MICF: An effective sanitization algorithm for hiding sensitive patterns on data mining

Yu-Chiang Li[a], Jieh-Shan Yeh[b], Chin-Chen Chang[a,c]

[a] *Department of Computer Science and Information Engineering, National Chung Cheng University, Taiwan, R.O.C.*

{lyc, ccc}@cs.ccu.edu.tw

[b] *Department of Computer Science and Information Management, Providence University, Taiwan, R.O.C.*

jsyeh@pu.edu.tw

[c] *Department of Information Engineering and Computer Science, Feng Chia University, Taiwan, R.O.C.*

Abstract

Data mining mechanisms have widely been applied in various businesses and manufacturing companies across many industry sectors. Sharing data or sharing mined rules has become a trend among business partnerships, as it is perceived to be a mutually benefit way of increasing productivity for all parties involved. Nevertheless, this has also increased the risk of unexpected information leaks when releasing data. To conceal restrictive itemsets (patterns) contained in the source database, a sanitization process transforms the source database into a released database that the counterpart cannot extract sensitive rules from. The transformed result also conceals non-restrictive information as an unwanted event, called a side effect or the “misses cost.” The problem of finding an optimal sanitization method, which conceals all restrictive itemsets but minimizes the misses cost, is NP-hard. To address this challenging problem, this study proposes the Maximum Item Conflict First (MICF) algorithm. Experimental results demonstrate that the proposed method is effective, has a low sanitization rate, and can generally achieve a significantly lower misses cost than those achieved by the MinFIA, MaxFIA, IGA and Algo2b methods in several real and artificial datasets.

Keywords: Data mining; Association rule; Privacy-preserving; Sensitive rule hiding

1. Introduction

Data mining is an interdisciplinary field bringing together techniques to extract information from large databases [1]. Over the past few years, data mining mechanisms have widely been applied in various business organizations (for example, retail, insurance, finance, banking, and communication) and manufacturing companies such as Texas Instruments (fault diagnosis), Ford (harshness, noise, and vibration analysis), Motorola (CDMA Base Station Placement), Boeing (Post-flight Diagnostics), and Kodak (data visualization) [2].

Large companies use powerful data acquisition systems (such as minicomputers, microprocessors, and senor networks) to collect, analyze, and transfer data. The role of knowledge discovery in databases (KDD) and data mining methodologies, therefore, has become extremely important for extracting useful knowledge from huge amounts of raw data. [2, 3]

For design procedure capture, which is a specific engineering knowledge that can be captured from the design events monitored during design process [4], data mining techniques can be used in different stages of production. Manufacturing is typically a controlled process such as wafer fabrication. Using mining tools to analyze the collected data can result in efficient strategies to improve the production process, find out the unusual steps during the manufacturing process [5], and improve the reliability of system identification [6].

Data mining is an evolutionary step along the path of problem solving through data analysis [1]. Releasing collected data or mined rules for sharing has become a crucial trend among business partnerships as it results in increased productivity for all companies involved. Nevertheless, the released data have also increased the risk of incurring sensitive information leaks [7]. Organizations should evaluate and decrease the risk of disclosing information. Therefore, developing efficient approaches to maintaining an organization’s competitive edge in business by restricting information leakages has become an important issue. Consider the following two scenarios:

Scenario one: Some paper manufacturers have their own databases that record their patterns of stock and sale. For their mutual benefit, multiple companies decide to share their databases to cooperatively generate information and trends found in the shared large database. However, each company prefers to keep their own strategic patterns confidential from the others. Thus, a company can both uncover more interesting trends from the combined shared database than that available only from their own database, and can prevent sensitive information from being discovered by other companies [8].

Scenario two: The captured design procedure knowledge helps companies to understand how experienced designers carry out their designs and guide other designers to design better. Moreover, the knowledge can be used for training novice designers so that they can quickly learn how to execute prominent designs [4]. In order to preserve strategic or sensitive intelligence and still share such knowledge among allied companies, a data sanitization process or privacy-preserving techniques must be applied.

An intelligent system can be developed to achieve this. Moreover, the privacy-preserving data mining will be one of the key techniques in such a system. The mechanisms usually transform the source database into a new one from which sensitive information cannot be extracted. The procedure of transforming the source database into a new database that hides some sensitive rules is called the *sanitization process* [9].

In the association rule mining, all rules are derived from the frequent itemsets (patterns). Accordingly, one essential and efficient way to protect some restrictive patterns is to decrease their support values, which can be done by deleting or modifying items in several transactions. Such approaches usually follow two restrictions: (1)

reduce the impact on the source database; and (2) find an appropriate balance between privacy and knowledge discovery. An itemset which must be concealed is called a restrictive itemset. The impact on a source database of deleting items from transactions can be measured by the sanitization rate. The sanitization rate is defined as the ratio of deleted items to the total support values of restrictive itemsets in the source dataset. The sanitization process can also conceal some non-restrictive itemsets, which is a side effect called the "misses cost." An optimal sanitization process, which both conceals all restrictive patterns and minimize the misses cost, is an NP-hard problem [7].

The Item Grouping Algorithm (IGA) has been proposed to enforce privacy in mining frequent itemsets [10]. IGA groups restrictive itemsets and assigns a victim item to each group. This approach can reduce the impact on the database if the sanitized transaction contains more than one restrictive itemset.

IGA has a low misses cost, but can be improved further by reducing the number of deleted items. Moreover, it must deal with the overlap between groups. To deal with this problem, this study proposes a novel algorithm called Maximum Item Conflict First (MICF). The degree of conflict of an item in a sensitive transaction is defined as the number of restrictive patterns affected when an item is deleted. MICF selects an item with the maximum degree of conflict for deletion. Therefore, MICF simultaneously decreases the support values for the maximum number of restrictive patterns and reduces the number of items to be removed from the source database.

This study focuses on the task of deleting items from transactions to conceal frequent itemsets in association rule mining and on the issue of no hiding failure. All association rules derivable from these hidden frequent itemsets are thus also hidden. That is, no extra artificial itemset can be generated from the sanitized dataset.

The rest of this paper is organized as follows. Section 2 presents an overview of the current methods for solving the problem of privacy-preserving association rule mining. Section 3 explains the proposed Maximum Item Conflict First (MICF) algorithm for hiding all restrictive itemsets. The time complexity analysis is represented in Section 4. Section 5 provides experimental results and evaluates the performance of the proposed algorithm. Finally, we conclude in Section 6 with a summary of our work.

2. Background and Related Work

2.1. Mining association rules

The problem of mining association rules have been first presented in 1993 [11]. Recently, mining association rules plays one of the most important roles in data mining. Given a transaction database, mining association rules attempts to discover the significant relationship among items. The formal definition is as follows.

Let *DB* denote a transaction database, which is a set of transactions. $DB = \{T_1, T_2, ..., T_z\}$. Let $I = \{i_1, i_2, \ldots, i_n\}$ be all set of items in *DB*. Thus, $\forall\, T_q \in DB$, $T_q \subseteq I$, $1 \le q \le z$. Let *X* be a set of items, called an itemset, where $X \subseteq I$. A transaction T_q includes an itemset called *X* that T_q supports. The notation $X \Rightarrow Y$ presents an association rule, where $X \subseteq I$, $Y \subseteq I$ and $X \cap Y = \phi$. The rule $X \Rightarrow Y$ has two attributes *support* and *confidence*, respectively, denoted as $Sup(X \Rightarrow Y)$ and $Conf(X \Rightarrow Y)$. Let DB_X present the set of transactions in which each transaction includes *X*. The mathematic definition is as follows. $Sup(X \Rightarrow Y) = |DB_{X \cup Y}|/|DB|$, and $Conf(X \Rightarrow Y) = Sup(X \Rightarrow Y)/Sup(X)$. Users pre-define the minimum support (*minSup*) and the minimum confidence (*minConf*) thresholds. The two attributes of each association rule are above the two minimum thresholds, respectively. An itemset *X* is called a frequent itemset if $Sup(X) \ge minSup$. All association rules can directly be derived from the set of frequent itemsets [11, 12].

