n_2 + n_1 + n_1 \times NlogN)$. When N is large, $n_1 \times NlogN$ grows faster than $n_1 \times N$ and $NlogN$, $n_1 \times n_2$ and n_1. Thus, the running time of the Minimum Frequency Item Algorithm takes $O(n_1 \times NlogN)$. $\square$

4.3 The Maximum Frequency Item Algorithm

Unlike the Minimum Frequency Item Algorithm, the idea behind the Maximum Frequency Item Algorithm, denoted by MaxFIA, is to select as a victim item, for a given restrictive pattern, the restrictive pattern item with the maximum support in the database. After identifying a victim item for each restrictive pattern, the algorithm selects the sensitive transactions to sanitize based on their degree of conflict. The sketch of the Maximum Frequency Item Algorithm is presented below:

Maximum_Frequency_Item_Algorithm
Input: D, R_P, ψ
Output: D'
Step 1. For each restrictive pattern $rp_i \in R_P$ do
1. $T[rp_i] \leftarrow$Find_Sensitive_Transactions(rp_i, D);
Step 2. For each restrictive pattern $rp_i \in R_P$ do
1. $Victim_{rp_i} \leftarrow item_v$ such that $item_v \in rp_i$ and
$\forall\ item_k \in rp_i$ support($item_k$, D) $\geq$ support($item_v$, D)
Step 3. For each restrictive pattern $rp_i \in R_P$ do
1. $NumbTrans_{rp_i} \leftarrow |T[rp_i]| \times (1 - \psi)$ // $|T[rp_i]|$ is the number of sensitive transactions for rp_i
Step 4. $D' \leftarrow D$
For each restrictive pattern $rp_i \in R_P$ do
1. Sort_Transactions($T[rp_i]$); //in ascending order of degree of conflict
2. $TransToSanitize \leftarrow$ Select first $NumbTrans_{rp_i}$ transactions from $T[rp_i]$
3. in D' foreach transaction $t \in TransToSanitize$ do
3.1. $t \leftarrow (t - Victim_{rp_i})$
End

The steps of this algorithm are very similar to those in the Minimum Frequency Item Algorithm. The only difference is in regard to the selection of the victim item. Apart from this, the running time for the Maximum Frequency Item Algorithm is $O(n_1 \times NlogN)$. This proof is similar to that in Theorem 2.

4.4 The Item Grouping Algorithm

The main idea behind the Item Grouping Algorithm, denoted by IGA, is to group restricted patterns in groups of patterns sharing the same itemsets. If two restrictive patterns intersect, by sanitizing the conflicting sensitive transactions containing both restrictive patterns, one would take care of hiding these two restrictive patterns at once and consequently reduce the impact on the released database. However, clustering the restrictive patterns based on the intersections between patterns leads to groups that overlap since the intersection of itemsets is not transitive. By solving the overlap between clusters and thus isolating the groups, we can use a representative of the itemset linking the restrictive patterns in the same group as a victim item for all patterns in the group. By removing the victim item from the sensitive transactions related to the patterns in the group, all sensitive patterns in the group will be hidden in one step. This again will minimize the impact on the database and reduce the potential accidental hiding of legitimate patterns.

The sketch of the Item Algorithm is presented below:

Item_Grouping_Algorithm
Input: D, R_P, ψ
Output: D'
Step 1. For each restrictive pattern $rp_i \in R_P$ do
1. $T[rp_i] \leftarrow$Find_Sensitive_Transactions(rp_i, D);
Step 2.
1.Group restrictive patterns in a set of groups GP such that $\forall\ G \in GP, \forall\ rp_i, rp_j \in G$, rp_i and rp_j share the same itemset I. Give the class label α to G such that $\alpha \in I$ and $\forall \beta \in I$, support(α, D) $\leq$ support(β, D).
2. Order the groups in GP by size in terms of number of restrictive patterns in the group.
3. Compare groups pairwise G_i and G_j starting with the largest. For all $rp_k \in G_i \cap G_j$ do
3.1. if size(G_i) $\neq$ size(G_j) then remove rp_k from smallest(G_i, G_j)
3.2. else remove rp_k from group with class label α such that support(α, D) $\leq$ support(β, D) and α, β are class labels of either G_i or G_j
4. For each restrictive pattern $rp_i \in R_P$ do
4.1. $Victim_{rp_i} \leftarrow \alpha$ such that α is the class label of G and $rp_i \in G$
Step 3. For each restrictive pattern $rp_i \in R_P$ do
1. $NumbTrans_{rp_i} \leftarrow |T[rp_i]| \times (1 - \psi)$ // $|T[rp_i]|$ is the number of sensitive transactions for rp_i
Step 4. $D' \leftarrow D$
For each restrictive pattern $rp_i \in R_P$ do
1. Sort_Transactions($T[rp_i]$); //in descending order of degree of conflict
2. $TransToSanitize \leftarrow$ Select first $NumbTrans_{rp_i}$ transactions from $T[rp_i]$
3. in D' foreach transaction $t \in TransToSanitize$ do
3.1. $t \leftarrow (t - Victim_{rp_i})$
End

