Disclosure Limitation of Sensitive Rules *

M. Atallah[1] E. Bertino[2] A. Elmagarmid[3] M. Ibrahim[3] V. Verykios[4]

[1]CERIAS and Department of Computer Sciences, Purdue University
[2]Dipartimento di Scienze dell' Informazione, Universita' di Milano, Italy
[3]Department of Computer Sciences, Purdue University
[4]College of Information Science and Technology, Drexel University

Abstract

Data products (macrodata or tabular data and microdata or raw data records), are designed to inform public or business policy, and research or public information. Securing these products against unauthorized accesses has been a long-term goal of the database security research community and the government statistical agencies. Solutions to this problem require combining several techniques and mechanisms. Recent advances in data mining and machine learning algorithms have, however, increased the security risks one may incur when releasing data for mining from outside parties. Issues related to data mining and security have been recognized and investigated only recently.

This paper, deals with the problem of limiting disclosure of sensitive rules. In particular, it is attempted to selectively hide some frequent itemsets from large databases with as little as possible impact on other, non-sensitive frequent itemsets. Frequent itemsets are sets of items that appear in the database "frequently enough" and identifying them is usually the first step toward association/correlation rule or sequential pattern mining. Experimental results are presented along with some theoretical issues related to this problem.

1 Introduction

Securing data against unauthorized accesses has been a long-term goal of the database security research community and the government statistical agencies. Solutions to such a problem require combining several techniques and mechanisms. In particular, it is well known that simply restricting access to sensitive data does not ensure full data protection. It may, for example, be the case that sensitive, or "high" data items can be inferred from non-sensitive, or "low" data, through some inference process based on some knowledge the user has. Such a problem, known as the "inference problem", has been widely investigated [9, 5], and possible solutions have been identified. In general, all those approaches address the problem of how to prevent disclosure of sensitive data through the combination of known inference rules with non-sensitive data [3]. Examples of inference rules are deductive rules and functional dependencies. Those approaches, however, do not deal with the problem of how to prevent the discovery of the inference rules themselves. In other words, rules are not considered as sensitive "knowledge".

Recent advances in data mining techniques and related applications [6] have, however, increased the security risks one may incur when releasing data. The main goal of such techniques is to enable the rapid and efficient discovery of hidden intensional knowledge from a, possibly very large, set of data items. The use of such techniques would therefore enable users to easily acquire, not only knowledge that could be used to infer sensitive data, but also sensitive knowledge. Note that knowledge usually acquired through data mining techniques cannot be considered as absolute. It can be rather characterized as probabilistic knowledge. However, even such probabilistic knowledge may provide sensitive information to users [3].

Issues related to data mining and security have been recognized and investigated only recently. So, only a few approaches have been devised till now. These approaches are discussed in Section 2. However, there is still no comprehensive view of those issues and of the possible spectrum of solutions. There is, for example, the need of analyzing specific data mining techniques in the light of the security problem which was mentioned previously.

In this paper, a contribution is made towards addressing such a need in the context of a specific type of knowledge. Such type of knowledge, known as association rules, consists of a set of statements of the form "90% of air-force

*Portions of this work were supported by sponsors of the Center for Education and Research in Information Assurance and Security.

basis having super-secret plane A, also have helicopters of type B". An association rule is usually characterized by two measures, the support and the confidence. In general, algorithms for the discovery of association rules detect only rules whose support is higher than a minimum threshold value. We refer to such rules as "significant rules". The problem that we deal with in this paper is how to modify a given database so that the support of a given set of sensitive rules, mined from the database, decreases below the minimum support value. We would like, however, to remark that our approach is a simple building block that by itself does not provide a comprehensive solution to the problem of data mining and security. However, it can be considered as a basic ingredient of such a comprehensive solution.

The remainder of this paper is organized as follows. First we review current approaches addressing data mining and security. We then present a formulation of our problem and show that the optimal solution to it is NP-hard. We then present some heuristics. Finally, we outline further work.

2 Related Work

Security and privacy threats arising from the use of data mining techniques have been first pointed out in an early paper by O' Leary [8] and recently in the seminal paper by Clifton and Marks [4]. The authors in [4] outline possible solutions to prevent data mining of significant knowledge, that include releasing only subsets of the source database, fuzzyfying the source database, and augmenting the source database. They also point out a "research agenda" that includes several issues to be investigated. Among those issues, a relevant one to our approach, is the analysis of mining algorithms, which gives the criteria that must be used by the algorithm in order to decide whether or not rules are relevant, so that one can prevent the mining of sensitive rules. The paper of Clifton and Marks, however, does not analyze any specific data mining technique or algorithm, whereas this paper deals with a specific technique.

