Frequent Itemset Hiding Algorithm Using Frequent Pattern Tree Approach

by

Rami Alnatsheh

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy
in

Computer Information Systems

Graduate School of Computer and Information Sciences

Nova Southeastern University

2012

An Abstract of a Dissertation Submitted to Nova Southeastern University in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy

Frequent Itemset Hiding Algorithm Using Frequent Pattern Tree Approach

by
Rami Alnatsheh

December 2012

A problem that has been the focus of much recent research in privacy preserving data-mining is the frequent itemset hiding (FIH) problem. Identifying itemsets that appear together frequently in customer transactions is a common task in association rule mining. Organizations that share data with business partners may consider some of the frequent itemsets sensitive and aim to hide such sensitive itemsets by removing items from certain transactions. Since such modifications adversely affect the utility of the database for data mining applications, the goal is to remove as few items as possible. Since the frequent itemset hiding problem is NP-hard and practical instances of this problem are too large to be solved optimally, there is a need for heuristic methods that provide good solutions. This dissertation developed a new method called Min_Items_Removed, using the Frequent Pattern Tree (FP-Tree) that outperforms extant methods for the FIH problem. The FP-Tree enables the compression of large databases into significantly smaller data structures. As a result of this compression, a search may be performed with increased speed and efficiency.

To evaluate the effectiveness and performance of the Min_Items_Removed algorithm, eight experiments were conducted. The results showed that the Min_Items_Removed algorithm yields better quality solutions than extant methods in terms of minimizing the number of removed items. In addition, the results showed that the newly introduced metric (normalized number of leaves) is a very good indicator of the problem size or difficulty of the problem instance that is independent of the number of sensitive itemsets.

Acknowledgements

In the name of God (Allah), Most Gracious, Most Merciful…and they shall say: "Praise be to Allah, who hath guided us to this: never could we have found guidance, had it not been for the guidance of Allah" The Holy Quran (7:43)

I would like to express my deep appreciation and gratitude to my advisor, Dr. Sumitra Mukherjee, for the patient guidance and mentorship he provided to me, all the way from when I started scouting for ideas for my research effort, through to completion of this degree. Dr. Mukherjee's intellect and down to earth humility is unmatched, I am very fortunate to have had the chance to work with him. I would also like to thank my committee members, Dr. Michael Laszlo, and Dr. Junping Sun for their guidance, and useful suggestions.

I thank both of my parents who have always encouraged me during my childhood to continue my studies and aspire for the best education.

Finally, I dedicate this dissertation to my wife, Shereen, who has supported me with her love and encouragement and provided all the help she could during the past five years, and to my adorable child Ahmad.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Advances in the technologies that facilitate the acquisition, storage, and mining of data have provided an opportunity for businesses to store copious amounts of data. Such data may be analyzed in order to gain a deeper insight and to improve an organization's efficiency (Menon, Sarkar, & Mukherjee, 2005).

This data may also be shared between organizations to gain competitive advantage (Menon & Sarkar, 2007). Despite this advantage, the disclosure of confidential information may also put the original owner of the database at a disadvantage. In order to preserve privacy and eliminate any unintended disclosures of information, it is necessary for the database to be modified before it is shared with any other organization.

A transactional database contains a set of tuples, each of which represents a transaction that contains multiple items. These items may represent, for example, products. A subset of items in a transaction is usually called an itemset. An itemset that is contained in no fewer than a predefined number of transactions is considered a frequent itemset.

To generate association rules, several techniques for knowledge discovery and mining in transactional databases are used. The most prominent techniques utilize the recurring patterns of items within a transactional database, i.e., the frequent itemsets (Menon &

Sarkar, 2007). Generating frequent itemsets is an intermediate step to generating association rules and consequently to revealing hidden or new relations. This suggests that hiding the frequent itemsets that are associated with certain association rules that the owner of the database would like to conceal would preserve the privacy of itemsets containing sensitive information when the database is shared (Menon & Sarkar, 2007; Oliveira & Zaïane, 2002).

In order to hide a sensitive frequent itemset within a transactional database, the support level (i.e. the number of transactions that support that itemset within the database) must be brought below a certain threshold that is known as the mining threshold. To accomplish this, the original database must be modified. One possible way to hide frequent itemsets is to delete certain items from transactions. Items that are supported in a sensitive itemset are evaluated for removal. This reduces the support level for a sensitive itemset (Oliveira & Zaïane, 2002; Verykios, Bertino, Fovino, Provenza, Saygin, & Theodoridis, 2004). Removal of items from the original database may affect its utility. Such an adverse effect may be measured in several ways. Menon, Sarkar, and Mukherjee (2005) introduce the accuracy metric, which takes into account the percentage of transactions that are modified. Others, such as Verykios et al. (2004) and Menon and Sarkar (2007), consider the number of the nonsensitive frequent itemsets that were hidden after the removal process—sanitization—as a measure of the reduced utility of the database.

Whether the objective in solving the frequent itemset hiding problem is to minimize the number of transactions that are modified or to minimize the number of nonsensitive frequent itemsets that are lost, the sanitization of the database is actually achieved by removing certain items from a subset of transactions. This suggests yet another candidate measure for information loss—the total number of items removed from all transactions in the process of concealing the sensitive itemsets. This is the core problem addressed in this dissertation. The measure may be interpreted as the edit distance between the original database and its sanitized version. The aim is to minimize the total number of items removed from all transactions. However, this version of the frequent itemset hiding problem is non-deterministic polynomial-time hard (NP-hard) and practical instances of this problem are too large to be solved optimally.

This dissertation aims to develop a new method, using the Frequent Pattern Tree (FP-Tree) introduced by Han, Pei and Yin (2000). The method attempts to find good solutions to the frequent itemset hiding problem. The FP-Tree enables the compression of large databases into significantly smaller data structures. As a result of this compression, a search can be performed with increased speed and efficiency.

The total number of items that were hidden using the new method was compared to the total number of items that were hidden in other methods suggested in the literature (e.g. the heuristic that minimizes the non frequent itemsets lost in shared databases presented by Menon and Sarkar [2007]). More details about the FP-Tree and the proposed method that made use of the FP-Tree are discussed in the upcoming sections.

Problem Statement and Goal

The problem addressed by this dissertation is the development and evaluation of a new heuristic that attempts to find good solutions for a version of the frequent itemset hiding problem that aims to minimize the total number of items that are removed from all transactions in sanitizing the database.

Problem Description

Definitions

Following the notation mentioned in Menon and Sarkar (2007), $\mathcal{J}$ is defined to be a set of items. Any subset $\mathcal{J}_i \subseteq \mathcal{J}$ is an itemset, and any transaction defined over $\mathcal{J}$ is the tuple $\langle \imath, \mathcal{J}_i \rangle$, where $\imath$ is the transaction label/id and $\mathcal{J}_i$ is the itemset represented in that transaction. A transaction $\langle \imath, \mathcal{J}_i \rangle$ supports an itemset $\mathcal{J}_{\mathcal{J}}$ if $\mathcal{J}_{\mathcal{J}} \subseteq \mathcal{J}_i$. A database $\mathcal{D}$ over $\mathcal{J}$ is a set of transactions over $\mathcal{J}$. The cover of an itemset $\mathcal{J}_i$ in the database $\mathcal{D}$ is the set of transactions of $\mathcal{D}$ that support $\mathcal{J}_i$. The support $\sigma(\mathcal{J}_i, \mathcal{D})$, of an itemset $\mathcal{J}_i$ in the database $\mathcal{D}$ is the number of transactions of $\mathcal{D}$ that supports $\mathcal{J}_i$, or it can be expressed as $\sigma(\mathcal{J}_i, \mathcal{D}) = |\{j | \langle j, \mathcal{J}_j \rangle \in \mathcal{D},\ \mathcal{J}_i \subseteq \mathcal{J}_j\}|$. $\sigma(\mathcal{J}_i, \mathcal{D})$ can be represented simply as σ_i for notational convenience. An itemset is *frequent* if there are at least a predefined minimum number, σ_{min}, of -transactions supporting it. The set of frequent itemsets $\mathcal{F}(\sigma_{min})$ at minimum support level σ_{min} is the set of all itemsets with a minimum support of σ_{min}. The frequent itemset-mining problem is to identify the set $\mathcal{F}(\sigma_{min})$, given a database $\mathcal{D}$

and a minimum support threshold σ_{min}. $\mathcal{F}$ will be used to represent $\mathcal{F}(\sigma_{min})$ for notational convenience.

Define $\mathcal{F}^{\mathcal{R}}(\sigma_{min}) \subseteq \mathcal{F}(\sigma_{min})$ to be a set of frequent patterns that the owner of the database would like to conceal before sharing the database—i.e., sensitive itemsets. A transaction $i \in \mathcal{D}$ is said to be *sanitized* if it is altered in such a way that it no longer supports any itemset in $\mathcal{F}^{\mathcal{R}}(\sigma_{min})$. Hiding an itemset $j \in \mathcal{F}^{\mathcal{R}}(\sigma_{min})$ will involve reducing the support level for j to below σ_{min}, which is considered safe by the owner of the database for that specific itemset. This sanitization process will affect the utility of the database as mentioned earlier.

An example is shown in Table 1 for a database $\mathcal{D}$ over a set of items $\mathcal{J} = \{1,2,3,4,5,6,7,8,9,10\}$. In Table 2 the itemsets that have a support level of 2 or more are displayed, which is the set of frequent itemsets $\mathcal{F}$ for $\sigma_{min} = 2$. For example itemset {1,2,3} is only supported three times in t1, t6, and t8, which gives it a support level of 3 and is included in Table 2, while for itemset {1,8} the support level is 1 since it is only supported in t1, so this itemset will not be shown in Table 2 since it is not a frequent itemset.