Steps 1 and 3 are identical to the respective steps in the previous algorithms. Step 4 is slightly different from Step 4 in the previous algorithms since the sensitive transactions are now ordered in descending order of their degree of conflict so that more conflicting transactions are selected for sanitization instead of non conflicting ones. The reason is that since the victim item now represents a set of restrictive patterns (from the same group), sanitizing a conflicting transaction will allow many restrictive patterns to be taken care of at once per sanitized transaction. The goal of step 2 is to identify a victim item per restrictive pattern. This is done by first clustering restrictive patterns in a set of overlapping groups GP (task 1), such that all restrictive patterns in the same group G share some items that are the same. The shared items are the class label of the group. For example, the patterns "ABC" and "ABD" would be in the same group labelled either A or B (depending on support of A and B - task 1, line 3). However, "ABC" could also be in another group if there was one where restrictive patterns shared "C." Tasks 2 and 3 identify such overlap between groups and eliminate it by favouring larger groups or groups with a class label with higher support in the database.

Theorem 3 *The running time of the Item Grouping Algorithm is $O(n_1 \times NlogN)$, where n_1 is the number of restrictive patterns in D and N is the number of transactions in D.*

Proof. Let D be the source database, N the number of transactions in D, n_1 the number of restrictive patterns in D, n_2 the maximum number of items in a restrictive pattern, n_3 the maximum number of items in one transaction, and let C_1 the number of clusters in V_c.

From Theorem 1, we know that the running times of steps 1, 3, and 4 are $O(n_1 \times N)$, $O(n_1 \times n_2)$, and $O(n_1 \times NlogN)$ respectively. So we have to analyze the running time of step 2.

In step 2, the following computations are performed: (a) Sort the clusters by descending order of the number of patterns. This takes $O(C_1 \times logC_1)$; (b) Given that the clusters are sorted by size, the clusters with overlapped patterns can be removed. This process requires an entire scan over the clusters and takes $O(C_1)$; (c) Now, the algorithm deals with clusters that have the same size, share more than one item and have the same patterns. To select the clusters with the same size takes $O(C_1)$ in the worst case; (d) The last computation is to check the support of the items shared by the same clusters in order to select the cluster with the least support item. Doing so implies one access to the inverted file for the shared items in each cluster, since the inverted file is built on a hash table. In the worst case, all clusters in V_c have the same size. So the last computation takes $O(C_1)$. Thus, the total cost of step 2 is $O(C_1 \times logC_1 + C_1 + C_1 + C_1)$. When C_1 is large, $C_1 \times logC_1$ grows faster than C_1, so this cost can be simplified to $O(C_1 \times logC_1)$.

The running time of the Item Grouping Algorithm is the sum of the running times for each step executed, i.e., $O(n_1 \times N + C_1 logC_1 + n_1 \times NlogN + NlogN + NlogN)$. When N is large, $n_1 \times NlogN$ grows faster than $n_1 \times N$, $C_1 \times logC_1$, and $NlogN$. Thus, the running time of the Item Grouping Algorithm takes $O(n_1 \times NlogN)$, and this completes the proof of Theorem 3. □

5 Experimental Results

We performed two series of experiments: the first to measure the effectiveness of our sanitization algorithms and the second to measure the efficiency and scalability of the algorithms. All the experiments were conducted on a PC, AMD Athlon 1900/1600 (SPEC CFP2000 588), with 1.2 GB of RAM running a Linux operating system. To measure the effectiveness of the algorithms, we used a dataset generated by the IBM synthetic data generator to generate a dataset containing 500 different items, with 100K transactions in which the minimum size per transaction is 40 items. The effectiveness is measured in terms of the number of restrictive patterns effectively hidden, as well as the proportion of legitimate patterns accidentally hidden due to the sanitization. Some types of sanitization could also lead to the discovery of non-existing patterns in the original database D, but not in D' the sanitized database. As depicted in Figure 1 of Section 2.4, if P is the set of all mining patterns in the original database D and we have some restricted patterns R_P, the legitimate patterns are $\tilde{}R_P$ such that $P = \tilde{}R_P \cup R_P$. After sanitization, the patterns P' that should be discovered from the sanitized database D' should be equal to $\tilde{}R_P$ and only $\tilde{}R_P$. However, this is not the case, and we have three possible problems, as illustrated in Figure 1.