A recent paper by Clifton [3] presents an interesting approach to the problem of data mining and security. The approach is based on releasing a sample of the source database so that the rules mined from the released sample are not significant. A main result of the paper is to show how to determine the right sample size by using lower bounds from pattern recognition. The proposed approach is independent from any specific data mining technique. The main difference between such approach and ours is that we aim at a finer tuning of the intensional knowledge to be released. In other words, our aim is on how to reduce the significance of a given rule, or sets of rules, by possibly leaving unaltered the significance of the other rules, or by minimally changing it. By contrast, the approach by Clifton aims at estimating the error which is introduced on the significance of the rules by reducing the sample sizes. It is worth noting, however, that the two approaches can be used together as part of a comprehensive environment supporting the security administration.

In addition to work dealing specifically with the issue of data mining and security, it is also important to mention work in the area of security for statistical databases [10]. Such work deals with the problem of limiting disclosure of individual data items and at the same time ensures that correct statistics can be derived from the released database. The main difference between the work on security for statistical databases and the work presented in this paper, or more generally in the area of data mining, is that in the latter two cases, even if individual data items are allowed to be directly accessed, the intensional knowledge which is derived can be controlled. However, techniques used in statistical databases, such as data fuzzyfication or data swapping, could also be used in the other context.

3 Association Rules and Sanitization

In this section, the notion of association rules is precisely defined and a formulation of the problem is given. It is then proven that the problem of finding an optimal sanitization of the source database is NP-Hard. This is done for a number of (progressively more realistic) notions of what it means to "sanitize". The proofs are based on reductions of the problem addressed in this paper to the Hitting-Set problem.

3.1 The Problem

The problem of association rule mining was initially presented in [1]. The authors in [2] extended and formalized the problem as follows: Let $\mathcal{I} = \{i_1, i_2, \ldots, i_m\}$ be a set of literals, called items. Let D be a database of transactions, where each transaction T is an itemset such that $T \subseteq \mathcal{I}$. Associated with each transaction is a unique identifier, called its TID. A set of items $X \subset \mathcal{I}$ is called an *itemset*. A transaction T *contains* an itemset X, if $X \subseteq T$. An *association rule* is an implication of the form $X \Rightarrow Y$ where $X \subset \mathcal{I}$, $Y \subset \mathcal{I}$, and $X \cap Y = \emptyset$. The rule $X \Rightarrow Y$ holds in the set of transactions D with *confidence* c if $\frac{|X \cup Y|}{|X|} \geq c$ where $|A|$ is the number of occurrences of the set of items A in the set of transactions D. The rule $X \Rightarrow Y$ has *support* s if $\frac{|X \cup Y|}{N} \geq s$ where N is the plurality of the transaction set D.

In this paper we focus on the discovery of large/frequent itemsets, in order to keep the discussion simple. These itemsets have greater support than the minimum support specified. This restriction implies that the only way to hide a rule is by decreasing the support of its corresponding large itemset. Exhaustive search of frequent sets is obviously infeasible for all but small itemsets: the search space of

potential frequent itemsets increases exponentially with the number of items in the transactions. A more efficient method for the discovery of frequent itemsets can be based on the following iterative procedure: in each pass the algorithm starts with a *seed* set of large itemsets, called *candidate* itemsets. The support of these itemsets is computed during the pass over the data. At the end of the pass, it is determined which of the candidate itemsets are actually large, and they become the seed for the next phase. This process continues until no new large itemsets are found. In the first pass, the support of individual itemsets is counted in order to be determined which of them are large.

The specific problem we address can be stated as follows. Let D the source database, let $\mathcal{R}$ be a set of significant association rules that are mined from D, and let R_h beas et of rules in $\mathcal{R}$. How can we transform D into a database D' so that all (or the maximum number of) rules in $\mathcal{R}$ can still be mined from D' but for the rules in R_h. D' then becomes the *released database*. Therefore, our problem is to reduce the support of the rules in R_h below the given threshold. We refer to such a transformation as *sanitization* of D. In addition to preserve as much knowledge as possible, the transformation we seek, should also be achieved at as much low cost as possible.