Recently, various algorithms have been proposed to efficiently discover the frequent itemsets, including level-wise algorithms [13, 14] and pattern-growth methods [15, 16]. Apriori is the most famous one, which is a level-wise algorithm containing multiple passes. In each pass, Apriori generates a candidate set of frequent *k*-itemsets (frequent itemsets with length *k*), which are combined from two arbitrary frequent (*k* – 1)-itemsets, and scan the entire transaction database to determine the frequent itemsets.

Then, Apriori generates candidate (k + 1)-itemsets for the next pass.

2.2. Privacy-Preserving in Mining Frequent Itemsets

The advances in data mining have been proven beneficial for business, but also have introduced new problems in the preservation of privacy [17]. The effort of preserving privacy can be grouped into two major classes: classification rules privacy-preserving [18-20] and association rules privacy-preserving [7, 10, 21, 22]. The former class has been widely investigated. In general, those methods attempt to prevent disclosure of sensitive data so that using non-sensitive data to infer sensitive data becomes more difficult [17]. However, they do not prevent the discovery of the inference rules themselves. Accordingly, researchers have paid attention to the later class in recent years. A sensitive association rule that should be hidden is called a restrictive rule. Restrictive rules always can be generated from frequent itemsets. Therefore, hiding a restrictive itemset implies hiding all the rules which contain the itemset. Such a frequent itemset is called the restrictive itemset.

Let *DB* represent a source database, *RI* represent the set of restrictive itemsets and ~*RI* represent the set of non-restrictive itemsets. The problem of data sanitization is to transform *DB* into a sanitized database *DB′*, called the released database, so that the most itemsets in ~*RI* can still be discovered from *DB′* while all itemsets in *RI* cannot. The process also usually unexpectedly hides non-restrictive itemsets. It is called the side effect or the misses cost of the process. The formula of the misses cost is as follows [10].

Misses cost = $\frac{|\sim RI(DB)| - |\sim RI(DB')|}{|\sim RI(DB)|}$, where $|\sim RI(DB)|$ and $|\sim RI(DB')|$ are the numbers of non-restrictive itemsets mined from *DB* and *DB′*, respectively.

Atallah *et al*. first proposed a heuristic algorithm (Sanit.) to reduce the support values of restrictive itemsets. They also have proved that the optimal sanitization process is an NP-hard problem [7]. The “Sanit.” algorithm creates a frequent itemset graph and sorts restrictive itemsets in decreasing order of their support values. Then, for a restrictive itemset, Sanit. selects an itemset with the maximum support value in each level of the itemset graph until at the top level, the victim item is selected. Sanit. also selects a sensitive transaction to delete the victim item while the sanitization affects a minimum number of the frequent 2-itemsets.

2.3. Other Sanitization Algorithms

Dasseni *et al*. proposed two schemes of sanitizing a database [23]. One uses the decreasing confidence values of the sensitive rules; the other use the decreasing support values of the rules. The authors have also implemented three algorithms Algo1a, Algo1b and Algo2a, respectively.

Saygin *et al*. described another concept to prevent the discovery of sensitive rules, which applies unknown values to replace original values [24, 25]. That method can obscure the set of sensitive rules. The authors have yet to formally prove the safety of that method.

Oliverira and Zaïane introduced the Item Grouping Algorithm (IGA) to enforce privacy in mining frequent itemsets [10]. IGA groups restrictive itemsets in identical groups with which itemsets share the same sub-itemset. Clustering the restrictive itemsets leads to the overlap between groups. Therefore, IGA assigns a victim item to each group and sorts groups according to the itemset number of groups. If the groups overlap, IGA maintains each distinct itemset only once in the group in which the itemset first appeared. If the sanitized transaction contains more than one restrictive itemset, deleting a victim item from the transaction could lead to reduction of the support values of a set of restrictive itemsets at the same time. This operation can lower the impact on the database and reduce accidental hiding of non-restrictive itemsets. IGA has low misses cost in the scenario of no restrictive itemsets can be disclosed but no efficient method to optimally cluster the restrictive itemsets based on the intersections between itemsets.

Recently, Verykios *et al*. extended the two schemes of the study [22]. The authors also introduce two further algorithms Algo2b and Algo2c. Among the support decreasing algorithms, Algo2b lost the least rules. Several researchers are also actively working in the fields of data sanitization [21, 24] and rule sanitization [26].

3. Maximum Item Conflict First (MICF) Method

Given the restrictive itemsets, frequent itemsets, and source database, the goal of the sanitization process is to protect restrictive frequent itemsets against the mining techniques used to discover them. The sanitization process, which decreases the support values of restrictive itemsets by removing items from sensitive transactions essentially, includes four sub-problems:

(1) identifying the set of sensitive transactions for each restrictive itemset;

(2) selecting the partial sensitive transactions to sanitize;

(3) for each selected transaction, identifying an item from it to be deleted (called the victim item); and

(4) rewriting the modified database to disk.

Employing distinct algorithms for the first or the fourth sub-problem does not affect the released database. Accordingly, users can apply an algorithm that deals with these two sub-problems as fast as possible. To solve the second sub-problem, most algorithms sort the sensitive transactions by transaction size or by the degree of conflict of the transactions. Selecting a victim item in each sensitive transaction significantly affects the side effect. Consequently, this study also uses the simple sorting method on the second sub-problem. For the third sub-problem, this study proposes the Maximum Item Conflict First (MICF) method to choose an appropriate victim item for each designated sensitive transaction.

Definition 3.1. If a concealed restrictive itemset cannot be extracted from the released database with an arbitrary *minSup*, the concealed itemset has no hiding failure occurrence.

Reducing the support value of an itemset implies that the itemset is hidden in a high *minSup* scenario. The database owner can set an appropriate privacy threshold (ψ) to hide the restrictive itemsets. The privacy threshold is the percentage of the support count to that of the restrictive itemset in *DB*. The hiding failure occurs when some restrictive itemsets can be extracted from a released database. A sanitization process with a low ψ cannot avoid the hiding failure except for $\psi = 0\%$. That is, to achieve a no hiding failure situation, no restrictive itemsets can appear in the released database.

3.1. Sensitive Transactions Retrieval

Definition 3.2. Let *DB*, a source transaction database, be a set of all transactions. DB_X denotes a set of transactions each transaction containing itemset *X* in *DB*. In addition, each *k*-itemset $X \subseteq I$ has an associated set of transactions $DB_X \subseteq DB$, where $X \subseteq T_q$ and $T_q \in DB_X$.

For example, in Table 1, $DB_{\{A, C\}} = \{T01, T02, T08\}$.

Theorem 3.1. A sanitization process concealing restrictive itemsets by deleting items from the source database does not generate any artificial itemset.

Proof. Let *DB* be the source database and DB' be the released database. Since DB' is generated by removing items from *DB*, clearly, there is an onto mapping *f* from *DB* to DB', so that $f(T_q) = T_q'$, where $T_q \in DB$, $T_q' \in DB'$ and $T_q' \subseteq T_q$.