Assuming that the owner of the database would like to conceal itemsets {3, 7} and {6, 7}, i.e., $\mathcal{F}^{\mathcal{R}} = \{\{3,7\},\{6,7\}\}$, where $\sigma_{min} = 2$, the goal is to bring the support level of every itemset in $\mathcal{F}^{\mathcal{R}}$ below σ_{min}. This can be done by sanitizing database $\mathcal{D}$. As a consequence of the sanitization process, the utility of the database might be reduced. The reduction in utility can be measured in several ways. The utility measure that was used in this research is the total number of items removed from all transactions.

Table 1. Example Database $\mathcal{D}$

Transaction ID			**Items**			
t1	1	2	3	8		
t2	6	7	8	9		
t3	3	4	7			
t4	4	5	7	8	9	
t5	3	6	7			
t6	1	2	3	4	5	9
t7	3	4	6	7		
t8	1	2	3	4	5	7
t9	8	9				
t10	7	8	9	10		

Table 2. Itemsets With Support Level of 2 or More

Itemsets	**Support Level**	**Itemsets**	**Support Level**	**Itemsets**	**Support Level**
1, 2	3	**1, 5**	2	**3, 7**	4
1, 2, 3	3	**2, 3**	3	**4, 5**	3
1, 2, 3, 4	2	**2, 3, 4**	2	**4, 5, 7**	2
1, 2, 3, 4, 5	2	**2, 3, 4, 5**	2	**4, 5, 9**	2
1, 2, 3, 5	2	**2, 3, 5**	2	**4, 7**	4
1, 2, 4	2	**2, 4**	2	**4, 9**	2
1, 2, 4, 5	2	**2, 4, 5**	2	**5, 7**	2
1, 2, 5	2	**2, 5**	2	**5, 9**	2
1, 3	3	**3, 4**	4	**6, 7**	3
1, 3, 4	2	**3, 4, 5**	2	**7, 8**	3
1, 3, 4, 5	2	**3, 4, 7**	3	**7, 8, 9**	3
1, 3, 5	2	**3, 5**	2	**7, 9**	3
1, 4	2	**3, 6**	2	**8, 9**	4
1, 4, 5	2	**3, 6, 7**	2		

The Constrained Optimization Problem Formulation

The goal of this dissertation is to minimize the total number of items removed in the process of sanitizing a database. The impact on the database was measured by the difference between the number of items in the original database and its sanitized version. For this purpose $S(i)$ is defined as the support level for item i in the original database $\mathcal{D}$ and $S^{'}(i)$ as the support level for item i in the sanitized version of the database $\mathcal{D}^{'}$. $SI(j)$ is defined as the support level for itemset j in the original database and $SI'(j)$ is defined as the support level for itemset j in the sanitized database. Database $\mathcal{D}$ contains n items. The constraint optimization problem is constructed as:

$$minimize \sum_{i=1}^{n} |S'(i) - S(i)| \qquad (1)$$

$$such\ that$$

$$SI'(j) < \sigma_{min}, \quad \forall j \in \mathcal{F}^{R} \qquad (2)$$

The objective function (1) must be minimized to maximize the utility of the database after modification. Constraint (2) is necessary to guarantee that the sensitive itemsets are hidden after sanitization. Again, the version of the frequent itemset hiding problem that is addressed here is NP-hard and practical instances of this problem are too large to be solved optimally.

The problem can be very large if the database contains a large number of items and transactions. Using the FP-Tree to condense the whole transactional database into a much smaller data structure helped in reducing the problem size and facilitated finding a satisfactory solution that is better than the other techniques that were mentioned in extant literature. The following section (Relevance and Significance) will introduce other techniques, mentioned in previous literature, which attempted to find a satisfactory solution to the frequent itemset hiding problem. The FP-Tree approach presented by Han et al. (2000) will also be presented.

Relevance and Significance

This section will discuss the two studies that are relevant to the frequent itemset hiding problem—Menon, Sarkar, and Mukherjee's (2005) maximization of accuracy (total percentage of transactions that are not sanitized) within a shared database and Menon and Sarkar's (2007) minimization of the number of nonsensitive frequent itemsets lost during sanitation before sharing databases. Finally, the FP-Tree will be discussed in terms of how an FP-tree was used to identify a small set of items whose removal leads to the concealment of all sensitive frequent itemsets.

Menon, Sarkar, and Mukherjee (2005) discussed the frequent itemset hiding problem and presented a heuristic that aims to minimize the number of modified transactions in the sanitized version of the database. They defined *accuracy* as the percentage of unmodified transactions in the database that will be shared. Following the same notation as Menon et al. (2005), parameter a_{ij} is defined to be 1 if transaction $i \in \mathcal{D}$ supports itemset $j \in \mathcal{F}^R(\sigma_{min})$, where $\mathcal{F}^R(\sigma_{min})$ is the set of sensitive itemsets. Otherwise, a_{ij} is 0. Variable x_i is set to 1 if transaction $i \in \mathcal{D}$ is sanitized; otherwise, it is 0. Notice that

σ_j is the current support for itemset $j \in \mathcal{F}^R(\sigma_{min})$. The problem that was addressed by Menon et al. (2005) is shown below.

$$minimize \sum_{i \in \mathcal{D}} x_i \tag{3}$$

$$s.t. \sum_{i \in \mathcal{D}} a_{ij} x_i \geq (\sigma_j - \sigma_{min} + 1) \qquad \forall\, j\ \mathcal{F}^R(\sigma_{min}), \tag{4}$$

$$x_i \in \{0,1\} \,\forall\ i \in \mathcal{D} \tag{5}$$

The objective function in (3) tries to minimize the number of transactions that will be sanitized, while constraint (4) guarantees that more than $(\sigma_j - \sigma_{min})$ transactions supporting each sensitive item set $j \in \mathcal{F}^R(\sigma_{min})$ have to be sanitized. Constraint (5) imposes the binary requirement on the x_i variables.

This mainly concerns the total number of transactions that are modified. Menon et al. (2005) successfully found an exact solution to this problem by using an integer programming method. Nevertheless, they did not consider the total number of items that are removed during the sanitization process, the core problem that is addressed by this dissertation.

On the other hand, Menon and Sarkar (2007) considered a novel approach to the frequent itemset hiding problem—minimizing the total number of nonsensitive frequent itemsets that are hidden after the sanitization process. Like Menon, et al. (2005), Menon and Sarkar (2007) followed an integer programming approach, dividing the problem into two smaller problems, which may then be solved in sequence. The heuristic they used produces suboptimal solutions. Their approach was successful in terms of reducing the total number of the nonsensitive frequent itemsets that are hidden after the sanitization process, but, as in Menon, et al. (2005), it did not consider the minimization of the total number of items removed from the database. Again, this is the main problem addressed by this study.

Introduction to the FP-Tree

Most of the frequent pattern mining studies in transactional databases used an Apriori-like candidate set generation approach (Han et al., 2000). The Apriori heuristic attains good performance through reducing the size of the candidate sets. However, the Apriori heuristic might be costly if it is handling a large number of candidate sets. If the patterns are long, the cost of scanning the database increases.

According to Han et al. (2000), the bottleneck of the Apriori-like methods lies at the candidate set generation and test. The use of a different data structure, FP-Tree, and a new algorithm, FP-Growth might help to tackle this bottleneck. The FP-Tree is an extended prefix tree structure for storing compressed information about frequent patterns. It is used as the first step towards mining the complete set of frequent patterns using the FP-Growth method that was suggested by Han et al. (2000). The FP-Tree in general saves the costly database scans in subsequent mining processes. Figure 1 contains the pseudocode that will generate an FP-Tree for a transactional database $\mathcal{D}$ (Han & Kamber, 2006).

Referring to the pseudocode (Figure 1), notice that the FP-Tree algorithm attempts to condense the database $\mathcal{D}$. This will be useful as the number of transactions increases to large numbers. Illustrating the process, for example, consider the database $\mathcal{D}$ that was shown in Table 1. When an FP-Tree is generated from the database $\mathcal{D}$ (Figure 2), this will yield the list L (Table 3). List L contains the list of items in Database $\mathcal{D}$ and their support level. Also, the items within each transaction are reordered and sorted according to their support level in Database $\mathcal{D}$ in descending order (Table 4).

```
Algorithm Generate_FP_Tree(𝒟)

a)  Scan Database 𝒟 and generate L; i.e. the list of items in 𝒟 and their support level.
    For each transaction T ∈ 𝒟
        For each item i ∈ T
                if i ∈ L then increment i.count in L.
                else add i to L and set i.count to 1.
        End For
        Sort L in descending order according to i.count.
    End For

b)  Generate the FP-Tree, i.e. Tree.
    Create the root of Tree and label it as "root".
    For each Transaction T ∈ 𝒟
        Sort items in T according to the order of L, let the new sorted items in T be [p|P], where p is the first element in T
        and P is the remaining list.
        Call insert_tree([p|P]|, Tree)
    End For

c)  Procedure insert([p|P], ∝)
        {
        If ∝ has a child node N such that N.item.name = p.item.name
        Then increment N.item.count by 1
        Else create a new node N where N.item.count = 1

        N.parent = ∝
        }
        If P is not empty then call insert_tree ([P], N) recursively.
```

Figure 1. Algorithm Generate_FP_Tree($\mathcal{D}$).