3.2 Optimal Sanitization Is NP-Hard

Let A be the set of large itemsets that are "good" in the sense that we do not wish to make them small. Let B be the set of large itemsets that are "bad", i.e., we want to make them small. These two goals can be incompatible, so the problems we formulate below are based on the notion that we want to make all of B's itemsets small, while making as few as possible of A's itemsets small. We prove the NP-hardness of three optimization problems based on this notion. The first (called PROBLEM 1) is really just a "warmup" for PROBLEM 2 because its framework is somewhat unrealistic: It assumes that we can remove support for the various individual items independently of one another. Its proof is simple and serves as an easy introduction to the (more elaborate) reduction needed for proving PROBLEM 2 (whose framework is realistic – more on this below). Finally, we prove the NP-hardness of PROBLEM 3 which is the problem that we focus on this paper. More specifically, in PROBLEM 3 we can modify a transaction by deleting some items from it.

PROBLEM 1: Given two sets A and B of subsets of a finite set J, such that no element of B is a subset of any element of A and no element of A is a subset of any element of B, find a set of elements R in J such that every subset in B contains at least one of those elements while minimizing the number of subsets of A that contain elements from R.

Note: The idea is that by removing support from the items in R we make all of B's itemsets small, while affecting as few of A's itemsets as possible.

The framework considered next (for PROBLEM 2 below) is more realistic, in that we now weaken the support for itemsets in B by deleting some transactions from the database. Of course there are many ways of doing this, some of which have more impact on A than others. The goal is to do it in a way that minimizes this impact on A.

Note: In our formulation of PROBLEM 2 the notion of "large" is stated in terms of the actual number of transactions containing the itemset (rather than as a percentage). There is no loss of generality in doing so, because the number of transactions in the database can easily be kept constant; whenever we delete a transaction we can keep the total size of the database constant by replacing the deleted transaction with another transaction that has no effect on either A or B (for example the new transaction could contain only items that have very small support in the database, possibly zero support, i.e., "new" items). We henceforth call t the threshold for "largeness", i.e., an itemset is considered large iff there are at least t transactions containing it.

PROBLEM 2: We are given a set J of items, an integer t, a database D of transactions (each of which is a subset of J), sets A and B each of which contains a collection of subsets of J (the "itemsets"). Each itemset in A or in B has support $\geq t$ in D. The problem is to compute a subset D' of D such that deleting D' from D results in a database where every itemset in B has support $< t$, and the number of itemsets in A that have support $< t$ is minimized.

PROBLEM 3: We are given a set J of items, an integer t, a database D of transactions (each of which is a subset of J), sets A and B each of which contains a collection of subsets of J (the "itemsets"). Each itemset in A or in B has support $\geq t$ in D. What we are allowed to do is at a finer granularity than in PROBLEM 2: We can now individually modify a transaction instead of deleting it, by deleting some items from it. The problem is then to modify some transactions in D such that in the resulting database every itemset in B has support $< t$, and the number of itemsets in A that have support $< t$ is minimized.

The proofs of the NP-hardness of PROBLEM 1, 2 and 3 are based on reductions from the NP-hard problem of HITTING SET, which we review next.

HITTING SET (page 222 of the book by Garey and Johnson on NP-completeness [7]): Given a set C of subsets of a finite set S, find a smallest subset S' of S such that every subset in C contains at least one element in S'. The problem remains NP-hard even if every subset in C consists of no more than 2 elements of S.

Proposition 1. *PROBLEM 1 is NP-hard.*

Proof. Given an instance of HITTING SET, here is how to create an instance of PROBLEM 1 such that a polynomial time solution to the latter implies a polynomial time solution

to the former. Let $S = \{1, \ldots, n\}$ for HITTING SET. Then for PROBLEM 1 here is what A, B, J look like (in terms of the C and S of the HITTING SET problem instance):

$J = S \cup \{n+1\}$, (i.e., $J = \{1, \ldots, n, n+1\}$)

$$A = \{\{1, n+1\}, \{2, n+1\}, ..., \{n, n+1\}\}$$

$B = C$, (hence $n+1$ does not appear anywhere in B)

The R that solves the instance of PROBLEM 1 is equal to the S' that solves the instance of HITTING SET. □

Theorem 1. *PROBLEM 2 is NP-hard.*

Proof. A reduction from HITTING SET. Let S be $\{1, \ldots, n\}$, and C be the collection of subsets of S, for HITTING SET. To solve HITTING SET one must find a minimum subset S' of S such that every x in C contains at least one element of S'. Without loss of generality, we can assume that S does not contain a redundant element, i.e., no pair i,j of elements from S is such that "$x \in C$ contains i" implies "x contains j" (otherwise i is redundant and we can completely ignore it when solving the problem). Also, because HITTING SET remains NP-hard if every x in C consists of no more than two elements, we assume that this is the case. We can also assume that no x in C consists of a single element, because such an element surely has to be part of S' (i.e., the problem remains NP-hard if we assume that C is a set of unordered pairs of elements of S). We next explain how the instance of PROBLEM 2 is created. First we introduce some notation.