Problem 1 occurs when some restrictive patterns are discovered. We call this problem ***Hiding Failure***, and it is measured in terms of the percentage of restrictive patterns that are discovered from D'. Ideally, the hiding failure should be 0%. The hiding failure is measured by $HF = \frac{\#R_P(D')}{\#R_P(D)}$ where $\#R_P(X)$ denotes the number of restrictive patterns discovered from database X. In our framework, the proportion of restrictive patterns that are nevertheless discovered from the sanitized database can

be controlled with the privacy threshold ψ, and this proportion ranges from 0% to 100%. Note that ψ does not control the *hiding failure* directly, but indirectly by controlling the proportion of sensitive transactions to be sanitized for each restrictive pattern.

Problem 2 occurs when some legitimate patterns are hidden by accident. This happens when some non-restrictive patterns lose support in the database due to the sanitization process. We call this problem ***Misses Cost***, and it is measured in terms of the percentage of legitimate patterns that are not discovered from D'. In the best case, this should also be 0%. The misses cost is calculated as follows: $MC = \frac{\#\sim R_P(D) - \#\sim R_P(D')}{\#\sim R_P(D)}$ where $\# \sim R_P(X)$ denotes the number of non-restrictive patterns discovered from database X. Notice that there is a compromise between the misses cost and the hiding failure. The more restrictive patterns we hide, the more legitimate patterns we miss. This is basically the justification for our privacy threshold ψ, which with tuning, allows us to find the balance between privacy and disclosure of information whenever the application permits it.

Problem 3 occurs when some artificial patterns are generated from D' as a product of the sanitization process. We call this problem **Artifactual Patterns**, and it is measured in terms of the percentage of the discovered patterns that are artifacts. This is measured as: $AP = \frac{|P'| - |P \cap P'|}{|P'|}$ where $|X|$ denotes the cardinality of X. One may claim that when we decrease the frequencies of some items, the relative frequencies in the database may be modified by the sanitization process, and new patterns may emerge. However, in our experiments, AP was always 0% with all algorithms regardless of the values of ψ.

5.1 Measuring effectiveness

We selected for our experiments a set of ten restrictive patterns from the dataset ranging from two to five items in length, with support ranging from 20% to 40% in the database.

Note that the higher the support for the restrictive patterns, the larger the number of sensitive transactions, and the greater the impact of the sanitization on the database.

We ran the Apriori algorithm to select such patterns. The time required to build the inverted file in main memory was 4.05 seconds. Based on this inverted file, we retrieved all the sensitive transactions in 1.02 seconds. With our ten original restrictive patterns, 22479 patterns (out of 1866693) became restricted in the database since any pattern that contains restrictive patterns should also be restricted. Thus, in our experiments R_P contained the 22479 restrictive patterns.

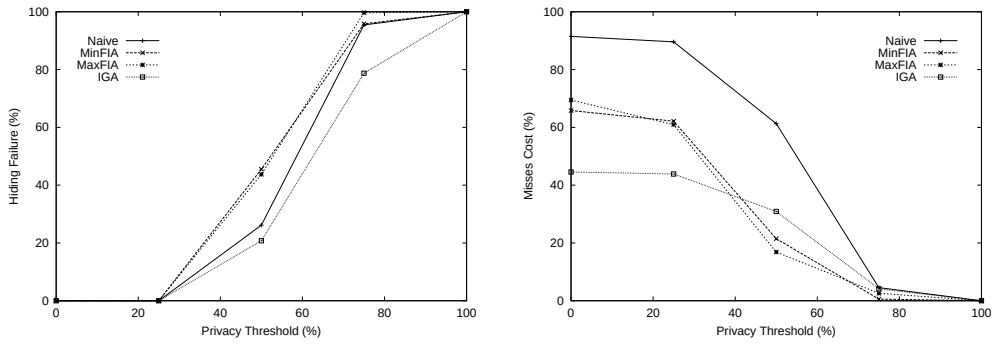

Figure 6: Effect of ψ on the hiding failure and the misses cost

Figure 6 shows the effect of the privacy threshold ψ on the hiding failure and the misses cost for all four algorithms, considering the minimum support threshold $\sigma = 10\%$. As can be observed, when ψ is 0%, no restrictive association is disclosed for all four algorithms. However, 90% of the legitimate patterns in the case of Naïve, 69% in the case of MaxFIA, 65% in the case of MinFIA, and 44% in the case of IGA are accidentally hidden.