Table 3. List L, Contains Items and Their Support Level in Descending Order According to the Support Level

Item	Support Level
7	7
3	6
4	5
8	5
9	5
1	3
2	3
5	3
6	3
10	1

Table 4. Sorted Transactions According to the Order of L

Transaction ID	Items				
t1	3	8	1	2		
t2	7	8	9	6		
t3	7	3	4			
t4	7	4	8	9	5	
t5	7	3	6			
t6	3	4	9	1	2	5
t7	7	3	4	6		
t8	7	3	4	1	2	5
t9	8	9				
t10	7	8	9	10		

Examining the FP-Tree in Figure 2, the tree contains multiple nodes from the root node to the leaf nodes. Each node contains an item, and the count of that item is listed between brackets. The count of each item in the node is its count or support level within the transactions that are represented by all the branches where the node is positioned. A branch is defined as the path from the leaf node to the root node. Each branch is marked as B*x*, where *x* is the branch number which starts from 1 to the total number of branches.

A branch can share a node with another branch. For example, the sub-tree that is represented by branches B7 and B8 represents transactions t1 and t6. The node that contains item 3 is supported in both transactions and is positioned in both branches; hence its count is 2. On the other hand there are two nodes that contain item 1 within the same sub-tree; each node is positioned in only one branch; therefore, each node has a count of 1 for item 1. The total count for each item should match the original support level in transactions t1 and t6 for that item. In this case, item 1 has a total count of two.

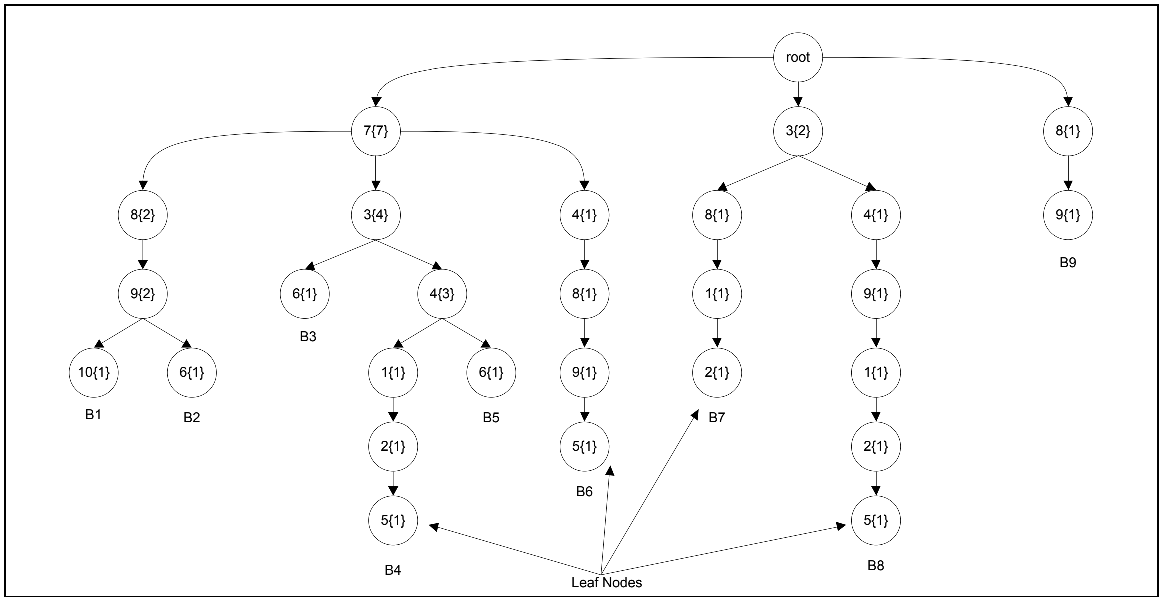

Figure 2. The FP-Tree for the database $\mathcal{D}$ shown in Table 1.

As seen in Figure 2, the database $\mathcal{D}$ contains 10 transactions and is condensed into the FP-Tree that contains nine leaf nodes. This occurs because the transaction t3 = {7, 3, 4} is supported within transaction t7 = {7, 3, 4, 6} and t8 = {7, 3, 4, 1, 2, 5}. One of the main advantages of using the FP-Tree is that the count of the items in nodes 3, 4, and 7 are just incremented in order to add transaction t3 to the FP-Tree, without the need to add new nodes. This advantage is beneficial when the database has a large number of transactions and there is an increased probability of having similar or repeated transactions.

For the purpose of this dissertation, the FP-Tree is generated for only the transactions that support sensitive and frequent itemsets, i.e., relevant transactions. Weights are assigned (as discussed in the Methodology section) to items and branches within the FP-Tree so that certain nodes are chosen to be modified during the sanitization process.

Barriers and Issues

Attallah, Bertino, Elmagarmid, Ibrahim and Verykios (1999) have shown that the frequent itemset hiding problem is an NP-Hard problem. Attallah et al. (1999) and others have proposed multiple methods to find a satisfactory solution for the frequent itemset hiding problem. Most methods employ a heuristic-based approach and depend on minimizing the support level of the sensitive itemsets by deleting items in transactions (Menon & Sarkar, 2007; Menon, Sarkar, & Mukherjee, 2005). None of the previously suggested heuristic-based methods took into account the absolute difference between the total number of items in the original database and the total number of items in the sanitized version of the database.

Chapter 2

Review of Literature

Introduction

Privacy preserving was the main area that was addressed in this study. This chapter will present a history of privacy preserving, its dimensions, and its classifications. The focus of the literature review will be on heuristic-based approaches since the suggested FP-Tree approach falls into that category and it is the most addressed topic in Association Rule Hiding studies (Verykios & Gkoulalas-Divanis, 2008). The suggested FP-Tree approach was considered in the context of Association Rule Hiding and its classification was discussed in detail. Several other conclusions were made that were taken into consideration in the methodology section.

Historical Review of Association Rule Hiding

Wang and Hong (2007) defined two broad classifications for privacy preserving in data mining: output privacy and input privacy. Output privacy is mainly concerned with minimally modifying the data in order to prevent any private or sensitive information from being disclosed as a result of using any mining technique. Examples of output privacy preserving methods include perturbation, blocking, aggregation or merging, swapping, sampling, and some alteration methods. Input privacy, on the other hand, is mainly concerned with manipulating the data so that mining results are not affected or are minimally affected. Examples of input privacy preserving methods include reconstruction-based techniques and cryptography-based techniques.

On the other hand, Verykios and Gkoulalas-Divanis (2008) defined another two broad categories for privacy preserving methods: data hiding and knowledge hiding. Data hiding removes confidential, private information from the data before its disclosure to third parties; the main concern is the data itself. Knowledge hiding is concerned with the information or knowledge within that data that may be discovered with data mining techniques.

D. E. O'Leary (1991) was the first to suggest that data mining approaches need to be considered in a way to avoid breaches in security and privacy after applying these methods to data (Verykios & Gkoulalas-Divanis, 2008). The term "Association Rule Hiding" was first used by Atallah et al. in their 1999 workshop paper (Verykios & Gkoulalas-Divanis, 2008). Atallah et al. (1999) started to apply some of the ideas that were suggested by Clifton and Marks (1996), e.g., the fuzzification of the database and the release of samples from the database rather than the entire database.

Knowledge hiding techniques, specifically Association Rule Hiding, were the main focus of this research. Following the classification approach of Verykios and Gkoulalas-Divanis (2008) a taxonomy of the frequent itemset and Association Rule Hiding algorithms will be presented. The next sections will discuss the dimensions of

Association Rule Hiding, offer a classification of Association Rule Hiding algorithms and provide a description of each classification.

Dimensions of Association Rule Hiding Algorithms

Association Rule Hiding algorithms can be considered with several dimensions in mind—they can achieve hiding by either reducing the support level (support-based) or reducing the confidence level (confidence-based) of the association rule. These algorithms can also be either heuristic-based or exact. Although heuristic approaches are efficient, fast, and can handle a large database size, they do not guarantee an optimal solution.

Association Rule Hiding algorithms are driven by the primary goal to, generally, hide sensitive itemsets and by a secondary goal to minimize any side effects that may result from the hiding process, e.g., loss of nonsensitive itemsets, reduction in total number of items, and the production of new association rules. Unlike a heuristic-based approach, an exact approach considers all the goals of Association Rule Hiding. Admittedly, using such an approach may increase computational cost, potentially to the point where computational resources are exhausted because an optimal solution does not exist. In this case, stipulations set by the goals may be relaxed to allow the algortihm to reach a solution (Verykios & Gkoulalas-Divanis, 2008; Modi, Rao, & Patel, 2010).

Association Rule Hiding algortihms may use distortion or blocking techniques in rule hiding. Distortion is the process of replacing 1s with 0s and 0s with 1s, while blocking is the process of replacing the original values in the original database with question marks (Verykios & Gkoulalas-Divanis, 2008; Modi, Rao, & Patel, 2010). Algorithms can also hide one association rule or multiple association rules during a single iteration of the algorithm—these are known as single-rule or multiple-rule schemes, respectively (Verykios & Gkoulalas-Divanis, 2008).

Verykios and Gkoulalas-Divanis (2008) notes that some algorithms may require a pre-mining step before the hiding process, while others do not. Pre-mining may introduce some breaches of privacy since some frequent itemsets may be generated before the hiding process takes place, allowing the potential for sensitive itemsets to be revealed to the individual who is executing the hiding process (Yildiz & Ergenc, 2011; Wang & Hong, 2007). Pre-mining also introduces inefficiencies and delays in the hiding process since most pre-mining processes require the generation of candidate sets which results in multiple scans of the database and an increase in execution time (Yildiz & Ergenc, 2011; Wang & Hong, 2007).