We use $f(i)$ to denote the set of items that appear with i in one or more x in C (including i itself), that is, $f(i) = \{j : j = i$ or $\{i, j\} \in C\}$. (Note that no $f(j)$ properly contains another $f(i)$ because otherwise i or j is redundant.)

We use $g(i)$ to denote $f(i) \cup \{n+1\}$.

The symbols J,t,A,B,D in what follows are as in the definition of PROBLEM 2 that was given earlier in this section. The instance of PROBLEM 2, which can easily be created in polynomial time, has the following parameters:

$J = S \cup \{n+1\}$, (i.e., the items are $1, 2, \ldots, n, n+1$)

$t = 2$, i.e., two occurrences are needed to be considered "large".

$A = \{f(1), f(2), ..., f(n)\}$, (note that item n+1 appears nowhere in A)

$$B = \{x \cup \{n+1\} : x \in C\}$$

The database D consists of $2n$ transactions:

$$D = \{f(1), g(1), f(2), g(2), \ldots, f(n), g(n)\}.$$

Observe that each itemset $\{i, j, n+1\}$ in B has support of precisely 2 (from transactions $g(i)$ and $g(j)$. Each itemset $f(i)$ in A also has support of precisely 2 (from transactions $f(i)$ and $g(i)$).

A solution to PROBLEM 2 does not delete an $f(i)$ transaction because that would decrease support for itemset $f(i)$ in A without decreasing support for any itemset in B (because the latter contain item $n+1$ whereas $f(i)$ does not). If the solution deletes transaction $g(i)$ then:

1. it decreases the support of every itemset of B that contains i from 2 to 1, i.e., it makes that itemset small; this corresponds to a solution in HITTING SET that selects i to be in S'.

2. it decreases the support of itemset $f(i)$ in A from 2 to 1, i.e., it makes the itemset small; minimizing the number of times this happens in the solution to PROBLEM 2 corresponds to minimizing the size of S' in HITTING SET.

The above observations imply that, from the set of transactions deleted by the solution to PROBLEM 2, we can obtain the S' that solves HITTING SET. □

Theorem 2. *PROBLEM 3 is NP-hard.*

Proof. A reduction from HITTING SET. Let S be $\{1, \ldots, n\}$, and C be the collection of subsets of S, for HITTING SET. To solve HITTING SET one must find a minimum subset S' of S such that every x in C contains at least one element of S'. As in the proof of Theorem 1, S does not contain a redundant element, and every x in C consists of two elements. We next explain how the instance of PROBLEM 3 is created. The symbols J, t, A, B, D in what follows are as in the definition of PROBLEM 3 that we gave earlier.

$$J = S \cup \{n+1, \ldots, 3n\}.$$

That is, the items are $\{1, 2, \ldots, 4n\}$. To improve the readability of our proof, we use the notation $\#_i$ for item $n+i$, $\&_i$ for item $2n+i$, and $\$_i$ for item $3n+i$, $1 \leq i \leq n$.

As in the proof of PROBLEM 2, we use $f(i)$ to denote the set of items that appear with i in one or more x in C (including i itself), that is,

$$f(i) = \{j : j = i \text{ or } \{i, j\} \in C\}.$$

(Note that no $f(j)$ properly contains another $f(i)$ because otherwise i or j is redundant.)

We use $g(i)$ to denote $f(i)$ as well as all $\#_j$, $\&_j$,a nd $\$_j$ for which $j \in f(i)$, i.e.,

$$g(i) = f(i) \cup \{\#_j : j \in f(i)\} \cup \{\&_j : j \in f(i)\} \cup \{\$_j : j \in f(i)\}.$$

For every $x = \{i, j\} \in C$ we use $p(x)$ to denote the pair $\{\#_i, \#_j\}$ and we use P to denote the set of all such pairs, i.e., $P = \{p(x) : x \in C\}$.