When ψ is equal to 100%, all restrictive patterns are disclosed and no misses are recorded for legitimate patterns. What can also be observed is that the hiding failure for IGA is better than that for the other approaches. In addition, the impact of IGA on the database is smaller and the misses cost of IGA is the lowest among all approaches until $\psi = 40\%$. After this value, MinFIA and MaxFIA yield better results than IGA's.

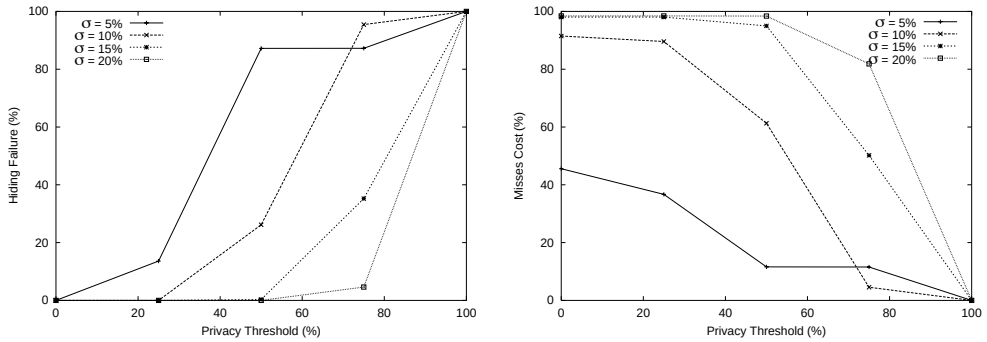

Figure 7: Effect of support threshold on privacy preservation (Naïve)

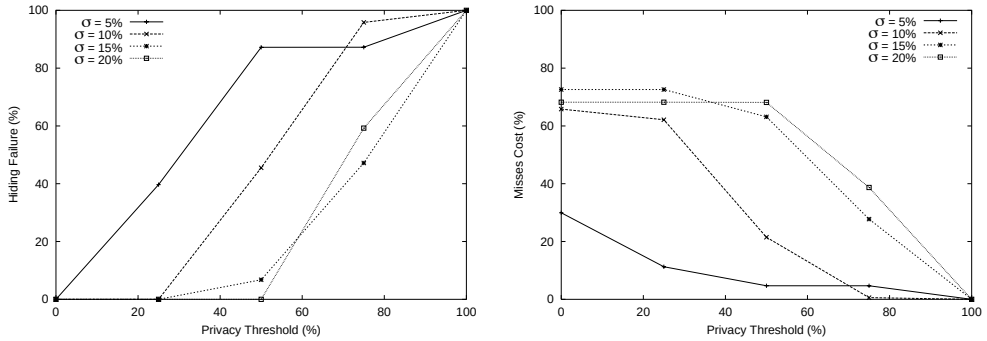

Figure 8: Effect of support threshold on privacy preservation (MinFIA)

Figures 7, 8, 9, and 10 show the effect of varying the support threshold for the patterns in the mining process. Notice that the higher the support, the more effective the hiding of patterns even with a more relaxed privacy threshold. However, the misses are more recurrent. In practice, the support threshold for mining frequent patterns should always be set to 0% since before sanitizing the database one cannot know the support threshold that the user will select. Thus, for better privacy preservation, the security administrator should assume the lowest support threshold possible.

We could measure the dissimilarity between the original and sanitized databases by computing the difference between their sizes in bytes. However, we believe that this dissimilarity should be measured

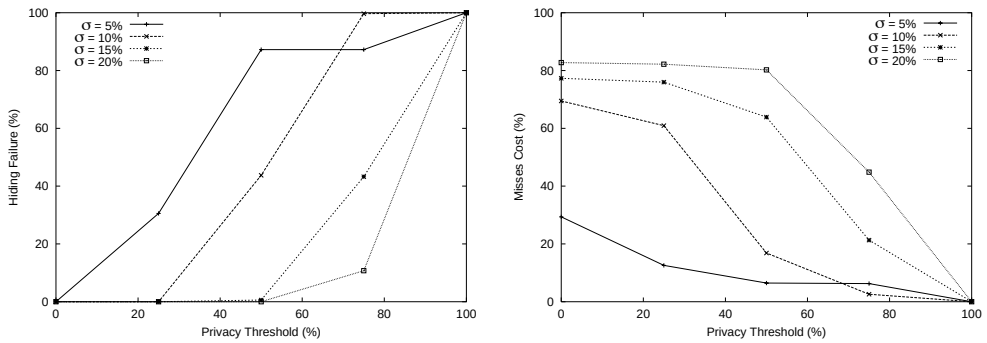

Figure 9: Effect of support threshold on privacy preservation (MaxFIA)