Classification of Association Rule Hiding Algorithms

Verykios and Gkoulalas-Divanis (2008) classify the Association Rule Hiding Algorithms into heuristic-based approaches, border-based approaches, and exact approaches. Modi et al. (2010) contribute two additional approaches—reconstruction-based approaches and cryptography-based approaches. This section covers several of these classes of Association Rule Hiding Algorithms.

Because the FP-Tree approach that was examined by this study is classified as a heuristic-based approach, the focus will be primarily on this type of approach. Some exact approaches will also be examined, as they will serve as a basis for comparison.

Heuristic Based Approaches

Heuristic-based approaches, as mentioned earlier, are efficient, fast, and scalable algorithms that sanitize the database (Verykios & Gkoulalas-Divanis, 2008). As a result, they have been the focus of multiple research efforts despite their susceptibility to side effects that may affect the quality of the output database (Verykios & Gkoulalas-Divanis, 2008). Heuristic approaches are usually either support-based or confidence-based, but may also be a hybrid of both support-based and confidence-based approaches. Reducing the support level or confidence level may be achieved by distortion techniques or blocking techniques; however, most of the heuristic-based approaches in the literature employ distortion techniques (Verykios & Gkoulalas-Divanis, 2008; Modi, Rao, & Patel, 2010).

Support-Based and Confidence-Based Distortion Schemes

Atallah et al. (1999) were the first to attempt a heuristic-based approach in this domain. Guided by several assumptions, sensitive itemsets were hidden by reducing their support level or their confidence level (Modi, Rao, & Patel, 2010). A weakness of this approach is that it did not take into account the amount of loss for large itemsets as long as they remained frequent in the sanitized database (Verykios & Gkoulalas-Divanis, 2008). Despite this shortcoming, Atallah et al. (1999) provided a proof that the Association Rule

Hiding Problem, when attempting to find an optimal solution, is an NP-Hard problem (Verykios & Gkoulalas-Divanis, 2008; Modi, Rao, & Patel, 2010).

Dasseni, Verykios, Elmagarmid, and Bertino (2001) suggested three heuristics that considered the reduction of the support level or the confidence level of sensitive itemsets. The goal of these three heuristics was to hide the sensitive itemsets while minimizing the loss of support for the nonsensitive and frequent itemsets (Verykios & Gkoulalas-Divanis, 2008). The suggested algorithms failed to minimize the undesired side effects such as losing a nonsensitive and frequent itemset or creating a new frequnet itemset (Verykios & Gkoulalas-Divanis, 2008). Verykios, Elmagarmid, Bertino, Saygin and Dasseni (2004) extended the work of Dasenni et al. (2001) by evaluating the performance on input databases of various sizes and using various sizes of sensitive itemsets. Furthermore, they improved upon the process of maintaining the knowledge within the sanitized database by implementing a process that chooses the item with the highest support level and removed that item from the transaction with lowest number of items.

Oliveira and Zaïane (2002) suggested the first multiple rule hiding heuristic. They came up with three algorithms that attempts to hide the sensitive itemsets while concurrently reducing the effect on nonsensitive itemsets (Verykios & Gkoulalas-Divanis, 2008). The suggested algorithms are very efficient, only requiring two scans of the database. However, a more efficient algorithm that requires only one scan of the database was introduced by Oliveira and Zaïane (2003) (Verykios & Gkoulalas-Divanis, 2008).

Amiri (2007) introduced yet another set of multiple rule hiding heuristics that outperformed the algorithm that was suggested by Oliveira and Zaïane (2003), though at a higher computational cost. The goal of these suggested methods was to reduce the distortion of the sanitized database while removing sensitive itemsets. The algorithms presented by Amiri (2007) proposed the removal of the entire transaction with the least number of nonsensitive itemsets and the most number of sensitive itemsets in his Aggregate approach. However, in his Dissagregate approach, Amiri (2007) suggested removing the item from the transaction that will affect the most sensitive itemsets and the least nonsensitive itemsets.

Wu, Chiang and Chen (2007) modified the work of Dasenni et al. (2001) and introduced special constraints to reduce the possible side effects. In order to ensure that the major constraint (i.e, sensitive itemsets should be hidden) is satisfied, the alogrithm might relax some of the other less crucial constraints. This is done without the consent of the user and is a major drawback of this algorithm (Verykios & Gkoulalas-Divanis, 2008).

Pontikakis, Tsitsonis, and Verykios (2004) suggested two distortion-based heuritics. They use an effective data structure to choose the appropriate transaction to be sanitized. However, the suggested algorithm introduces undesirable effects such as hiding nonsensitive and frequnet itemsets or creating new rules (Verykios & Gkoulalas-Divanis, 2008).

Wang and Hong (2007) suggested the use of a Pattern-Inversion Tree (PI-Tree) that stores details about the database, which can be used after scanning the database one time. In every node, the PI-Tree stores the item, its frequency, and the transaction IDs that support the item. By iterating through the various sensitive rules, the algorithm attempts to identify the number of transactions that are required to lower the confidence-level of the rule below detection level as well as the number of transactions that are required to lower the support-level of that rule below detection level. The algorithm then proceeds by reducing either the confidence-level or the supportlevel by choosing the method that will modify the least number of transactions. Using a tree structure allows the algorithm to iterate quickly through the structure and avoids any pre-mining (Wang & Hong, 2007).

Modi, Rao, and Patel (2010) proposed a heuristic-based approach that achieves hiding by clustering the sensitive association rules and then decreasing the support of the Right-Hand Side (R.H.S) of rule clusters. The algorithm requires pre-mining to generate the candidate sets, including the sensitive rules. By reducing the support of the R.H.S item in the rule $X \Rightarrow Y$, the algorithm reduces the confidence-level faster than reducing the supportlevel of X. Experimental results were reported by Modi et al. (2010), exhibiting better performance compared to other heuristic-based approaches.

Yildiz and Ergenc (2011) proposed four different types of heuristics that do not require pre-mining. Based on the analysis of the experimental results of the four approaches, the study concluded that the performance of the sanitization alogrithm depends on the pattern that is distorted. For example, the study showed that selecting the shortest pattern with the maximum number of frequent itemsets showed the best overall performance. However, Yildiz and Ergenc (2011) did not compare their results to other results from any existing heuristic-based algorithm in the literature, although this was suggested as a next step.

Support-Based and Confidence-Based Blocking Schemes

Saygin, Verykios, and Clifton (2001) were the first to use “unknowns”, i.e., inserting question marks rather than transposing 0s and 1s with one another. Saygin et al. (2001) proposed three heuristics, two of which depend on reducing the confidence-level of the association rule. The third heuristic depends on reducing the support level of the association rule. The major contribution of Saygin et al. (2001) was the introduction of the safety factor, which provides a mechanism to measure how safe the data is from being discovered by an adversary (Saygin, Verykios, & Clifton, 2001; Verykios & Gkoulalas-Divanis, 2008).

Wang and Jafari (2005) proposed two heuristics that deal with hiding the predictive association rules—rules that contain the sensitive items on their Left-Hand Side (LHS). Both methods are based on reducing the confidence level. One of the major contributions to blocking-based schemes is the work of Pontikakis, Theodoridis, Tsitsonis, Chang, and Verykios (2004). Pontikakis et al. proposed the concept of the ghost rules, which make it more difficult for an adversary to identify any sensitive rules (Pontikakis, Theodoridis, Tsitsonis, Chang, & Verykios, 2004; Verykios & Gkoulalas-Divanis, 2008).

Border-Based Approaches

Sun and Yu (2005) were the first to suggest the use of border revision in association rule hiding (Verykios & Gkoulalas-Divanis, 2008; Modi, Rao, & Patel, 2010). They proposed a heuristic approach that considers the border of the nonsensitive and frequent itemsets. The borders are used to track the effects that result from modifying transactions during the hiding process in an attempt to preserve the quality of the sanitized database. Moustakides and Verykios (2006) extended the work of Sun and Yu (2005) by implementing a revised and more efficient border-based algorithm (Moustakides & Verykios, 2006; Verykios & Gkoulalas-Divanis, 2008).

Exact Approaches

As discussed previously, an exact approach will find the optimal solution, if it exists, at the expense of extra computational cost. Two prominent studies were discussed earlier in the Relevance and Significance section of Chapter 1—Menon, Sarkar, and Mukherjee (2005) and Menon and Sarkar (2007). One main advantage of both of these studies is that they introduce several steps in order to reduce the time taken to solve the Constraint Satisfaction Problem (CSP) (Verykios & Gkoulalas-Divanis, 2008).

Gkoulalas-Divanis and Verykios (2006) proposed another exact alogirthm that utilizes the positive and revised negative borders to identify candidate itemsets for sanitization. Although the algorithm imposes a CSP that is larger than the CSP in Menon et al. (2005), the authors suggested that the algorithm achieves a satisfactory level of efficiency.

Reconstruction-Based Approaches

A newer method that is not sufficiently addressed in the literature is the reconstruction-based method. The method that can be classified as knowledge sanitization utilizes the concept of inverse frequent set mining (Chen, Orlowska, & Li, 2004; Guo, 2007). Guo (2007) suggested a new framework that utilizes an itemset lattice. The itemset lattice contains information about the original database. The itemset lattice is an all-partial-ordered subset of items that are generated from transactions. The itemset lattice is sanitized and a new database with limited side effects is generated from the lattice itself (Guo, 2007).