Let *minSup* be the pre-defined minimum support threshold. Consider an arbitrary itemset *X*, which $X \subseteq T_q' = f(T_q) \subseteq T_q$, for all $T_q' \in DB'_X$. That is, $|DB_X| \geq |DB_X'|$. Let $d = |DB_X| - |DB_X'| \geq 0$. Assume that there is an itemset *X* which is frequent in DB_X', but infrequent in DB_X. We get

$$Sup(X) \text{ in } DB' = \frac{|DB_X'|}{|DB'|} \geq minSup > \frac{|DB_X|}{|DB|} = Sup(X) \text{ in } DB. \qquad (1)$$

Since $|DB'| \geq |DB_X'| \geq 0$,

$$\frac{|DB_X'|}{|DB'|} \leq \frac{|DB_X'| + d}{|DB'| + d} = \frac{|DB_X|}{|DB'| + d}. \qquad (2)$$

Comparing Eqs. (1) and (2), we obtain $\frac{|DB_X|}{|DB|+d} > \frac{|DB_X|}{|DB|}$. Therefore, $|DB| + d < |DB|$. It is a contraction. Therefore, a sanitization process concealing restrictive itemsets by deleting items from the source database does not generate any artificial itemset.

Q.E.D.

Definition 3.3. Let *RI* be the set of all restrictive itemsets in *DB*, and *db* be the set of all transactions containing at least one restrictive itemset. That is, $db = \{T_q \in DB \mid \exists X \in RI, X \subseteq T_q\}$. A transaction T_q in *db* is called a sensitive transaction. Moreover, db_X denotes a set of all sensitive transactions that contain restrictive itemset *X* in *db*. This is, $db_X = \{T_q \in db \mid T_q \supseteq X\}$.

For example, Table 2 lists the set of all restrictive itemsets sorted by their support count, and *db* = {*T01*, *T04*, *T05*, *T06*, *T08*} as shown in Table 3. Each transaction of *db* contains at least one restrictive itemset.

Definition 3.4. The degree of conflict of a sensitive transaction T_q is the number of restrictive itemsets which are included in T_q, denoted as $Tdegree(T_q)$. If $Tdegree(T_q) > 1$, the transaction T_q is called a conflicting transaction.

Table 3 lists the conflict degree of each sensitive transaction. For example, *Tdegree*(*T01*) = 3 because the transaction *T01* is a superset of three restrictive itemsets, {*A*, *B*}, {*C*, *D*} and {*B*, *D*}.

Table 1. Example of a transaction database

TID	Transaction
T01	{A, B, C, D}
T02	{A, C, E, F, G}
T03	{A, E, G, I, J}
T04	{B, C, D, E}
T05	{B, D, F, H}
T06	{A, B, D, F, H}
T07	{B, C, F, H, J}
T08	{A, B, C, D}

Table 2. The sorted set of all restrictive itemsets

RID(r_j)	Restrictive itemset	Support count
r_1	{D, F}	2
r_2	{A, B}	3
r_3	{C, D}	3
r_4	{B, D}	5

Table 3. The set of all sensitive transactions

TID	Transaction	Tdegree(T_q)
T01	{A, B, C, D}	3
T04	{B, C, D, E}	2
T05	{B, D, F, H}	2
T06	{A, B, D, F, H}	3
T08	{A, B, C, D}	3

The proposed sanitization algorithm, Maximum Item Conflict First (MICF), initially scans the database to load all sensitive transactions *db* in the main memory. Therefore, the sanitization process sanitizes transactions in the main memory instead of on the disk. For each restrictive itemset *X*, MICF extracts db_X from *db* and sorts transactions in increasing order of $Tdegree(T_q)$. MICF employs an index lookup table to efficiently sort transactions of db_X and to reduce the memory space requirement. Each transaction of db_X is associated with a pair of two values in the lookup table: the first is the index value pointing to the transaction and the second is the value of $Tdegree(T_q)$.

3.2. Sanitization Algorithm

Definition 3.5. Let r_j be the *j*-th restrictive itemset in *RI*, where $1 \leq j \leq |RI|$. If the sanitization process is handling r_j, the remaining set of restrictive itemsets is denoted as RI'. That is, $RI' = \{r_{j+1}, r_{j+2}, \ldots, r_{|RI|}\}$.

Definition 3.6. For the restrictive itemset r_j, the degree of conflict of an item i_p in a sensitive transaction T_q is the number of restrictive itemsets in RI', in which each restrictive itemset contains i_p and is included in T_q, denoted as $Idegree(i_p, r_j, T_q)$, where $i_p \in r_j \subseteq T_q$.

For example, in Table 1 and Table 2, for the first restrictive itemset $r_1 = \{D, F\}$, $Idegree(\{D\}, r_1, T06) = 1$ and $Idegree(\{F\}, r_1, T06) = 0$. Deleting the item $\{F\}$ from *T06*, does not decrease the support value of any other restrictive itemset. However, removing the item $\{D\}$ from *T06*, also reduces the occurrence count of the fourth restrictive itemset, $\{B, D\}$, by one.

The value of item conflict degree in a transaction indicates the number of restrictive itemsets whose support value is decreased simultaneously if the item is removed from the transaction. For each sensitive transaction, removing the distinct victim item leads to a different set of frequent itemsets being recovered from the released database with the same minimum support threshold. Therefore, the third sub-problem of the sanitization process usually plays the most important role in the sanitization algorithm. To reduce the number of deleted items from the source database, MICF selects a victim item which has the maximum degree of conflict among items in a sensitive transaction. Deleting the victim item from a sensitive transaction simultaneously influences a larger number of restrictive itemsets to reduce their support value than other itemsets in the transaction. The reason is that each restrictive itemset affected contains the victim item and is a subset of the sensitive transaction. If more than one item has the maximum degree of conflict, MICF selects the item with the lowest support value as the victim item.

The pseudo-code of the MICF algorithm is as follows.

Algorithm MICF

Input: (1) *DB*: a source transaction database, (2) ψ: a privacy threshold, and (3) *RI*: the set of all restrictive itemsets

Output: DB': released transaction database

Procedure:

```
// Victim: store the victim item
// MaxIdegree: store the maximum value of the conflict degree among items in a transaction
1.  sort(RI); // sort $r_j$ in increasing order of $Sup(r_j)$
2.  Extract db from DB into main memory; // first scanning DB and
    calculate each Tdegree($T_q$)
3.  foreach $r_j$ in RI
4.      RI := RI - $r_j$
5.      Extract $db_{r_j}$ from db;
6.      sort($db_{r_j}$); // sort $db_{r_j}$ in increasing order of Tdegree($T_q$)
7.      N_repeat := $\lceil |db_{r_j}| \times (1-\psi) \rceil$;
8.      for k:=1 to N_repeat
9.          $T_q$ := k-th transaction in $db_{r_j}$;
10.         MaxIdegree := -1;
11.         foreach $i_p \in r_j$  // $r_j \subseteq T_q$
12.             calculate Idegree($i_p$, $r_j$, $T_q$);
13.             if (Idegree($i_p$, $r_j$, $T_q$)>MaxIdegree)
14.                 MaxIDegree := Idegree($i_p$, $r_j$, $T_q$);
15.                 Victim := $i_p$;
16.             elseif (Idegree($i_p$, $r_j$, $T_q$)==MaxIdegree)
17.                 if support(Victim)>support($i_p$)
```

```
18.                   Victim := $i_p$;
19.          delete Victim from $T_q$;
20. foreach $T_q \in DB$ // second scanning DB
21.      if $T_q$ is modified as $T_q'$ in db
22.          write $T_q'$ to DB';
23.      else
24.          write $T_q$ to DB';
25. return DB';
```

Selecting a restrictive itemset with the lowest support value to sanitize can reduce the number of deleted itemsets in the source database. Therefore, In Line 1, MICF sorts restrictive itemsets in increasing order of their support values. For a certain restrictive itemset to reduce the misses cost, MICF also sorts the sensitive transactions in increasing order of their degrees of conflict as shown in Line 6.