For every $x = \{i, j\} \in C$ we use $q(x)$ to denote $\{\#_i, \#_j, \&_i, \&_j\}$ and we use Q to denote the set of all such $q(x)$'s, i.e., $Q = \{q(x) : x \in C\}$.

For every $x = \{i, j\} \in C$ we use $t(x)$ to denote $\{\#_i, \#_j, \&_i, \&_j, \$_i, \$_j\}$ and we use T to denote the set of all such $t(x)$'s, i.e., $T = \{t(x) : x \in C\}$.

The other parameters of the instance of PROBLEM 3, which can easily be created in polynomial time, are described next.

$t = 2$, i.e., two occurrences are needed to be considered "large".

The set A now consists of the $f(i)$'s as well as the $p(x)$'s, $q(x)$'s, and $t(x)$'s:

$$A = \{f(1), f(2), \ldots, f(n)\} \cup P \cup Q \cup T.$$

$$B = \{\{i, j, \#_i, \#_j\} : \{i, j\} \in C\}.$$

The database D consists of $2n$ transactions:

$$D = \{f(1), g(1), f(2), g(2), \ldots, f(n), g(n)\}.$$

Observe that each itemset $\{i, j, \#_i, \#_j\}$ in B has support of precisely 2 (from transactions $g(i)$ and $g(j)$). Each itemset $f(i)$ in A also has support of precisely 2 (from transactions $f(i)$ and $g(i)$). Itemsets $\{\#_i, \#_j\}$, $\{\#_i, \#_j, \&_i, \&_j\}$, and $\{\#_i, \#_j, \&_i, \&_j, \$_i, \$_j\}$ in A each have support 2 (from transactions $g(i)$ and $g(j)$).

A solution to PROBLEM 3 does not modify an $f(i)$ transaction because that would decrease support for itemset $f(i)$ in A without decreasing support for any itemset in B. Now suppose the solution modifies a transaction $g(i)$. The solution surely does not remove a $\&_j$ or $\$_j$ from transaction $g(i)$ because that would decrease support for itemsets of A without decreasing support for any itemset in B. We also claim that the solution does not decrease support for B by removing a $\#_j$ from transaction $g(i)$, i.e., that $g(i)$ is always modified by removing from it item(s) in $\{1, \ldots, n\}$. To see that this is so, assume to the contrary that $g(i)$ was modified by removing a $\#_j$ from it (we call this a "type 1" modification). We now show that this leads to a contradiction.

Comment. Observe that deleting j (where $j \neq i$) from $g(i)$ can always be replaced by deleting i from $g(i)$. This is because, although both have the same effect on A (i.e., weakening $f(i)$), deleting i weakens *every* $x \in B$ where $i \in x$ whereas deleting $j \neq i$ weakens only $\{i, j, \#_i, \#_j\} \in B$.

The deletion of $\#_j$ from transaction $g(i)$ weakens in A the (at least three) itemsets $p(x)$, $q(x)$, $t(x)$ where $j \in x$. On the other hand, modifying $g(i)$ by removing from it an item $j \in \{1, \ldots, n\}$ (a "type 2" modification) rather than removing a $\#_j$, would have the same effect on B but would weaken only $f(i)$ in A. This implies that globally replacing, in the solution, all of the type 1 modifications by the corresponding type 2 modifications would not make a difference for B but would decrease the number of itemsets of A that are affected, implying that the solution could not have been optimal.

We henceforth assume that the solution contains only type 2 modifications to transactions. A type 2 modification that an item from transaction $g(i)$ corresponds to a solution in HITTING SET that selects i to be in S'. It also decreases the support of itemset $f(i)$ in A from 2 to 1, i.e., it makes the itemset small; minimizing the number of times this happens in the solution to PROBLEM 3 corresponds to minimizing the size of S' in HITTING SET.

The above observations imply that, from the set of transactions modified by the solution to PROBLEM 3, we can obtain the S' that solves HITTING SET. □

4 Heuristic Approach

In this section, we describe the heuristic approach that we propose to solve the Optimal Sanitization problem. Before the algorithm is presented some preliminary definitions are given and the data structures needed for the algorithm are described with some examples where necessary. Some complexity analysis results follow the formal presentation of the algorithm as well as experimental results from the performance evaluation of the proposed heuristic are given.