Cryptography-Based Approaches

The cryptography-based approach is a form of input privacy in which data is encrypted within the original database. This approach addresses privacy issues for organizations that store databases at multiple sites (Modi, Rao, & Patel, 2010). Vaidya and Clifton (2002) proposed a two-party algorithm to discover sensitive itemsets without allowing the two sites to discover the individual transaction values. Kantarcioglu and Clifton (2004) addressed mining association rules over horizontal partitioned data securely.

Other Approaches

Other approaches were suggested in the literature to handle Association Rule Hiding. Zhu and Du (2010) showed that an attacker can discover the sensitive association rules given that there is not enough blindage in the hiding process. They came up with a novel K-anonymous metric to ensure that a sensitive association rule cannot be distingushed from at least $K - 1$ other association rules in the specific region. Gkoulalas-Divanis and Verykios (2009), on the other hand, presented the framework of database extension. They minimally introduced an extension to the original database in order to hide sensitive rules and bring the support level of sensitive itemsets below the detection-level.

Summary

The history of privacy preserving was discussed with a focus on Association Rule Hiding algorithms. The Dimensions of Association Rule Hiding algorithms were also discussed. Finally, a Classification of Association Rule Hiding algorithms with a focus on heuristic-based approaches was presented.

The FP-Tree algorithm can be classified as an output privacy method that is heuristic-based, support-based, distortion-based, and uses multiple rules. The FP-Tree approach minimally modifies the data in order to prevent any sensitive itemset disclosure after the mining operation. It also maximizes the utility of the database; hence, it is an output privacy approach (Wang & Hong, 2007). The FP-Tree approach uses an FP-Tree structure to store the transactions in a condensed form. It also uses a heuristic that will iterate through the structure to remove items; hence it is a heuristic-based approach. The FP-Tree approach hides sensitive itemsets by reducing the support level of the sensitive itemsets; hence, it is a support-based approach. The FP-Tree approach reduces the support of sensitive itemsets by removing items from transactions; hence, it is a distortion-based approach. Finally, the FP-Tree approach removes one item in every iteration. This may reduce the support of one or more sensitive itemsets; hence, it is a multiple-rule method.

The FP-Tree has started to gain popularity in Association Rule Hiding. Two studies by Wang and Hong (2007) and Guo (2007) have used the FP-Tree to address the Association Rule Hiding problem. This study used the FP-Tree in an implementation similar to Wang and Hong (2007), but with major differences. The FP-Tree used in this study is a support-based approach, while the one used by Wang and Hong (2007) was both support-based and confidence-based. The PI-Tree in Wang and Hong (2007) differs from the suggested FP-Tree; this will be elaborated upon in the Methodology section.

Previous studies have suggested that the quality of the sanitization process may be dependent on several factors. For example, the pattern that is selected to be distorted can affect the quality of the sanitization (Yildiz & Ergenc, 2011). This and several other factors were considered when evaluating the method implemented by the current study.

Most of the studies that were mentioned in the literature review did not consider the total number of items that are removed from transactions during the sanitization process. This research and the suggested FP-Tree approach was an attempt to tackle this unexplored area in Association Rule Hiding.

Chapter 3

Methodology

Introduction

This chapter discusses the methodological approach of this dissertation. The proposed method, which attempts to find satisfactory solutions to the frequent itemset hiding problem, will be discussed in detail. Supporting evidence for the use of the proposed method will also be provided. Details of the performance evaluation of the method along with the code implementation that will be used will be discussed.

Research Methods Employed

The purpose of this study, as previously stated, is to find satisfactory solutions for a version of the frequent itemset hiding problem, in which the total number of items removed from all transactions is minimized. The proposed method is similar in concept to the method that Menon and Sarkar (2007) presented, by which certain items are removed from transactions. The proposed method in this research utilized the FP-Tree.

The FP-Tree in the proposed method is generated for relevant transactions. This FP-Tree contains multiple nodes that extend from the root node to the leaf nodes. The FP-Tree contains multiple branches; a branch is defined as the path from a leaf node to the root node including all nodes in that path. A branch can share more than one node with other branches. Each branch represents one transaction or multiple transactions that are nearly or completely identical. Each node contains an item with its count—its support level in the transactions that are represented by the branch that the node lies in. The same item can be contained by several other nodes within the FP-Tree, i.e., in other branches.

The FP-Tree represents the relevant transactions. Sanitizing the FP-Tree will sanitize the relevant transactions. In every iteration of the sanitization process, a node will be chosen to be modified by reducing the count of the item from the chosen node by 1. In order to identify the node that needs to be modified, weight functions for branches and weight functions for items were used. Based on the outcome of the weight functions, the branch is identified and the item is also identified. The node that contains the chosen item within the chosen branch will be modified. Weight functions are chosen to minimize the total number of items that are removed during the sanitization process. Weight functions will be discussed in the upcoming sections (Weight Functions for Items and Weight Functions for Branches).

For the purpose of this dissertation, list L' is defined as the list of items that are included by all sensitive frequent itemsets. The method will iterate over the items in list L' and the branches in the FP-Tree, after which weights will be assigned to each item and branch in order to determine which node should be modified. The support level in the FP-Tree for the sensitive and frequent itemsets during this process will gradually decrease. When the support level for every sensitive and frequent itemset in $\mathcal{F}^{\mathrm{C}}$ becomes lower than σ_{min}, the sanitization process is complete. The resultant FP-Tree will be a sanitized FP-Tree.

If the chosen node is a node shared between more than one branch, the chosen branch will split as a result of the modification process. An example is illustrated in Figure 3, where the node that contains item 4 in branch B4 is to be modified. This node is shared between branches B4 and B5. Notice that the node containing item 4 after the split still exists in branch B5, but its count is reduced from 3 to 2. In addition to this, branch B4 does not contain any node that contains item 4 since the original support level for item 4 in branch B4 was 1.

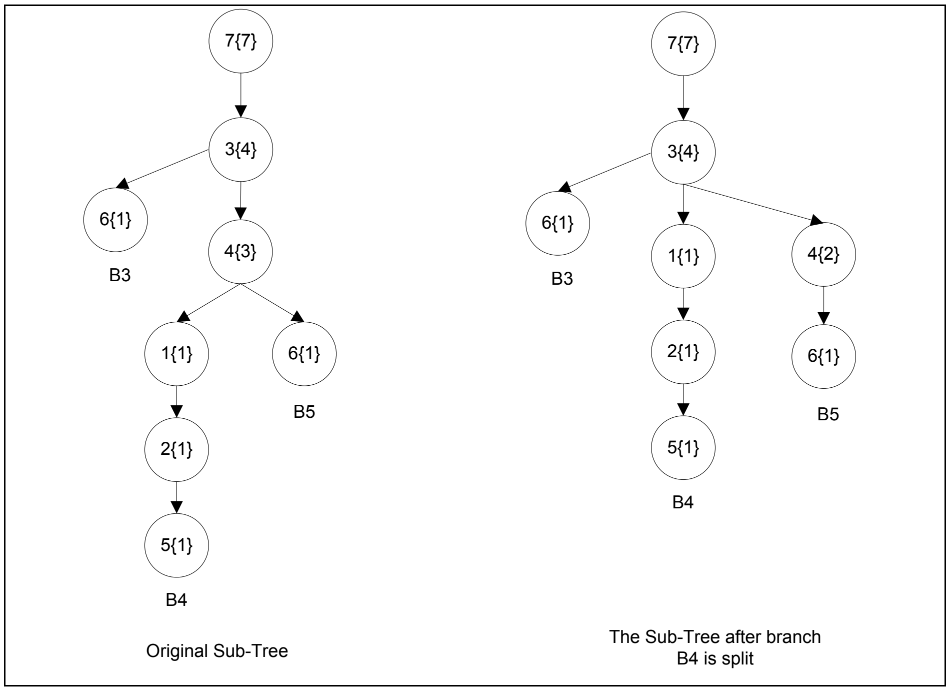

Figure 3. Branch B4 splits when the node that contains item 4 is modified. The chosen node is a shared node.

Extracting the transactions from the sanitized FP-Tree is the next step. Each transaction is constructed by iterating through the nodes in the FP-Tree at different levels.

Each node is visited only once, and all items are extracted to build the transactions. The resultant database is the sanitized version of the original database that contained relevant transactions, i.e., database $\mathcal{D}''$.

The pseudocode for the proposed method is shown in Figure 4 below. The algorithm uses the database $\mathcal{D}$, the minimum support level σ_{min}, and the list of sensitive and frequent itemsets $\mathcal{F}^{\mathrm{C}}$, where $\mathcal{F}^{\mathrm{C}} = \mathcal{F}^{R}$ initially. The support level for each itemset in $\mathcal{F}^{C}$ will be extracted during the generation of the FP-Tree. Step (a) of the algorithm generates the database $\mathcal{D}'$ that contains only relevant transactions. An FP-Tree is also generated in this step using the database $\mathcal{D}'$ by calling the Generate_FP_Tree($\mathcal{D}'$) procedure that was described earlier (Figure 1).

Yildiz and Ergenc (2011) mentioned that selecting the shortest pattern and maximum of frequent itemsets is the approach that yields the best results. Depending on the criteria used to identify which node will be modified, results will vary. Whether the item or the branch is chosen first as a basis of selecting a node will also influence results. In order to find the optimal solution, a variety of methods shall be used.