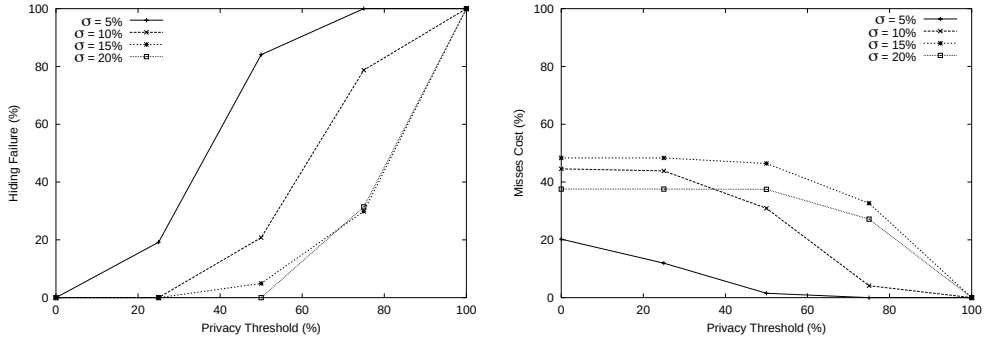

Figure 10: Effect of support threshold on privacy preservation (IGA)

comparing their contents, instead of their sizes in bytes. Comparing their contents is more intuitive and more reasonable as well.

To measure the dissimilarity between the original and the sanitized datasets we could simple compare the difference of their histograms. In this case, the horizontal axis of a histogram contains all items in the dataset, while the vertical axis corresponds to their frequencies. The sum of the frequencies of all items gives the total of the histogram. So the dissimilarity between D and D' is given by:

$$Dissimilarity(D, D') = \frac{1}{\sum_{i=1}^{n} f_D(i)} \times \sum_{i=1}^{n} [f_D(i) - f_{D'}(i)]$$

where $f_X(i)$ represents the frequency of the ith item in the dataset X.

Figure 11 shows the differential between the initial size of the database and the size of the sanitized database with respect to the privacy threshold ψ. To have the smallest impact possible on the database, the sanitization algorithm should not reduce the size of the database significantly. As can be seen in the first graph, IGA is the one that impacts the least on the database for all values of the privacy threshold ψ. In the worst case, when $\psi = 0\%$, 3.55% of the database is lost. MinFIA and MaxFIA lose 6.35% and 6.78% respectively, and Naïve reduces 16.41% of the database, with the same threshold. This is due to the fact that Naïve removes all items of a restrictive pattern in its corresponding sensitive transactions, while the other algorithms only remove one item for each restrictive pattern. Thus, as can be seen, the

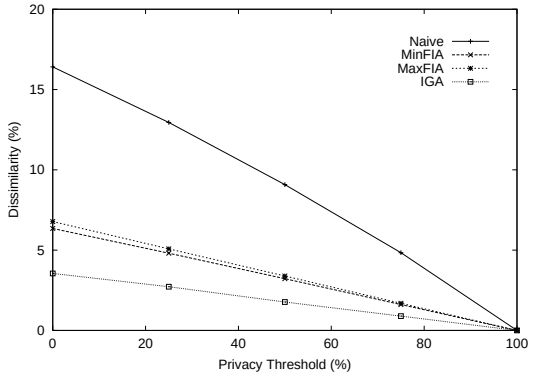

Figure 11: The difference in size between D and D'

four algorithms slightly alter the data in the original database, while enabling flexibility for someone to tune them.

5.2 CPU Time for the Sanitization Process

We tested the scalability of our sanitization algorithms vis-à-vis the size of the database as well as the number of patterns to hide. We varied the size of the original database D from 20K transactions to 100K transactions, while fixing the privacy threshold ψ and the support threshold to 0%, and keeping the set of restrictive patterns constant (10 original patterns). Figure 12 shows that the four algorithms increase CPU time almost linearly with the size of the database. Note that Naïve, MinFIA, and MaxFIA yield almost the same CPU time since they are very similar. These slight differences can be seen in Table 1. The I/O time (2 scans of the database) is also considered in these figures. This demonstrates good scalability with the cardinality of the transactional database.

Table 1: Results of CPU Time for the Sanitization Process (Figure 12)

Algorithm	Database Size				
	20K	40K	60K	80K	100K
Naïve	15.56	61.02	131.17	229.84	341.44
MinFIA	15.26	60.48	130.74	231.00	332.19
MaxFIA	15.23	60.07	129.41	227.63	331.32
IGA	13.49	53.66	119.67	208.45	312.79

The slight inflection in the curve is due to the fact that the larger the database, the more restrictive patterns can be derived from the 10 original restrictive patterns. This significantly increases the number of sensitive transactions to be touched. The number of restrictive patterns actually increases more than the number of transactions, making most sensitive transactions conflicting ones (i.e. with many restrictive patterns).