In Line 7, the process deletes $\lceil |db_{r_j}| \times (1-\psi) \rceil$ items from sensitive transactions for each restrictive itemset, where $|db_{r_j}|$ is the occurrence count of r_j in *db*. If an arbitrary later restrictive itemset $r_{j'}$ intersects with r_j and a restrictive transaction containing both the itemsets, by removing an intersected item from the transaction, the two support counts of r_j and $r_{j'}$ simultaneously will be decreased by one. In later pass for $r_{j'}$, in Line 5, the process extracts $db_{r_{j'}}$ from *db*, the transaction number of $db_{r_{j'}}$ must be less than the support count of $r_{j'}$ in *DB*. Consequently, the number of items removed from the source database can be reduced.

In Lines 8 to 19, MICF sanitizes sensitive transactions repeatedly until satisfying the requirement of the privacy threshold. However, users can still recover the hidden restrictive itemsets from the released database by using an Apriori-like tool with a low *minSup*. To guarantee no hiding failure occurrence, the privacy threshold must be set to 0%.

Example 3.1. Consider the sample transaction database as listed in Table 1, $\psi = 0\%$, and four restrictive itemsets listed in Table 2 by increasing order of their support counts.

First, MICF loads *db* into main memory and calculates the conflict degree of each transaction as listed in Table 3. For the first restrictive itemset {*D*, *F*}, MICF creates the index lookup table of $db_{\{D, F\}}$ from *db* and sorts the table in increasing order of the degree of transaction conflict as listed in Table 4. The occurrence count of {*D*, *F*} is two; it must be reduced to zero to hide the itemset without hiding failure. Then, for the first restrictive itemset r_1, select the sensitive transaction to sanitize in sequence. The maximum value of item conflict degree is max(*Idegree*({i_p}, r_1, *T05*)) = *Idegree*({*D*}, {*D*, *F*}, *T05*) = 1. Therefore, MICF deletes {*D*} from *T05* and selects the next transaction *T06* for sanitization until the support value achieves 0%.

Table 4. The sorted index lookup table of $db_{\{D, F\}}$

TID	Tdegree(T_q)
T05	2
T06	3

3.3. Handling very Large Databases

MICF is a memory-based algorithm. For a very large database, when the size of the set of sensitive transactions, *db*, exceeds the current available memory space, MICF will suffer from a large amount of page swaps between disk and main memory. Nonetheless, an algorithm, called MICF′, which combines a partition approach with MICF can be applied to handle such cases.

Definition 3.7. Let *RI* be the set of all restrictive itemsets in *DB*. Let DB^j be the *j*-th partition of *DB*, where $1 \le j \le m$, $\sum_{j=1}^{m} |DB^j| = |DB|$ and $\bigcup_{j=1}^{m} DB^j = DB$. The set of all sensitive transaction in DB^j is denoted as db^j. That is, $db^j = \{T_q \in DB^j \mid \exists X \in RI, X \subseteq T_q\}$.

Instead of loading *db* once, MICF′ loads db^j into the main memory several times.

Let *MEM* be the size of the current available memory space, S_j be the size of db^j and S_j' = $S_j + \Delta_j$, where $1 \le j \le m$ and Δ_j is the max memory requirement to store the index lookup table for an arbitrary restrictive itemset in *RI*. The upper bound of Δ_j is $\max\{|db^j| \times 2 \times \text{sizeof(int)} \mid 1 \le j \le m\}$, where sizeof(int) is the size in bytes of an integer variable. If $\max\{S_j' \mid 1 \le j \le m\} < MEM$, MICF′ can sanitize a very large database in the main memory based on the partition technique.

During the sanitization process, MICF′ loads all sensitive transactions of the current partition into the main memory each time. Then, MICF′ employs the maximum conflict first strategy to sanitize. It removes the restrictive itemsets if their support values are decreased to the privacy threshold. After sanitizing a partition, MICF′ loads all sensitive transactions of the next partition and deals with the remaining restrictive itemsets.

Fig. 1 is a sketch for MICF′. In this example, MICF′ loads all sensitive transactions from the second partition DB^2 into the main memory as shown in the slash area in Fig. 1. Then, for each r_j in *RI*, MICF′ creates an index lookup table of $db^2_{r_j}$ and removes victim items from db^2, in which each transaction is indexed by the lookup table of $db^2_{r_j}$, until r_j becomes non-restrictive or no sensitive transaction appears in db^2. After $db^2_{r_j}$ has been sanitized, MICF′ loads DB^3 into the main memory. The process is repeated until *RI* becomes an empty set. Without the redundancy, this study omits the detailed algorithm of MICF′.

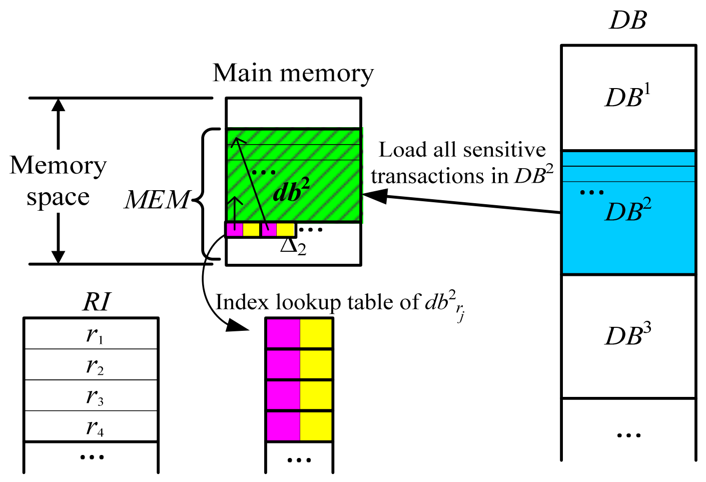


Fig. 1. MICF′ performs on a very large database

4. Time Complexity Analysis

4.1. Sanitization Rate

Definition 4.1. If removing an item from a transaction results in a reduction by one of the support count of some restrictive itemsets, where the current support count of each restrictive itemset r_j is greater than $\left\lfloor |db_{r_j}| \times \psi \right\rfloor$, this removal is called a valid item-sanitization; otherwise it is called an invalid item-sanitization. Clearly, the support count of r_j reduces one after the removal.

Theorem 4.1. Let a sanitization process have no invalid item-sanitization. Let r_j be an arbitrary restrictive itemset in *RI*, where $1 \le j \le |RI|$. Let privacy threshold be ψ. The upper bound and the lower bound of the valid item-sanitization number of the sanitization process for concealing all restrictive itemsets are $\sum_{j=1}^{|RI|} \left\lceil |db_{r_j}| \times (1-\psi) \right\rceil$ and $\max_{r_j \in RI} \left\lceil |db_{r_j}| \times (1-\psi) \right\rceil$, respectively.

Proof. According to Definition 4.1, a valid item-sanitization can reduce the support count of at least one restrictive itemset by one. The value $|db_{r_j}|$ is equal to the support count of r_j in the source database.

(1) Upper bound: In the worst case, each valid item-sanitization decreases the support of only one restrictive itemset by one. For concealing an arbitrary restrictive itemset, it is necessary to decrease its support count by removing $\left\lceil |db_{r_j}| \times (1-\psi) \right\rceil$ items from the transactions. Therefore, the whole sanitization process requires at most $\sum_{j=1}^{|RI|} \left\lceil |db_{r_j}| \times (1-\psi) \right\rceil$ valid item-sanitizations. After $\sum_{j=1}^{|RI|} \left\lceil |db_{r_j}| \times (1-\psi) \right\rceil$ valid item-sanitization times, no itemset is restrictive. The ($\sum_{j=1}^{|RI|} \left\lceil |db_{r_j}| \times (1-\psi) \right\rceil$+1)-th time deletion is invalid. Therefore, the upper bound of sanitization process is

$\sum_{j=1}^{|RI|} \lceil |db_{r_j}| \times (1-\psi) \rceil$.