The first method chooses the branch first and then the item in order to identify the node that will be modified based on the outcome of certain weight functions. The second method chooses the item first and then the branch in order to identify the node that will be modified based on the outcome of certain weight functions.

In step (b), if the branch is to be chosen first, then the branches are evaluated to determine the branch with the highest weight $\mathrm{B_H}$. The items that are contained in L' are then evaluated for branch B_H by generating the weight I_w for each item in L'. The item with the highest weight I_H and its corresponding node is chosen. On the other hand, if it is the item that is chosen first, then the relevant items that are contained in L' are evaluated by generating the appropriate weight I_w for each item. The item with the highest weight I_H is chosen and the branches are then evaluated to determine the branch with the highest weight B_H. The node that contains I_H within B_H will be modified by reducing its count by 1. If the node is shared between multiple branches, then the branch B_H will split.

Each time a node is modified, the algorithm modifies the support level of the itemsets in $\mathcal{F}^C$ that are affected. Each itemset in $\mathcal{F}^C$ is evaluated to determine if it still meets the criteria to be classified as a frequent itemset. If the itemset is no longer a frequent itemset, it is removed from $\mathcal{F}^C$. This process continues until the list $\mathcal{F}^C$ is empty and the resultant FP-Tree is sanitized. The items in L' are also evaluated to determine if they are still part of sensitive itemsets in $\mathcal{F}^C$. If an item is no longer included in any sensitive itemset in $\mathcal{F}^C$, that item will be removed from L'. In the next section, the weight functions that will be used to generate the weights for items and branches will be considered.

```
Algorithm Min_Items_Removed($\mathcal{D}, \sigma_{min}, \mathcal{F}^C$)

Input: $\mathcal{D}, \sigma_{min}$ and the list of sensitive and frequent itemsets $\mathcal{F}^C$

a)  Generate the FP-Tree for the relevant transactions only

    Scan the database $\mathcal{D}$ and identify transactions that contain sensitive itemsets.

    For each transaction T $\in \mathcal{D}$
        If T contains any sensitive itemsets in $\mathcal{F}^C$
                Add T to $\mathcal{D}'$
        End if
    End For

    Call Generate_FP_Tree($\mathcal{D}'$) → this will generate the FP-Tree, list L, and list L′ for all relevant transactions

b)  Start removing items from the FP-Tree. Decision will depend first on whether to choose the branch first or the item first.

    While $\mathcal{F}^C$ is not empty
        Remove any items in L′ that has a support level below $\sigma_{min}$
        If branch to be chosen first:
                Generate $B_w$ for each branch in the FP-Tree
                Choose the Branch that has the highest weight, i.e. $B_H$, if a tie exists between two branches or more
                choose the branch with the less number of items to minimize information loss
                Generate $I_w$ for each item $i \in$ L′ where $i$ lies in branch $B_H$.
                Choose the item with the highest weight in the chosen branch, i.e. $I_H$, if a tie exists choose the item
                randomly.
        If item to be chosen first:
                Generate $I_w$ for each item $i \in$ L′
                Choose the item with the highest weight, i.e. $I_H$, if a tie exists choose the item randomly.
                Generate $B_w$ for each branch in the FP-Tree that contains $I_H$
                Choose the Branch that has the highest weight, i.e. $B_H$, if a tie exists between two branches choose the
                branch with the less number of items to minimize information loss

        Reduce the count of the node that contains $I_H$ from $B_H$ by 1.
        Reduce the support level for $I_H$ in L′ by 1
        Reduce the support level for each itemset in $\mathcal{F}^C$ that was affected by 1
        Split Tree if needed for the branch that was modified
        For each itemset j in $\mathcal{F}^C$
                if support level of itemset j in $\mathcal{F}^C$ is less than $\sigma_{\min}$
                        remove itemset from $\mathcal{F}^C$
                        remove items that are unique to j from L′
                End if
        End For

    End While
```

Figure 4. Algorithm Min_Items_Removed($\mathcal{D}, \sigma_{\min}, \mathcal{F}^C$).

Weight Functions for Items

The weight function that is used to calculate the weight for each item was one of the main challenges in this dissertation. The weight function calculated the weight of an item in order to minimize the total number of items that are removed in the sanitized version of the database. For example, if item i is supported in most itemsets that belong to $\mathcal{F}^C$ and item k is supported only in one itemset in $\mathcal{F}^C$, then the assumption is that the weight for item i should be higher than the weight for item k. Consider when a node that contains item i is modified in the tree. This will increase the probability that more than one itemset in $\mathcal{F}^C$ will have its support level reduced. This will reduce the total number of items that are removed from the original database.

For this study, two weight functions for items were considered; the first weight function is shown in (6):

$$Weight\_item\_1(i) = \sum_{j \in \mathcal{F}^C} a_{ji} \times N_j \qquad (6)$$

Where i is the item in L' that is being evaluated, a_{ji} is set to 1 if the sensitive itemset j contains item i; otherwise, it is set to 0, where j belongs to the current set of sensitive itemsets $\mathcal{F}^C$. If the branch is to be chosen first, then $\mathrm{B_H}$ must contain item i. N_j is the size of sensitive itemset j in terms of number of items. Essentially the weight function in (6) is the sum of the size of sensitive frequent itemsets that contain this item. The pseudocode to generate the weights for all items using *Weight_item_*1(i) and to subsequently choose $\mathrm{I_H}$ is shown in Figure 5.

```
Algorithm _Find_$I_H$_Weight_item_1($L'$, $\mathcal{F}^C$, $B_H$ (if branch is chosen first))

Input:  $L'$, $\mathcal{F}^C$, $B_H$ (if branch is chosen first)
Choose the first item in $L'$ as $I_H$

a)   Preprocess each sensitive itemset in $\mathcal{F}^C$ to find the number of items it contains (Done only once):

        For each itemset $j$ in $\mathcal{F}^C$
                $j$.weight = number of items that are included in $j$
        End For

b)   Preprocess all items in $L'$ to calculate the weight of each item (Done only once):

        For each item $i$ in $L'$, $i$.weight =0
        For each itemset $j$ in $\mathcal{F}^C$
                For each item $i$ in $L'$
                        If $j$ contains $i$ then
                                $i$.weight = $i$.weight + $j$.weight
                        End If
                End For
        End For

c)   Sort items in $L'$ according to weight and store the items and their weights in this list (Done only once).

d)   If item is to be chosen first then choose the item with the highest weight in $L'$ to be $I_H$.

e)   If branch is to be chosen first then
        For each item $i$ in $L'$ starting from the item with the highest weight
                If $B_H$ supports $i$
                        Choose $i$ to be $I_H$ and end the selection process.
                End If
        End For

f)   After processing the node that contains item $I_H$ in the FP-Tree if any itemset $j$ in $\mathcal{F}^C$ becomes non frequent (and
     was removed from $\mathcal{F}^C$ ) then:
        For each item $c$ in $j$
                If $c \in L'$
                        $c$.weight = $c$.weight – $j$.weight
                End If
        End For
        Sort items in $L'$ according to weight. If any item's weight = 0 then remove that item from $L'$
```

Figure 5. Finding I_H using *Weight_item*_1(i).

The second weight function for items that is under consideration is shown in (7):

$$Weight\_item\_2(i) = S_{\mathcal{D}'}(i) \sum_{j \in \mathcal{F}^C} a_{ji} \tag{7}$$

Where i is the item in L' that is being evaluated, a_{ji} is set to 1 if the sensitive itemset j contains item i; otherwise, it is set to 0, where j belongs to the current set of sensitive itemsets $\mathcal{F}^C$. If the branch is to be chosen first, then B_H must contain item i. $S_{\mathcal{D}'}(i)$ is the support level of item i in the relevant database $\mathcal{D}'$. Essentially the weight function in (7) is the number of sensitive frequent itemsets that supports item i multiplied by the support of that item in the relevant database $\mathcal{D}'$. The pseudocode to generate the weights for all items using *Weight_item_*2(i) and to subsequently choose I_H is shown in Figure 6.

```
Algorithm _Find_I_H_Weight_item_2(L', $\mathcal{F}^C$, $B_H$ (if branch is chosen first))

Input:  L', $\mathcal{F}^C$, $B_H$ (if branch is chosen first)

Choose the first item in L' as $I_H$

a)      For each item $i$ in L' (and $i$ in $B_H$ if branch is chosen first)
        $i$.weight = 0
                For each itemset $j$ in $\mathcal{F}^C$
                        If $i \in j$
                                $i$.weight = $i$.weight + 1
                        End If
                End For
        $i$.weight = $i$.weight × $S_{\mathcal{D}'}(i)$
        If $i$.weight > $I_H$.weight
                $I_H = i$
        End If
        End For
        $S_{\mathcal{D}'}(I_H) = S_{\mathcal{D}'}(I_H) - 1$
```

Figure 6. Finding I_H using *Weight_item_*2(i).