We also varied the number of restrictive patterns to hide from approximately 1290 to 6600, while fixing the size of the database to 100K transactions and fixing the support and privacy thresholds as before. Figure 13 shows that our algorithms scale well with the number of patterns to hide. The figure

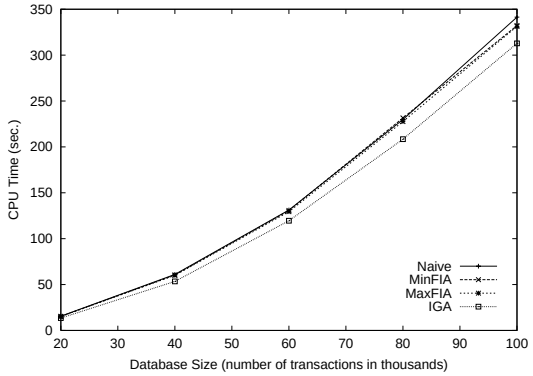

Figure 12: Sanitization process scalability with respect to database size

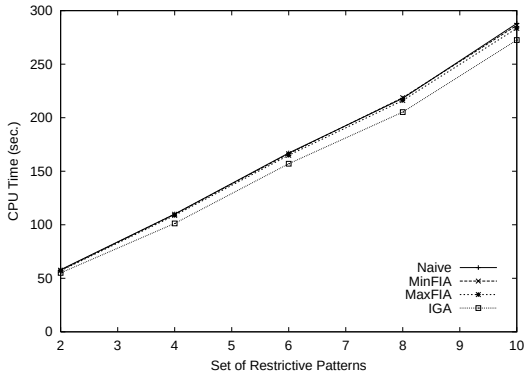

Figure 13: Sanitization process scalability with respect to number of restrictive patterns

reports the size of the original set of restricted patterns, which varied from 2 to 10. This makes the set of all restricted patterns range from approximately 1298 to 6608. Again, the algorithms Naïve, MinFia, and MaxFIA yielded results very similar as depicted in Table 2.

Table 2: Results of CPU Time for the Sanitization Process (Figure 13)

Algorithm	Number of Original Restrictive Patterns				
	2	4	6	8	10
Naïve	58.00	110.04	167.00	218.15	288.01
MinFIA	57.55	109.79	166.38	218.67	286.26
MaxFIA	57.09	108.72	164.89	216.08	283.50
IGA	55.01	101.25	156.98	205.22	272.54

This scalability is mainly due to the inverted files we use in our approaches for indexing the transactions per item and indexing the sensitive transactions per restrictive pattern. There is no need to scan the database again whenever we want to access a transaction for sanitization purposes. The inverted file gives direct access with pointers to the relevant transactions.

6 Related Work

Some effort has been made to address the problem of protecting sensitive information from discovery of highly sensitive knowledge in data mining. Early on, O'Leary [19] studied the threat imposed by data mining techniques regarding the process of uncovering hidden patterns from large databases. He emphasized the need for limiting the disclosure of personal information, a topic which has also been investigated by researchers in the statistical databases area [8].

Recently, researchers within the information security community have investigated the impact of data mining technology on database security [13, 7, 11, 9]. Such investigation considers how much information can be inferred or calculated from large data repositories made available through data mining algorithms and looks for ways to minimize the leakage of information. This effort has been restricted basically to classification [18, 1, 4] and association rules [5, 14, 21]. We focus on the latter category.

Atallah et al. [5] considered the problem of limiting disclosure of sensitive rules, aiming at selectively hiding some frequent itemsets from large databases with as little impact on other, non-sensitive frequent itemsets as possible. Specifically, the authors dealt with the problem of modifying a given database so that the support of a given set of sensitive rules, mined from the database, decreases below the minimum support value.

Dasseni et al. [14] extended the work presented in [5]. They investigated confidentiality issues of a broad category of association rules. This solution requires CPU-intensive algorithms and, in some way, modifies true data values and relationships. In the same direction, Saygin et al. [21] introduced a method for selectively removing individual values from a database to prevent the discovery of a set of rules, while preserving the data for other applications. They proposed some algorithms to obscure a given set of sensitive rules by replacing known values with unknowns, while minimizing the side effects on non-sensitive rules.