(2) Lower bound: In the best case, each valid item-sanitization can decrease the support of all restrictive itemsets by one. Using the privacy threshold, ψ, an arbitrary restrictive itemset requires decreasing its support count by $\lceil |db_{r_j}| \times (1-\psi) \rceil$. Therefore, the sanitization process requires at least $\max_{r_j \in RI} \lceil |db_{r_j}| \times (1-\psi) \rceil$ valid item-sanitizations. When the number of deleted items is less than $\max_{r_j \in RI} \lceil |db_{r_j}| \times (1-\psi) \rceil$, there exists at least one restrictive itemset that has a support value greater than ψ. Therefore, the lower bound of sanitization process is $\max_{r_j \in RI} \lceil |db_{r_j}| \times (1-\psi) \rceil$.

Q.E.D.

Definition 4.2. Let DelItem(*DB*) be the total number of items deleted using a sanitization process on *DB* with a set of restrictive itemsets *RI*. The percentage of DelItem(*DB*) to $\sum_{j=1}^{|RI|} \lceil |db_{r_j}| \times (1-\psi) \rceil$ is called the sanitization rate (SR). That is, SR = $\frac{\text{DelItem}(DB)}{\sum_{j=1}^{|RI|} \lceil |db_{r_j}| \times (1-\psi) \rceil}$.

The sanitization rate can be employed to measure the impact on the source database. When performing a sanitization process on a source database with a set of restrictive itemsets and a certain privacy threshold, a process with a high SR has a greater impact on the source database than one with a low SR. Minimizing the SR value means that the sanitization process has a minimum impact on the source database. For each valid item-sanitization, MICF selects a victim item of a sensitive transaction, of which the number of restrictive itemsets containing the item and included in the transaction is maximal. The support counts of these restrictive itemsets are reduced by one, while the item is deleted from the transaction. The total number of valid item-sanitizations can be reduced. Therefore, MICF has a low sanitization rate.

4.2. Time Complexity

Theorem 4.2. The running time of the MICF algorithm is $O(n_1 \times n_2 \times (\log n_2 + n_1 \times s_1^2 \times \log s_2))$, where $n_1 = |RI|$, s_1 is the maximum length of restrictive itemsets in RI, $n_2 = |DB|$, and s_2 is the maximum length of transactions in DB.

Proof. Let DB be the source transaction database, n_1 be the number of restrictive itemsets RI, s_1 be the maximum length of restrictive itemset in RI, n_2 be the number of transactions in DB and let s_2 be the maximum length of transactions in DB. MICF consists of four sub-procedures. According to the MICF algorithm in Section 3.2, the time complexity analysis is as follows:

(1) In Line 1, sorting the restrictive itemsets by support counts requires $O(n_1 \times \log n_1)$. In Line 2 of the MICF algorithm, for each transaction in DB, the algorithm searches each restrictive itemset to determine whether it is a subset of the transaction. If the transaction T_q is sensitive, MICF accumulates its $Tdegree(T_q)$ and loads T_q into the main memory. For each transaction T_q, MICF requires $O(n_1 \times s_1)$ binary searches to match individual item on T_q. The search cost of one binary search on T_q is $O(\log s_2)$. Therefore, the execution time complexity of n_2 transactions is $O(n_1 \times s_1 \times n_2 \times \log s_2)$. The total time complexity of Lines 1 and 2 is $O(n_1 \times \log n_1 + n_1 \times s_1 \times n_2 \times \log s_2)$. Considering that $n_1 \times \log n_1 << n_1 \times s_1 \times n_2 \times \log s_2$, the running time of the sub-procedure can be simplified to $O(n_1 \times s_1 \times n_2 \times \log s_2)$.

(2) In Line 5, in the worst case, $db_{r_j} = db = DB$. For each restrictive itemset, MICF requires $O(s_1)$ binary searches on n_2 sensitive transactions. We get that extracting db_{r_j} for an r_j requires $O(s_1 \times n_2 \times \log s_2)$. Therefore, the execution time complexity of n_1 restricitve itemsets are $O(n_1 \times s_1 \times n_2 \times \log s_2)$. Lines 4, 6 and 7 are executed n_1 times. Their time complexities are $O(n_1)$, $O(n_1 \times n_2 \times \log n_2)$ and $O(n_1)$, respectively. Considering that $n_1 << n_1 \times n_2 \times \log n_2$, the running time of the sub-procedure can be simplified to $O(n_1 \times s_1 \times n_2 \times \log s_2 + n_1 \times n_2 \times \log n_2)$.

(3) According to Theorem 4.1, Lines 9 to 19 are executed $O(\sum_{j=1}^{|RI|}\lceil |db_{r_j}| \times (1-\psi) \rceil)$ = $O(\sum_{j=1}^{|RI|}|db_{r_j}|)$ times**.** In the worst case, $\sum_{j=1}^{|RI|}|db_{r_j}|$ = $n_1 \times n_2$**.** The time complexity analysis of this sub-procedure focuses on Line 12 because the running time of Line 12 dominates the total running time of Lines 9 to 19. In Line 12, for each sensitive transaction, MICF requires finding out all sub-itemsets in RI'. Executing Line 12 once requires $O(n_1 \times s_1)$ binary searches on a sensitive transaction. Therefore, the time complexity of executing Line 12 once is $O(n_1 \times s_1 \times \log s_2)$. MICF calculates $Idegree(i_p, r_j, T_q)$ $O(n_1 \times n_2 \times s_1)$ times. The running time of this sub-procedure is $O(n_1 \times n_2 \times s_1 \times n_1 \times s_1 \times \log s_2) = O(n_1^2 \times s_1^2 \times n_2 \times \log s_2)$.

(4) In Lines 20 to 24, the time complexity of writing the released database on disk requires $O(n_2 \times s_2)$.

The running time of the MICF algorithm is the sum of running times for each sub-procedure executed. However, the second and third sub-procedures dominate the total execution time of MICF. Therefore, the time complexity of MICF is $O(n_1 \times s_1 \times n_2 \times \log s_2 + n_1 \times n_2 \times \log n_2 + n_1^2 \times s_1^2 \times n_2 \times \log s_2)$. Considering that $n_1 \times s_1 \times n_2 \times \log s_2 << n_1^2 \times s_1^2 \times n_2 \times \log s_2$, the running time of MICF can be simplified to $O(n_1 \times n_2 \times \log n_2 + n_1^2 \times s_1^2 \times n_2 \times \log s_2) = O(n_1 \times n_2 \times (\log n_2 + n_1 \times s_1^2 \times \log s_2))$.

Q.E.D.

5. Experimental Results

To measure the effectiveness of MICF, experiments were conducted on both simulated and real datasets to compare its misses costs and sanitization rates with that of Algo2b, MaxFIA, MinFIA and IGA. All experiments were performed on an AMD Barton ES 2900+ (2000MHz) PC with 1 GB of main memory, running Windows XP Professional. All algorithms were coded in Visual C++ 6.0, and employed the same array structure to store all restrictive transactions in the main memory.

Given a minimum support threshold and an original dataset, the NFP-tree algorithm [16], a varied FP-tree algorithm, was employed to generate all frequent itemsets for the original dataset. A certain number, $|RI|$, of restrictive itemsets were randomly selected from a set of frequent itemsets in each dataset with a length between three and eight, in which none of the itemsets was a subset of the other. To measure the misses cost of each algorithm in the same no hiding failure situation for each restrictive itemset, each algorithm sanitized transactions in sequence until the support value of the restrictive itemset became zero ($\psi = 0\%$).