Weight Functions for Branches

The weight function that is used to calculate the weight of each branch was another major challenge in this dissertation. The weight function should select the branch that will efficiently reduce the support of sensitive itemsets that belong to $\mathcal{F}^C$. Eventually, this will result in fewer total items being removed. Four weight functions for weighing branches were considered; the first weight function, which requires the branch to be chosen first, is shown in (8).

$$Weight\_branch\_1(b) = \sum_{j \in \mathcal{F}^C} c_{bj} \tag{8}$$

Where b is the branch that is being evaluated, c_{bj} is assigned a value of 1 if branch b contains transactions that support itemset j; otherwise, it is assigned a value of 0, where j belongs to $\mathcal{F}^C$. This algorithm is only used when a branch is chosen first. Essentially, the weight function in (8) is the number of sensitive itemsets that are contained within branch b. The pseudocode to generate the weights for all the branches using $Weight\_branch\_1(b)$ and subsequently selecting B_H is shown in Figure 7.

```
Algorithm _Find_B_H_Weight_branch_1(FP – Tree, $\mathcal{F}^C$,)

Input:  FP-Tree, $\mathcal{F}^C$
Choose the first branch in FP-Tree as $B_H$

a)  Preprocess all branches and calculate their weights, for each sensitive itemset $j$ store all branches that contain that
itemset in a new list defined as Branch_List (Done only once):
        For each branch $b$ in FP-Tree
        $b$.weight = 0
                For each itemset $j$ in $\mathcal{F}^C$
                        If ($j \in b$)
                                $b$.weight = $b$.weight + 1
                                add branch $b$ to Branch_List in association with $j$
                        End If
                End For
                If $b$.weight > $B_H$.weight
                        $B_H = b$
                End If
        End For

b)    After each iteration of the overall algorithm if any sensitive itemset $j$ becomes non frequent then:
      For each branch $b$ in Branch_List that contains itemset $j$
          $b$.weight = $b$.weight -1
      End For
c)    If there is a split within a branch reevaluate the branches that were affected and adjust Branch_List.
      Sort branches according to weight. Choose branch with highest weight for next iteration.
```

Figure 7. Finding B_H using $Weight\_branch\_1(b)$.

Another weight function that requires the item to be chosen first is shown in (9).

$$Weight\_branch\_2(b) = \sum_{j \in \mathcal{F}^C} c_{bj} \times a_{jI_H} \tag{9}$$

Where b is the branch that is being evaluated, c_{bj} is assigned a value of 1 if branch b contains transactions that support itemset j; otherwise, it is assigned a value of 0, where j belongs to the current set of sensitive itemsets $\mathcal{F}^C$. Branch b should contain item I_H. a_{jI_H} is assigned a value of 1 if itemset j contains item I_H and a value of 0 otherwise. Essentially, the weight function in (9) is the total number of sensitive itemsets in branch b that contain item I_H. The pseudocode that will generate the weights for all branches using *Weight_branch_2(b)* and then choose B_H is displayed in Figure 8.

```
Algorithm _Find_$B_H$_$Weight\_branch\_2$(FP – Tree, $\mathcal{F}^C$, $I_H$)

Input:  FP-Tree, $\mathcal{F}^C$, $I_H$
Choose the first branch in FP-Tree as $B_H$

a)      For each branch $b$ in FP-Tree
        $b$.weight = 0
                For each itemset $j$ in $\mathcal{F}^C$
                        If ($j \in b$ AND $I_H \in j$)
                                $b$.weight = $b$.weight + 1
                        End If
                End For
        If $b$.weight > $B_H$.weight
                $B_H = b$
        End For
        End If
```

Figure 8. Finding B_H using *Weight_branch_2(b)*.

Another weight function, which takes into account what Yildiz and Ergenc (2011) mentioned regarding the selection of the shortest pattern and maximum frequent itemsets, was evaluated. This weight function considers the length of the branch as well as the total number of itemsets in branch b that belong to $\mathcal{F}^C$ as shown in (10). This algorithm requires the branch to be chosen first.

$$Weight\_branch\_3(b) = \frac{1}{N_b} \sum_{j \in \mathcal{F}^C} c_{bj} \tag{10}$$

Where b is the branch that is being evaluated, c_{bj} is assigned a value of 1 if branch b contains transactions that support itemset j; otherwise, it is assigned a value of 0, where j belongs to the current set of sensitive itemsets $\mathcal{F}^C$. N_b is the number of nodes in branch b. Essentially, the weight function in (10) is the ratio between the number of sensitive itemsets that are contained by a branch and the number of nodes in the same branch. The shorter the branch and the more sensitive itemsets it contains, the higher the weight. The pseudocode that will generate the weights for all branches using *Weight_branch_3(b)* and subsequently select B_H is shown in Figure 9.

```
Algorithm _Find_B_H_Weight_branch_3(FP – Tree, $\mathcal{F}^C$)

Input:  FP-Tree, $\mathcal{F}^C$
Choose the first branch in FP-Tree as $B_H$

a)      For each branch $b$ in FP-Tree
        $b$.weight = 0
                For each itemset $j$ in $\mathcal{F}^C$
                        If ($j \in b$)
                                $b$.weight = $b$.weight + 1
                        End If
                End For
        $b$.weight = $b$.weight/$N_b$
        If $b$.weight > $B_H$.weight
                $B_H = b$
        End For
        End If
```

Figure 9. Finding B_H using *Weight_branch_*3(b).

Another weight function that requires the item to be chosen first and considers Yildiz and Ergenc (2011) regarding the selection of the shortest pattern and maximum frequent itemsets is shown in (11).

$$Weight\_branch\_4(b) = \frac{1}{N_b} \sum_{j \in \mathcal{F}^C} c_{bj} \times a_{jI_H} \tag{11}$$

Where b is the branch that is being evaluated, c_{bj} is assigned a value of 1 if branch b contains transactions that support itemset j; otherwise, it is assigned a value of 0, where j belongs to the current set of sensitive itemsets $\mathcal{F}^C$. Branch b should contain item I_H. a_{jI_H} is assigned a value of 1 if itemset j contains item I_H and a value of 0 otherwise. N_b is the number of nodes in branch b. Essentially, the weight function in (11) is the ratio between the number of sensitive itemsets that support item I_H and are contained by the branch and the number of nodes in the same branch. The shorter the branch and the more sensitive itemsets that support I_H, the higher the weight it is assigned. The pseudocode that will generate the weights for all branches using *Weight_branch_*4*(b*) and subsequently select B_H is shown in Figure 10.

```
Algorithm _Find_B_H_Weight_branch_3(FP – Tree, $\mathcal{F}^C$, $I_H$)

Input:  FP-Tree, $\mathcal{F}^C$, $I_H$
Choose the first branch in FP-Tree as $B_H$

a)          For each branch $b$ in FP-Tree
            $b$.weight = 0
                        For each itemset $j$ in $\mathcal{F}^C$
                                    If ($j \in b$ AND $I_H \in j$)
                                                $b$.weight = $b$.weight + 1
                                    End If
                        End For
            $b$.weight = $b$.weight/$N_b$
            If $b$.weight > $B_H$.weight
                        $B_H = b$
            End For
            End If
```

Figure 10. Finding B_H using *Weight_branch_*4(*b*).

Example

Using the same database that was shown in Table 1, one may demonstrate how the FP-Tree is used to find a satisfactory solution to the frequent itemset hiding problem. Recall that only relevant transactions that include sensitive itemsets will be used. The sensitive itemsets that will be used for demonstration in this example are shown in Table 5. The relevant transactions, i.e., database $\mathcal{D}'$, are shown in Table 6.

Table 5. $\mathcal{F}^C$, Sensitive and Frequent Itemsets That the Owner Would Like to Conceal

Sensitive Frequent Itemset					**Support level in $\mathcal{D}'$**
f1	1	2	3	4	2
f2	3	5			2
f3	4	5	7		2
f4	5	9			2
f5	8	9			4

Table 6. Transactions That Support Sensitive Itemsets or $\mathcal{D}'$

Transaction ID	**Items**					
t2	6	7	8	9		
t4	4	5	7	8	9	
t6	1	2	3	4	5	9
t8	1	2	3	4	5	7
t9	8	9				
t10	7	8	9	10		

The database that is generated in Table 6 above represents database $\mathcal{D}'$ that is generated by Algorithm Min_Items_Removed($\mathcal{D}, \sigma_{min}, \mathcal{F}^{C}$) that was previously described (Figure 4). Using this database, the FP-Tree will be generated as described in

Algorithm Generate_FP_Tree($\mathcal{D}'$) (Figure 1). The resulting FP-Tree is shown in Figure 11.