4.1 Preliminary Definitions

In order to better illustrate how the algorithm works, we represent all large itemsets in the database in terms of a graph, referred to as the *itemset graph*. In the remainder of the discussion, we use the term "i-itemset", where i is an integer, with the meaning of an itemset whose cardinality is i.

Definition 4.1 (Itemset Graph.) Let SLI be a set of large itemsets. An Itemset Graph over SLI is defined as follows: (i) there is a node for each itemset in SLI; and (ii) there is an edge from the node representing itemset i_h to the node representing itemset i_j, if $i_h \subset i_j$.

An itemset graph is the result of a breadth-first search in the space of potentially large itemsets. Each node, except from the itemset, contains other information that we will introduce later on. Assuming the existence of a database like the one shown in Table 1a, then the Figure 1 shows the graph for the example database for the large itemsets shown in Table 1b. There is an ordering imposed by the itemset graph in each large itemset. This ordering implies the level of a large itemset in the itemset graph.

TID	Items
T1	ABCD
T2	ABC
T3	ACD

(a)

Large Itemsets	Support
AB	2
AC	3
AD	2
BC	2
CD	2
ABC	2
ACD	2

(b)

Table 1. (a) The example itemset database; (b) the large itemsets (1-itemsets are not included) from the example database along with their support.

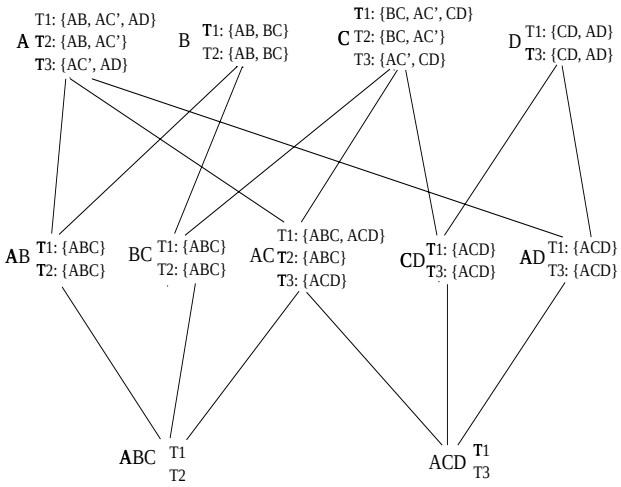

Figure 1. The graph of large itemsets.

Definition 4.2 (Level of an itemset.) The level of an itemset X is the length of the path from a large 1-itemset (reachable from X) to X.

Definition 4.3 (Level of an itemset graph.) The level of an itemset graph G is the length of the maximum path from any large 1-itemset to any other large itemset reachable from this large 1-itemset.

Of particular importance for the algorithm is the so-called "pivotal" itemsets.

Definition 4.4 (Pivotal itemset.) A pivotal itemset is a large k-itemset in the k-th level of the itemset graph.

This implies that the level of a pivotal itemset and the number of items in this itemset must be the same. The pivotal itemset is the itemset that a graph node represents. The itemsets inside the curly braces in the itemset graph are the additional information maintained in each node.

The straight lines in Figure 1 denote dependencies among the large itemsets in the sense that two connected itemsets (through one line or a path) are related either by the "provides support to" or "is supported from" binary relationships. These binary relationships are formally defined below.

Definition 4.5 ("Provides support to" relationship.) A large itemset X provides support to a large itemset Y iff $X \subset Y$.

Definition 4.6 ("Is supported by" relationship.) A large itemset Y is supported by a large itemset X iff $X \subset Y$.

The shortest path connecting the two itemset has length 1. When the relationships above apply to pivotal itemsets connected by a single line, we use the term "parent" and "child" to represent them.

The algorithm (as we will see below) performs a greedy search in the itemset graph. Of major importance in this search is the "prime" and "non-prime" itemset that we define below.

Definition 4.7 (Prime itemset.) Let X be a pivotal itemset, and Y be a child of X. Y is a prime itemset for X, if Y's support does not fall under the minimum support threshold given that X's support is decremented by one. Prime itemsets are not defined for pivotal itemsets of size equal to the level of the itemset graph.

Definition 4.8 (Non-prime itemset.) Let X be a pivotal itemset, and Y be a child of X. Y is a non-prime itemset if it is not a prime itemset. Non-prime itemsets are not defined for pivotal itemsets of size equal to the level of the itemset graph.

For each pivotal itemset, the itemset graph maintains the transactions supporting this itemset. For each pair of pivotal itemset and transaction, the itemset graph also maintains a list with the "prime" and "non-prime" itemsets.