Table 5 lists a summary of the datasets used in this experiment. Each transaction in these datasets was transformed from binary format to the text format "Transaction length n, Item$_1$, Item$_2$, …, Item$_n$". Table 6 lists the characteristics of the four sets of restrictive itemsets.

Table 5. Characteristics of four experimental datasets

Dataset	Transaction number	Distinct items	Average length	Shortest length	Longest length	Dataset size
T10.I6.D100k.N500	100,000	495	10.5	1	30	4.42MB
T20.I10.D100k.N500	100,000	496	19.9	1	49	7.97MB
BMS-WebView-1	59,602	497	2.5	1	267	0.78MB
BMS-WebView-2	77,512	3,340	4.6	1	161	1.86MB

Table 6. Characteristics of used restrictive itemsets in four experimental datasets

Restrictive itemsets in which dataset	|*RI*|	$\min_{r_j \in RI}\{\frac{\|DB_{r_j}\|}{\|DB\|}\}$	$\max_{r_j \in RI}\{\frac{\|DB_{r_j}\|}{\|DB\|}\}$	Average length	Shortest length	Longest length
T10.I6.D100k.N500	800	0.040%	0.132%	5.4	3	8
T20.I10.D100k.N500	400	0.100%	0.567%	5.8	3	8
BMS-WebView-1	800	0.065%	0.589%	4.5	3	6
BMS-WebView-2	800	0.024%	0.315%	5.2	3	6

5.1 Comparison of Misses Cost

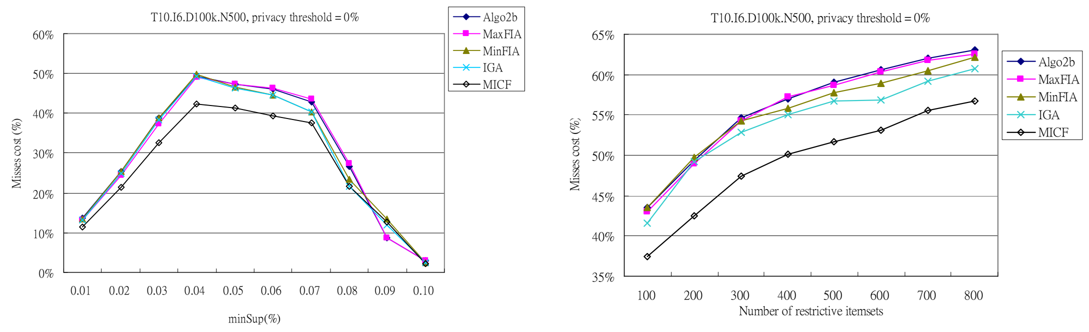
The IBM synthetic data generator [27] was used to generate two datasets, T10.I6.D100k.N500 and T20.I10.D100k.N500. The parameters used are similar to those in [12] to produce transaction-like datasets for market-basket analysis.

Figs. 2(a) and 3(a) plot the performance curves of misses cost associated with these algorithms over various *minSup* applied to T10.I6.D100k.N500 and T20.I10.D100k.N500, respectively, when the privacy threshold is 0%. In Figs. 2(b) and 3(b), to compare the misses costs associated with these algorithms over various numbers of restrictive itemsets, the values of *minSup* are set to 0.04% and 0.1%, respectively.

In Fig. 2(a), the range of *minSup* is between 0.01% and 0.1% with |*RI*| = 200. Fig. 2(a) demonstrates that MICF performs the best and attains the lowest misses cost, except in the case where *minSup* = 0.09%. Fig. 2 shows that MICF has the best result among these alternative algorithms in most scenarios.

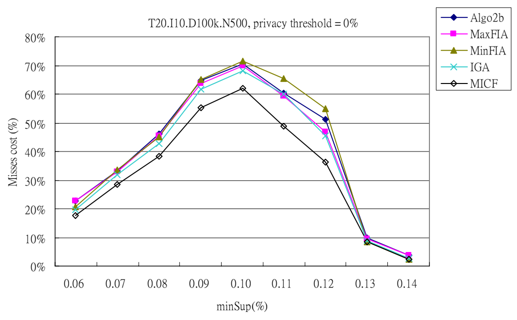
In Fig. 3(a), the range of *minSup* is between 0.06% and 0.14%, MICF performs the best among the five algorithms. For example, the misses cost of MICF is less than that of IGA by 6.02% when *minSup* = 0.1%. In Fig. 3(b), MICF has the lowest misses cost, except in the case where |*RI*| = 400.

A sanitization process can also reduce support counts of some non-restrictive itemsets in ~*RI*. Let ~*RI*′ denote the set of non-restrictive itemsets in *DB*, which support values decrease after a sanitization process. A process of frequent itemset discovery with a lower *minSup* has a higher possibility to recover itemsets in ~*RI*′. Therefore, in Fig. 2(a), for a low minimum support value between 0.01% and 0.04%, the misses costs increase with increasing *minSup*. However, for a higher *minSup* between 0.05% and 0.1% in Fig. 2(a), the misses cost decreases with increasing *minSup*. Decreasing the support value of an infrequent itemset both in *DB* and in *DB*′ does not concern the misses cost. The possibility of an itemset in ~*RI*′ becoming infrequent increases both in *DB* and in *DB*′ when *minSup* increases. Therefore, in Fig. 2(a) the misses cost curves increase then decrease with increasing *minSup*. The curves in Fig. 3(a) are similar.

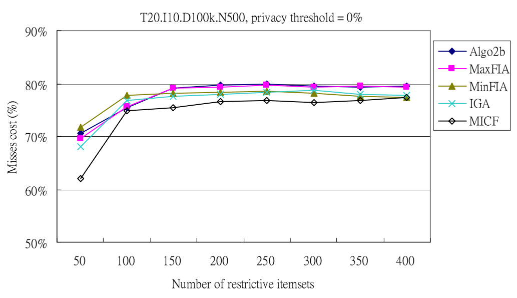


(a) (b)

Fig. 2. Comparison of the misses cost on T10.I6.D100k.N500 with (a) |*RI*| = 200 and (b) *minSup* = 0.04%



(a)

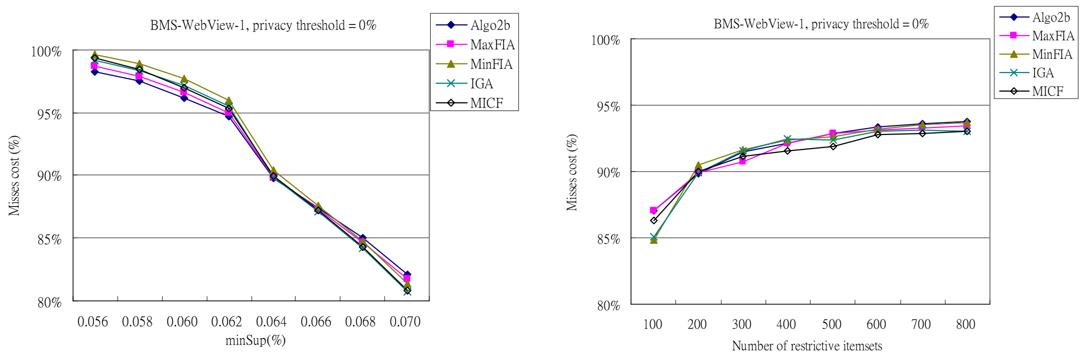


(b)

Fig. 3. Comparison of the misses cost T20.I10.D100k.N500 with (a) |*RI*| = 50 and (b) *minSup* = 0.1%