How efficient are the previous solutions considering the three potential problems presented in Figure 1B? In [5], the authors attempt to the theoretical approach, whereas in [14, 21] the authors focus more on the practical value. Such solutions deal with the problem 1 since they hide sensitive association rules. In addition, they aim at minimizing the problem 2. On the other hand, these solutions introduce the problem 3. The reason for this is simple - they add noise to the data by turning some items from 0 to 1 in some transactions. In doing so, they increase the frequencies of some items in the original data, so that a miner is allowed to discovery new patterns or even association rules that do not exist in the non cleansed database.

Our work differs from the related work in some aspects, as follows: First, the hiding strategies behind our algorithms deal with the problem 1 and 2 in Figure 1, and most importantly, they do not introduce the problem 3 since we do not add noise to the original data. Second, we study the impact of our hiding strategies in the original database by quantifying how much information is preserved after sanitizing a database. So, our focus is not only on hiding restrictive patterns but also on maximizing the discovery of patterns after sanitizing a database. More importantly, our sanitizing algorithms select

sensitive transactions with the lowest degree of conflict and remove from them the victim item with specific criteria, while the algorithms in related work remove and/or add items from/to transactions without taking into account the impact on the sanitized database. Third, our framework can achieve a reasonable performance since it is built on indexes. Another difference of our framework from the related work is that we "plug" a transaction retrieval search engine for searching transaction IDs through the transactional database efficiently. In addition, we present a taxonomy for sanitizing algorithms which has not been considered in the literature so far.

7 Conclusions

In this paper, we have introduced a new framework for enforcing privacy in mining frequent patterns, which combines three advances for efficiently hiding restrictive rules: inverted files, one for indexing the transactions per item and a second for indexing the sensitive transactions per restrictive pattern; a transaction retrieval engine relying on boolean queries for retrieving transaction IDs from the inverted file and combining the resulted lists; and a set of sanitizing algorithms. This framework aims at meeting a balance between privacy and disclosure of information.

In the context of our framework, the integration of the inverted file and the transaction retrieval engine are essential to speed up the sanitization process. This is due to the fact that these two modules feed the sanitizing algorithms with a set of sensitive transactions to be sanitized. It should be noticed that this index schema and the transaction retrieval engine are simple to be implemented and can deal with large databases without penalizing the performance since these two techniques are scalable.

The experimental results revealed that our algorithms for sanitizing a transactional database can achieve reasonable results. Such algorithms slightly alter the data while enabling flexibility for someone to tune them. In particular, the IGA algorithm reached the best performance, in terms of dissimilarity and preservation of legitimate frequent patterns. In addition, the IGA algorithm also yielded the best response time to sanitize the experimental dataset. This is because when the algorithm removes one item of a sensitive transaction, it hides some restrictive patterns that are clustered by such an item. Apart from the contributions of the sanitizing algorithms, in this paper we also introduced a taxonomy for our algorithms based on transaction sanitization.

Another contribution of this work includes three performance measures that quantify the fraction of mining patterns which are preserved in the sanitized database. The Hiding Failure measures the amount of restrictive patterns that are disclosed after sanitization. Misses Cost measures the amount of legitimite patterns that are hidden by accident after sanitization, and Artifactual Patterns measure the artificial patterns created by the addition of noise in the data. We evaluated such metrics by testing different values of the privacy threshold ψ for our algorithms.

The work presented herein adresses the issue of hiding some frequent patterns from transactional databases. All association rules deriavable from these frequent patterns are thus also hidden. This could make the approach sometimes restrictive. For instance, if the pattern ABC is restricted, the pattern $ABCD$ would also be restricted since it includes the previous one, and the association rule $ABC \rightarrow D$

would be hidden even though initially there was no restrictions on D. There is no means to specify the constraints on the association rules rather than the frequent patterns. One may want to express that $AB \rightarrow C$ is restricted but not $C \rightarrow AB$. However, this is not feasible at the frequent patterns level since both rules are derived from the same frequent pattern ABC. We are investigating new optimal sanitization algorithms that not only impact the support of a frequent pattern to be hidden but also impact on the confidence of some combinations of these itemsets (i.e. association rules) to allow for instance $A \rightarrow B$ but not $B \rightarrow A$. We are also investigating, in the context of privacy in data mining, association rules or other patterns, the integration of role-based access control in relational databases with rule-based constraints specifying privacy policies.

8 Acknowledgments

Stanley Oliveira was partially supported by CNPq (Conselho Nacional de Desenvolvimento Científico e Tecnológico) of Ministry for Science and Technology of Brazil, under Grant No. 200077/00-7. Osmar Zaïane was partially supported by a Research Grant from NSERC, Canada.

References

[1] D. Agrawal and C. C. Aggarwal. On the Design and Quantification of Privacy Preserving Data Mining Algorithms. In *Proceedings of ACM SIGMOD/PODS*, Santa Barbara, CA, May 2001.