4.2 The Sanitization Heuristic

The heuristic that we propose relies heavily upon the structure of the itemset graph. The input to the algorithm is a set of large itemsets that we need to hide. The algorithm first sorts these itemsets based on their support, and then tries to hide all of them, in an one-by-one fashion. After each pass, the algorithm keeps track of the newly hidden large itemsets, and performs a search into the list of remaining large itemsets that have not yet been hidden. If there is an itemset in the list that it becomes hidden after the current step, the algorithm removes this itemset from the list of remaining itemsets. Basically, the algorithm goes through the list of itemsets, and applies a bottom-up followed by a

```
1.  HI = itemset to be hidden
2.  k  = size of HI
3.  T  = set of transactions supporting HI
4.  while HI is not hidden
4a.   HI' = HI
4b.   for level k downto 2 do
4ba.    HI' = parent of HI' with maximum support
4bb.  end-for
4c.   Ts = transaction in T supporting HI' that
           affects the min. number of 2-itemsets
4d.   set HI' to 0 in Ts
4e.   propagate results forward
4f. end-while
```

Figure 2. Sketch of the Building Block of the Algorithm

Modifications	Number of Remaining Large Itemsets
T1-A	3
T1-B	**4**
T1-C	3
T2-A (algorithm selection)	5
T2-B	**4**
T2-C (algorithm selection)	5

Table 2. Number of remaining large itemsets for each one of the minimally modification actions.

top-down traversal of the itemset graph. In each traversal, the support of the currently scanned large itemset decreases by one.

Figure 2 illustrates the building block for the proposed heuristic approach. The algorithm takes as input the large itemset to be hidden. It then performs a greedy search through the ancestors of this itemset, selecting at each level the parent with the maximum support and setting the selected parent as the new itemset that needs to be hidden (step 4ba). At the end of this process, a large 1-itemset has been selected. The algorithm searches through the common list of transactions that support both the selected 1-itemset and the initial large itemset to be hidden in order to identify the transaction that affects the minimum number of 2-itemsets. After this transaction is identified, then it deletes the selected 1-itemset from the identified transaction (step 4d). In the sequel, it propagates the results of this modification to the graph of large itemsets.

In order to make the presentation simple, we illustrate our approach through an example. Suppose that we are given a database like the one shown in Table 1a. Suppose also that the minimum support of large itemsets (requested by the user) is 66.6% or in absolute numbers, 2 transactions per itemset. The number of large itemsets (excluding the ones with one item) are shown in Table 1b. Our goal is to hide a specific itemset, also provided as input. Hiding a large itemset means to decrease its support to a degree that cannot be considered as a favorable discovery by the competitors. Deciding upon the specific support threshold to use for rule hiding, in such a way that the outside parties will not be able to discover the sensitive rules, is an important issue that must carefully be considered, but we do not deal with it at this point. For this reason we assume that we are asked to lower the support of a given itemset by a certain amount that is also provided as an input.

Suppose that the itemset ABC has to be hidden. Suppose also that an itemset is hidden if its support is below 2. Therefore, the goal is to remove the support of this itemset from at least one of the transactions that provide support to this itemset (T1 or T2). As the Table 1b , there is a maximum of 6 itemsets in this case that may be maintained. Also, we should notice that for this algorithm, it does not matter whether a certain itemset loses support as long as it is kept above the minimum specified support.

ABC is a large 3-itemset. The algorithm first looks at all the subsets of size 2 of ABC. For each 2-itemset, it checks the support. For this case AB and BC have support 2 and AC has support 3. At this stage, it selects to concentrate on that large 2-itemset that will not be affected by modifying its support by one. For this reason, it will select itemset AC. In order to decrease the support of ABC through AC, the algorithm can only modify the support of AC in transactions T1 and T2, but not in T3. This is because, $\{T1, T2\}$ is the intersection of the transactions supporting both ABC and AC. From these two transactions, it will select the one that affects the minimum number of large-itemsets. In our case, T2 will be selected, since its list contains only one non-prime itemset, as opposed to the transaction T1, whose list contain two non-prime itemsets. In order for itemset AC to lose support from the transaction T2, either A or C must be turned into a 0 in T2. Since the lists of supported itemsets by the transaction T2 for both A and C contain the same number of prime and non-prime itemsets, the item to be modified can be selected at random.