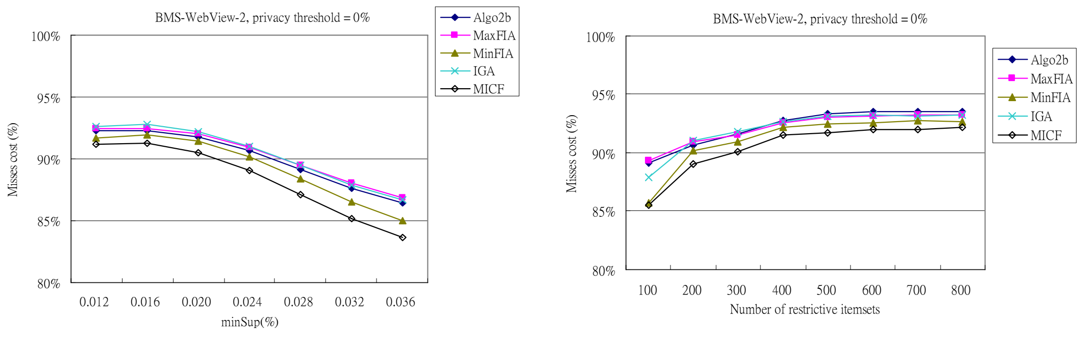
This study also compares the misses cost of algorithms running on two real datasets, BMS-WebView-1 and BMS-WebView-2, which are real datasets of several months' click stream data from two e-commerce web sites [28]. As shown in Figs. 4(a) and 4(b), the algorithms ran on BMS-WebView-1 over various *minSup* values and various numbers of restrictive itemsets, respectively. In Fig. 4(a), with |*RI*| = 200, Algo2b has the lowest misses cost for a minimum support below 0.064%. For a higher support value between 0.066% and 0.7%, IGA and MICF outperform other methods. In Fig. 4(b) with *minSup* = 0.064%, MICF performs the best among five methods when |*RI*| between 400 and 800. MinFIA, Algo2b and MaxFIA have the lowest misses cost with |*RI*| are 100, 200 and 300, respectively.

For BMS-WebView-2 with |*RI*| = 200, shown in Fig. 5(a), MICF has the best performance. As shown in Fig. 5(b), MICF outperforms IGA when *minSup* = 0.024%.



(a) (b)

Fig. 4. Comparison of the misses cost on BMS-WebView-1 with (a) |*RI*| = 200 and (b) *minSup* = 0.064%

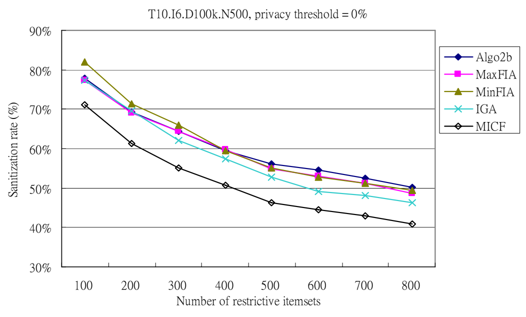


(a) (b)

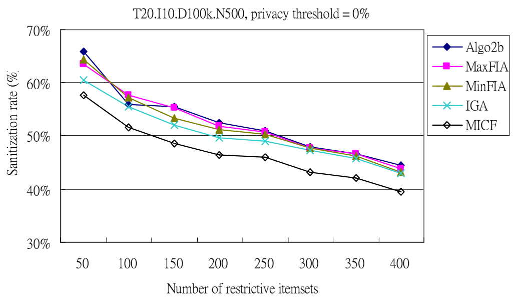
Fig. 5. Comparison of the misses cost on BMS-WebView-2 with (a) |*RI*| = 200 and (b) *minSup* = 0.024%

5.2 Comparison of Sanitization Rate

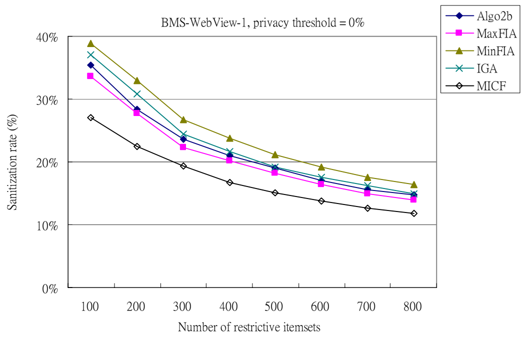
The sanitization rate was used to evaluate the impact on the source datasets. MICF and four alternate algorithms ran on four datasets (T10.I6.D100k.N500, T20.I10.D100k.N500, BMS-WebView-1 and BMS-WebView-2) with minimum support 0.04%, 0.1%, 0.064% and 0.024%, respectively. Each experiment set ψ to be 0%. In Figs. 6(a) to 6(d), MICF always has the lowest sanitization rate. The sanitization rates of MICF are less than that of IGA by at least 4.8%, 2.7%, 3.1% and 4.6%, respectively.



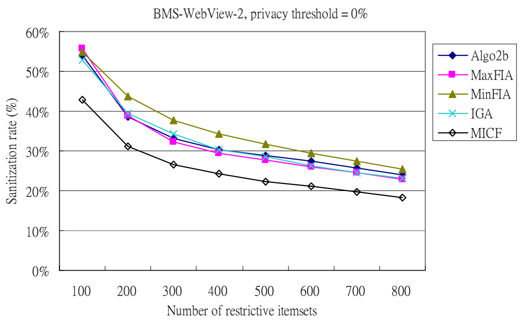
(a)



(b)



(c)

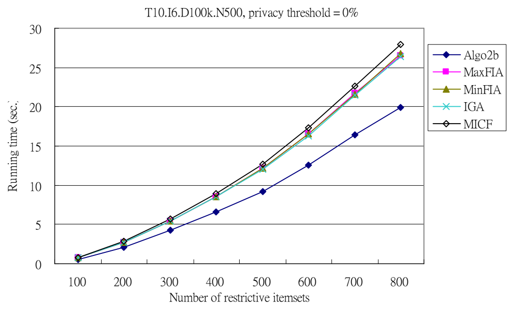


(d)

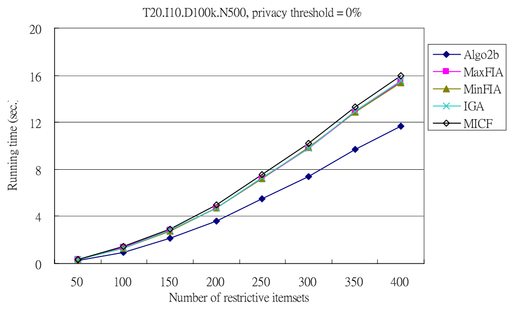
Fig. 6. Comparison of the sanitization rate on (a) T10.I6.D100k.N500 with *minSup* = 0.04%, (b) T20.I10.D100k.N500 with *minSup* = 0.1%, (c) BMS-WebView-1 with *minSup* = 0.064%, and (d) BMS-WebView-2 with *minSup* = 0.024%

5.3 Comparison of running time

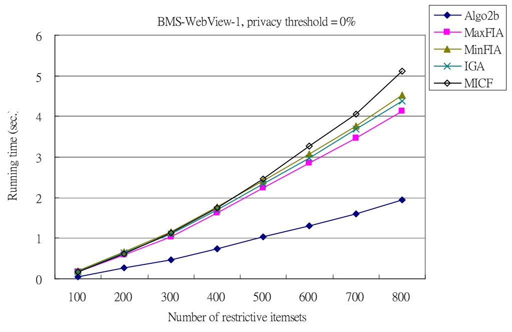
Figs. 7(a) to 7(d) present the running time curves of all algorithms on T10.I6.D100k.N500, T20.I10.D100k.N500, BMS-WebView-1 and BMS-WebView-2, respectively. To reduce the effect of disk reading and writing, the running time excluded the execution time of the disk I/O. The running time of MICF sub-linearly increased with the growth of the number of restrictive itemsets. In Fig. 7(c), Algo2b performed the fastest, followed in order of efficiency by MaxFIA, IGA, MinFIA, and MICF. In the other three figures, the performance of the three algorithms MaxFIA, IGA, and MinFIA were too close to be ordered. Although MICF had the longest running time among the five approaches, it ran quickly on the four test datasets. For example, in the worst case in Fig. 7(d), MICF only required 29.1 seconds to sanitize 800 restrictive itemsets.