[2] R. Agrawal, T. Imielinski, and A. N. Swami. Mining association rules between sets of items in large databases. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, pages 207–216, Washington, D.C., May 1993.

[3] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proceedings of the 20th International Conference Very Large Data Bases, VLDB*, pages 487–499, Santiago, Chile, September 1994.

[4] R. Agrawal and R. Srikant. Privacy-Preserving Data Mining. In *Proceedings of of the 2000 ACM SIGMOD International Conference on Management of Data*, pages 439–450, Dallas, Texas, May 2000.

[5] M. Atallah, E. Bertino, A. Elmagarmid, M. Ibrahim, and V. Verykios. Disclosure limitation of sensitive rules. In *Proceedings of IEEE Knowledge and Data Engineering Workshop*, Chicago, Illinois, November 1999.

[6] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison Wesley Longman, 1999.

[7] L. Brankovic and V. Estivill-Castro. Privacy Issues in Knowledge Discovery and Data Mining. In *Proceedings of Australian Institute of Computer Ethics Conference (AICEC99)*, Melbourne, Victoria, Australia, July 1999.

[8] S. Castano, M. Fugini, G. Martella, and P. Samarati. *Database Security*. Addison-Wesley Longman Limited, England, 1995.

[9] L. Chang and I. Moskowitz. An Integrated Framework for Database Privacy Protection. In *Proceedings of the 14h Annual IFIP WG 11.3 Working Conference on Database Security*, Schoorl, The Netherlands, August 2000.

[10] Z. Chen. *Data Mining and Uncertain Reasoning.* John Wiley and Sons, Inc., New York, NY, 2001.

[11] C. Clifton. Using Sample Size to Limit Exposure to Data Mining. *Journal of Computer Security*, 8(4):281–307, November 2000.

[12] C. Clifton, W. Du, M. Atallah, M. Kantarcioglu, X. Lin, and J. Vaidya. Distributed Data Mining to Protect Information Privacy. Proposal to the National Science Foundation, December 2001.

[13] C. Clifton and D. Marks. Security and Privacy Implications of Data Mining. In *Workshop on Data Mining and Knowledge Discovery*, pages 15–19, Montreal, Canada, February 1996.

[14] E. Dasseni, V. S. Verykios, A. K. Elmagarmid, and E. Bertino. Hiding Association Rules by Using Confidence and Support. In *Proceedings of the 4th Information Hiding Workshop (IHW2001)*, Pittsburg, PA, April 2001.

[15] M. Dietzfelbinger, A. R. Karlin, K. Mehlhorn, F. M. auf der Heide, H. Rohnert, and R. E. Tarjan. Dynamic Perfect Hashing: Upper and Lower Bounds. *SIAM Journal on Computing*, 23(4):738–761, 1994.

[16] A. Gyenesei. Data Mining Approach for Solving Decision Support Problems of Warehouse Networks, POLVAX (in Hungarian), pages 69-93.

[17] J. Han and M. Kamber. *Data Mining: Concepts and Techniques.* Morgan Kaufmann Publishers, San Francisco, CA, 2001.

[18] T. Johnsten and V. Security Procedures for Classification Mining Algorithms. In *Proceedings of 15th Annual IFIP WG 11.3 Working Conference on Database and Applications Security*, pages 293–309, Niagara on the Lake, Ontario, Canada, July 2001.

[19] D. E. O'Leary. Knowledge Discovery as a Threat to Database Security. In G. Piatetsky-Shapiro and W. J. Frawley (editors): Knowledge Discovery in Databases. AAAI/MIT Press, pages 507-516, Menlo Park, CA, 1991.

[20] J. S. Park, M-S. Chen, and P. S. Yu. An Effective Hash-Based Algorithm for Mining Association Rules. In Michael J. Carey and Donovan A. Schneider, editors, *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data*, pages 175–186, San Jose, California, November 1995.

[21] Y. Saygin, V. S. Verykios, and C. Clifton. Using Unknowns to Prevent Discovery of Association Rules. *SIGMOD Record*, 30(4), December 2001.

[22] C. Silverstein, S. Brin, and R. Motwani. Beyond Market Baskets: Generalizing Association Rules to Dependence Rules. *Data Mining and Knowledge Discovery*, 2(1):39–68, 1998.

[23] H. Toivonen. Sampling large databases for association rules. In T. M. Vijayaraman, Alejandro P. Buchmann, C. Mohan, and Nandlal L. Sarda, editors, *Proceedings of the 22nd International Conference on Very Large Data Bases*, pages 134–145, Mumbai, Bombay, India, September 1996. Morgan Kaufman.