Table 2 shows the number of remaining large itemsets after modifying a minimal number of items in the original database. The semantics of the symbols listed in the table is as follows: TI-J means turn to 0 in transaction TI the item J. The table lists all possible modifications. For each modification, the table reports the large itemsets remaining after the modification. Modifications identified by our algorithm are explicitly marked in the table. From the table, it is easy to see that our algorithmic approach has the best results for the example at hand.

LI	LI (hidden)	LI (Sanit.)	LI (Cyclic)
36	1	31	30
36	1	32	30
36	1	31	28
36	1	28	24
36	1	31	28
36	2	31	30
36	2	31	30
36	2	28	24
36	2	28	22

Table 3. Comparison between the number of large itemsets (LI) remaining after the application of the proposed Sanitization heuristic and the Cyclic algorithm.

4.3 Comparison Results

In order to evaluate the efficiency of the proposed heuristic, we have developed another algorithm, which we call it, the Cyclic algorithm. Just like the Sanitization heuristic, this algorithm starts off with a reference to a particular node in the itemset graph that needs to be hidden. To see how this approach works, let a large itemset X be made up of the items $(i_{j_1}, i_{j_2}, \ldots, i_{jk})$ and let the transactions that support it be $(t_{k_1}, t_{k_2}, \cdots, t_{k_n})$. In order to hide X, the algorithm first tries to hide the large 1-itemset i_{j_1} from t_{k_1}. Then the support of X is checked. If it is still above the threshold, then in the next iteration the large 1-itemset i_{j_2} will be hidden from t_{k_2} ands oo n. After k iterations, the algorithm goes back to trying to hide (i_{j_1}) again. After n iterations, the Cyclic algorithm selects to hide a large 1-itemset from transaction t_{k_1} once again. The iterations stop as soon as X becomes hidden.

Table 3 contains the results of comparing the difference in the number of remaining large itemsets after applying both of these algorithms to various input sets consisting of 10 transactions and 7 items per transaction, for a support level of 40%.

5 Concluding Remarks

The work reported in this paper can be extended in several directions and much work remains to be done. We recognize that there are mainly two major directions that we plan to concentrate on.

The first one is to investigate different selection criteria and evaluate their impact on the non-sensitive set of rules, as well as to apply various optimizations with respect to both the time and space complexity of the proposed heuristics.

The second direction, is to investigate the applicability of these heuristics to different rule induction schemes, such as classification rule mining, correlation rule mining, etc. Along the same lines, we are working on using not only the support but the confidence of the association rules in order to selectively hide a subset of those.

Also, it is important to devise metrics to quantify the difference between the released database and the original one. The quantification of such difference may allow the security administrator to be able to better trade between security and accuracy.

Acknowledgments

The work of Elisa Bertino was carried out when she was visiting Purdue University in summer 1999. The authors would like to acknowledge the contributions made by Yücel Saygin.

References

[1] R. Agrawal, T. Imielinski, and A. Swami. Mining Association Rules between Sets of Items in Large Databases. In *Proceedings of the ACM-SIGMOD Conference on Management of Data*, pages 207–216, Washington, DC, 1993.

[2] R. Agrawal, H. Mannila, R. Srinkant, H. Toivonen, and A. I. Verkamo. Fast Discovery of Association Rules. In U. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pages 307–328. AAAI Press/MIT Press, Menlo Park, CA, 1996.

[3] C. Clifton. Protecting Against data Mining through Samples. In *Proceedings of 13th IFIP WG11.3 Conference on Database Security*, Seattle, Washington, 1999. To appear.

[4] C. Clifton and D. Marks. Security and Privacy Implications of Data Mining. In *Proceedings of the 1996 ACM Workshop on Data Mining and Knowledge Discovery*, 1996.

[5] H. Delugach and T. H. Hinke. Wizard: A Database Inference Analysis and Detection System. *IEEE Transactions on Knowledge and Data Engineering*, 6(1):56–66, 1996.

[6] U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. From Data Mining to Knowledge Discovery: An Overview. In U. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pages 1–34. AAAI Press/MIT Press, 1996.

[7] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[8] 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*, pages 507–516. AAAI Press/MIT Press, Menlo Park, CA, 1991.

[9] D. Marks. Inference in MLS Systems. *IEEE Transactions on Knowledge and Data Engineering*, 6(1), 1996.

[10] A. Nabil and J. Wortmann. Security-Control Methods for Statistical Databases. *ACM Computing Surveys*, 21(4):515–556, 1989.