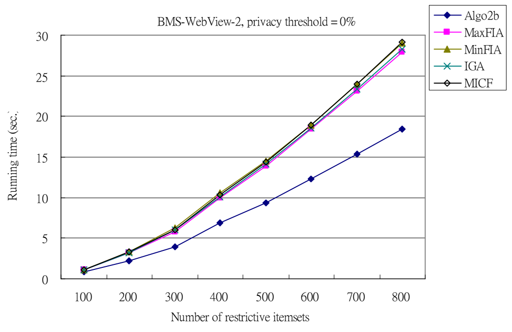
(a)



(b)



(c)



(d)

Fig. 7. Comparison of running time on (a) T10.I6.D100k.N500, (b) T20.I10.D100k.N500, (c) BMS-WebView-1, and (d) BMS-WebView-2

6. Conclusions

Data mining techniques can discover useful information from databases. Accurate input data leads to meaningful mining results, but problems arise when users provide fictitious data to protect their privacy. In this competitive but also cooperative business environment, companies need to share information with others, while at the same time protecting their own confidential strategic. For this kind of data sharing to be possible, this study proposes the Maximum Item Conflict First (MICF) algorithm to reduce the impact on the source database for preserving privacy in mining frequent itemsets. MICF employs the strategy of maximum degree of item conflict first to simultaneously decrease the support of the maximum number of restrictive itemsets. In our experimental results, MICF outperforms all other algorithms in several simulated and real datasets on misses costs for most cases. Furthermore, MICF always has the lowest sanitization rates than the other four methods.

The intention is to develop superior algorithms to further reduce the misses cost without hiding failure to protect sensitive data and without generating any artificial frequent itemset.

Acknowledgements

The authors would like to acknowledge the helpful comments made by the anonymous reviewers of this paper and thank Blue Martini Software, Inc. for kindly providing the BMS datasets.

References

[1] Cabena P, Hadjinian P, Stadler R, Verhees J, Zanasi A. Discovering data mining from concept to implementation. New Jersey: Prentice Hall PTR; 1998.

[2] Braha D (editor). Data mining for design and manufacturing: methods and applications. New York: Springer-Verlag; 2002.

[3] Kantardzic M. Data mining: concepts, models, methods, and algorithms. New York: John Wiley & Sons; 2002.

[4] Ishino Y, Jin Y. An information value based approach to design procedure capture. Advanced engineering informatics 2006; 20(1): 89-107.

[5] Soenjaya J, Hsu W, Lee ML, Lee T. Mining wafer fabrication: framework and challenges. In: Kantardzic MM, Zurada J, editors. Next generation of data-mining application. New York: John Wiley & Sons; 2005. p. 17-40.

[6] Saitta S, Raphael B, Smith IFC. Data mining techniques for improving the reliability of system identification. Advanced engineering informatics 2005; 19(4): 289-98.

[7] Atallah M, Bertino E, Elmagarmid A, Ibrahim M, Verykios V. Disclosure limitation of sensitive rules. Proceedings of 1999 workshop on knowledge and data engineering exchange, Chicage, IL; 1999. p. 45-52.

[8] Kantarcioglu M, Clifton C. Privacy-preserving distributed mining of association rules on horizontally partitioned data. IEEE transactions on knowledge and data engineering 2004; 16(9): 1026-37.

[9] Evfimievski A, Srikant R, Agrawal R, Gehrke J. Privacy preserving mining of association rules. Proceedings of 8th ACM SIGKDD international conference on knowledge discovery and data mining. Alberta, Canada; 2002. p. 217-28.

[10] Oliveira SRM, Zaïane OR. Privacy preserving frequent itemset mining. Proceedings of IEEE ICDM workshop on privacy, security and data mining. Maebashi City, Japan; 2002. p. 43-54.

[11] Agrawal R, Imielinski T, Swami A. Mining association rules between sets of items in large databases. Proceesings of 1993 ACM SIGMOD international conference on management of data, Washington, DC; 1993. p. 207-16.

[12] Agrawal R, Srikant R. Fast algorithms for mining association rules. Proceedings of 20th international conference on very large data bases, Santiago, Chile; 1994. p. 487-99.

[13] Park JS, Chen MS, Yu PS. An effective hash-based algorithm for mining association rules. Proceedings of 1995 ACM-SIGMOD international conference on management of data. San Jose, CA; 1995. p. 175-86.

[14] Brin S, Motwani R, Ullman JD, Tsur S. Dynamic itemset counting and implication rules for market basket data. Proceedings of 1997 ACM SIGMOD international conference on management of data, Tucson, AZ; 1997. p. 255-64.

[15] Han J, Pei J, Yin Y, Mao R. Mining frequent pattern without candidate generation: a frequent pattern tree approach. Data mining and knowledge discovery 2004; 8(1): 53-87.

[16] Li YC, Chang CC. A new FP-tree algorithm for mining frequent itemsets. In: Chi CH, Lam KY, editors. Lecture notes in computer science 3309, New York: Springer-Verlag; 2004. p. 266-77.

[17] Clifton C. Using sample size to limit exposure to data mining. Journal of Computer Security 2000; 8(4): 281-307.

[18] Johnsten T, Raghavan VV. Impact of decision-region based classification mining algorithms on database security. Proceedings of IFIP WG 11.3 13th international conference on database security. Seattle, WA; 1999. p. 177-91.

[19] Lindell Y, Pinkas B. Privacy preserving data mining. In: Bellare M, editor. Lecture notes in computer science 1880, New York: Springer-Verlag; 2000. p. 36-54.

[20] Agrawal D, Aggarwal C. On the design and quantification of privacy preserving data mining algorithms. Proceedings of 20th ACM symposium on principles of database systems, Santa Barbara, CA; 2001. p. 247-55.

[21] Oliveira SRM, Zaïane OR. Algorithms for balancing privacy and knowledge discovery in association rule mining. Proceedings of 7th international database engineering and applications symposium. Hong Kong, China; 2003. p. 54-63.

[22] Verykios VS, Elmagarmid AK, Bertino E, Saygin Y, Dasseni E. Association rule hiding. IEEE transactions on knowledge and data engineering 2004; 16(4): 434-47.

[23] Dasseni E, Verykios VS, Elmagarmid AK, Bertino E. Hiding association rules by using confidence and support. In: Moskowitz IS, editor. Lecture notes in computer sciences 2137, New York: Springer-Verlag; 2001. p. 369-83.

[24] Saygin Y, Verykios VS, Clifton C. Using unknowns to prevent discovery of association rules. ACM SIGMOD Record 2001; 30(4): 45-54.

[25] Saygin Y, Verykios VS, Elmagarmid AK. Privacy preserving association rule mining. Proceedings of 12th international workshop on research issues in data mining. San Jose, CA; 2002. pp. 151-58.

[26] Oliveira SRM, Zaïane OR, Saygin Y. Secure association rule sharing. In: Dai H, Srikant R, Zhang C, editors. Lecture notes in computer sciences 3056, New York: Springer-Verlag; 2004. p. 74-85.

[27] IBM Almaden Research Center. Synthetic data generation code for associations and sequential patterns. http://www.almaden.ibm.com/software/projects/hdb/Resources/datasets/syndata.html

[28] Zheng Z, Kohavi R, Mason L. Real world performance of association rule algorithm. Proceedings of 7th ACM-SIGKDD international conference on knowledge discovery and data mining. San Francisco, CA; 2001. p. 401-6.