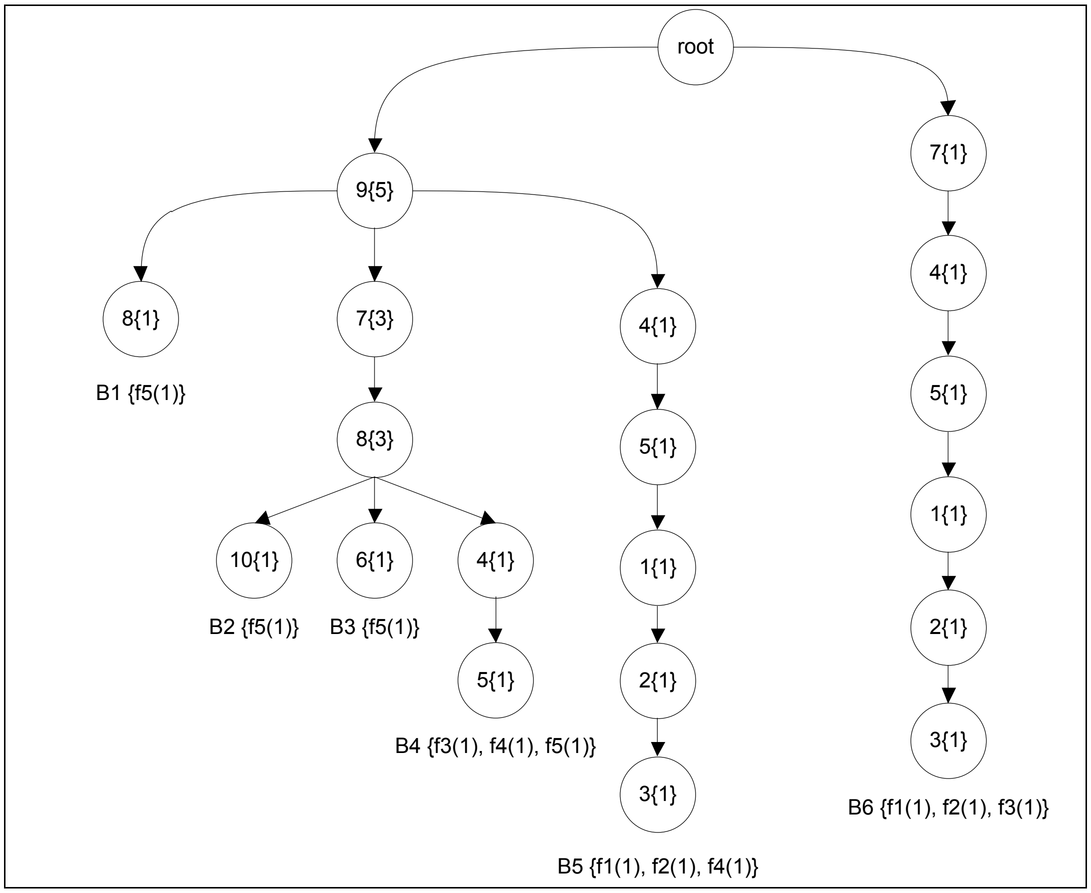

Figure 11. The FP-Tree generated for $\mathcal{D}'$ database.

Notice in Figure 11 that each branch is numbered (B1 to B6) and that each branch is marked, between brackets, with the sensitive and frequent itemsets that it contains. For example, Branch B3 supports itemset f5, and it includes only one instance of that itemset. Also, as a result of generating the FP-Tree, the list L′—items unique to sensitive frequent itemsets, shown in Table 7—is generated. In this example, the item will be chosen first in each iteration. Weights for items in L′ will be generated first in order to choose I$_H$. Following this, weights for each branch will be generated for the selection of B$_H$. *Weight_item*_1(*i*) and *Weight_branch*_2(*b*) will be used as a demonstration.

Table 7. List L′

Item	***Weight_item*_1(*i*) – initially empty**
9	
7	
8	
4	
5	
1	
2	
3	

Using the items listed above (Table 7), the weight for each item *i* can be generated using *Weight_item*_1(*i*). Evaluating item 9, it is found that:

$$Weight\_item\_1(9) = \sum_{j \in \mathcal{F}^C} a_{j9} \times N_j$$

$$= a_{(\mathrm{f1})9} \times N_{\mathrm{f1}} + a_{(\mathrm{f2})9} \times N_{\mathrm{f2}} + a_{(\mathrm{f3})9} \times N_{\mathrm{f3}} + a_{(\mathrm{f4})9} \times N_{\mathrm{f4}}$$

$$+ \quad a_{(\mathrm{f5})9} \times N_{\mathrm{f5}}$$

$$= 0 \times N_{\mathrm{f1}} + 0 \times N_{\mathrm{f2}} + 0 \times N_{\mathrm{f3}} + 1 \times 2 + 1 \times 2$$

$$= 4$$

The only sensitive itemsets that belong to $\mathcal{F}^{\mathrm{C}}$ and support item 9 are f4 and f5. Both f4 and f5 contain two items, so N_{f4} equals 2 and N_{f5} equals 2. The same procedure is followed for evaluating the rest of the items that belong to $\mathcal{F}^{\mathrm{C}}$. Following the same steps, the weights for all the items in L' are generated as seen in Table 8 below.

Table 8. Items in List L′ and Corresponding Weights

Item	***Weight_item*_1(*i*)**
9	4
7	3
8	2
4	7
5	7
1	4
2	4
3	6

Notice that there is a tie between items 4 and 5. One of these items must be selected at random. In this case, item 4 is chosen as I_H. Using I_H and *Weight_branch*_1(*b*), the weights for each branch *b* will be computed starting from branch B6.

$$Weight\_branch\_2(B6) = \sum_{j \in \mathcal{F}^C} c_{(B6)j} \times a_{jI_H}$$

$$= c_{(B6)\text{f1}} \times a_{(\text{f1})4} + c_{(B6)\text{f2}} \times a_{(\text{f2})4} + c_{(B6)\text{f3}} \times a_{(\text{f3})4}$$

$$+ c_{(B6)\text{f4}} \times a_{(\text{f4})4} + c_{(B6)\text{f5}} \times a_{(\text{f5})4}$$

$$= 1 \times 1 + 1 \times 0 + 1 \times 1 + 0 \times 0 + 0 \times 0$$

$$= 2$$

Referring back to Figure 11, only sensitive itemsets f1 and f3 support item 4 and both itemsets are supported in B6. The weight for branch B6 is 2. Following the same steps, the weights for all branches are generated as seen in Table 9 below.

Table 9. Branches in FP-Tree and Their Corresponding Weights

Branch	***Weight_branch_2(b)***
B1	0
B2	0
B3	0
B4	1
B5	1
B6	2

Branch B6 will be chosen and the node that contains item 4 will be modified. The corresponding new FP-Tree is shown below in Figure 12.

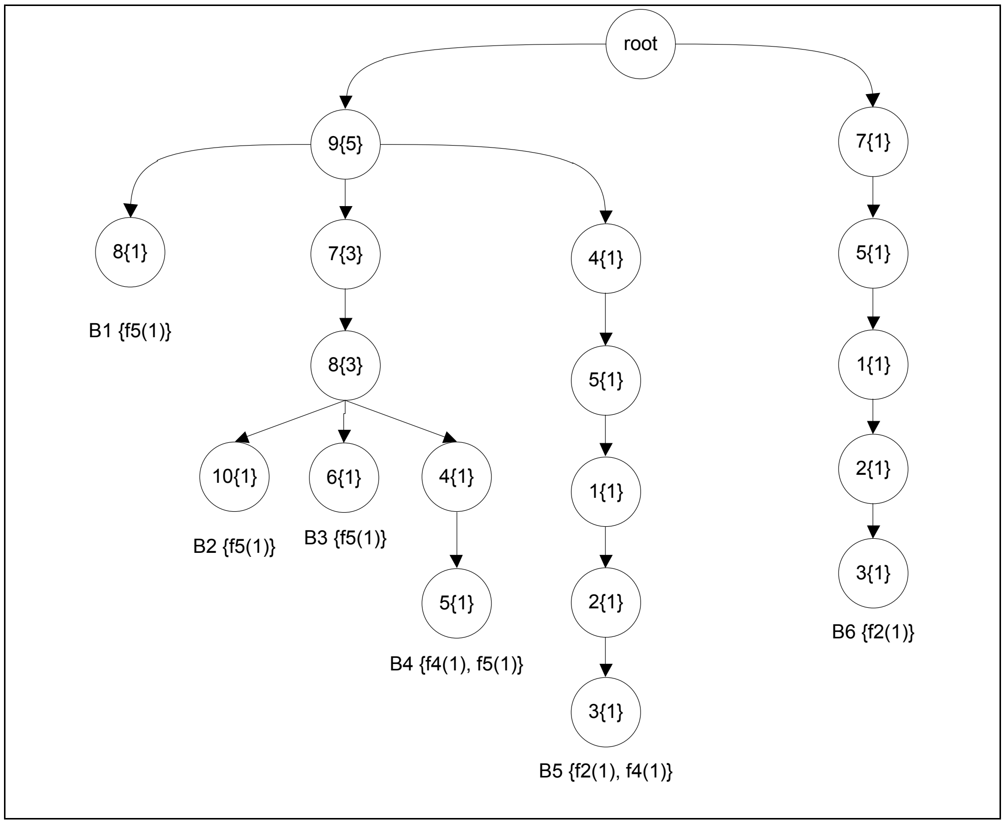

Figure 12. The new FP-Tree after the first iteration.

As can be seen from the new FP-Tree in Figure 12, f1 and f3 were removed from Branch B6; this will reduce the support level for each of these itemsets to 1. These itemsets are no longer considered frequent and the new $\mathcal{F}^C$ list is displayed below in Table 10. List L$'$ is also affected where items that are only unique to f1 and f3 are removed and the weights are adjusted (Table 11).

Table 10. $\mathcal{F}^C$, Sensitive Itemsets That are Still Frequent After Removing Item 4 from Branch B6

Sensitive Frequent Itemsets			**Support level**
f2	3	5	2
f4	5	9	2
f5	8	9	4

Table 11. List L$'$ After the Removal of Item 4

Item	$\boldsymbol{Weight\_item\_1(i)}$
9	4
8	2
5	4
3	2

The same steps that were described above by weighing items and branches iteratively is repeated until $\mathcal{F}^C$ is empty. Eventually, the final FP-Tree that is shown in Figure 13 is generated.

The final sanitized version of database $\mathcal{D}'$, i.e. $\mathcal{D}''$can be extracted from the final FP-Tree by processing each branch. For example, if branch B6 is analyzed, it is found that it represents the transaction $\{7, 5, 1, 2, 3\}$. This transaction is the sanitized version of the original transaction t8 = {7, 5, 4, 1, 2, 3} in $\mathcal{D}'$. The same approach can be followed to generate the final sanitized database version of $\mathcal{D}'$. This is shown in Table 12.

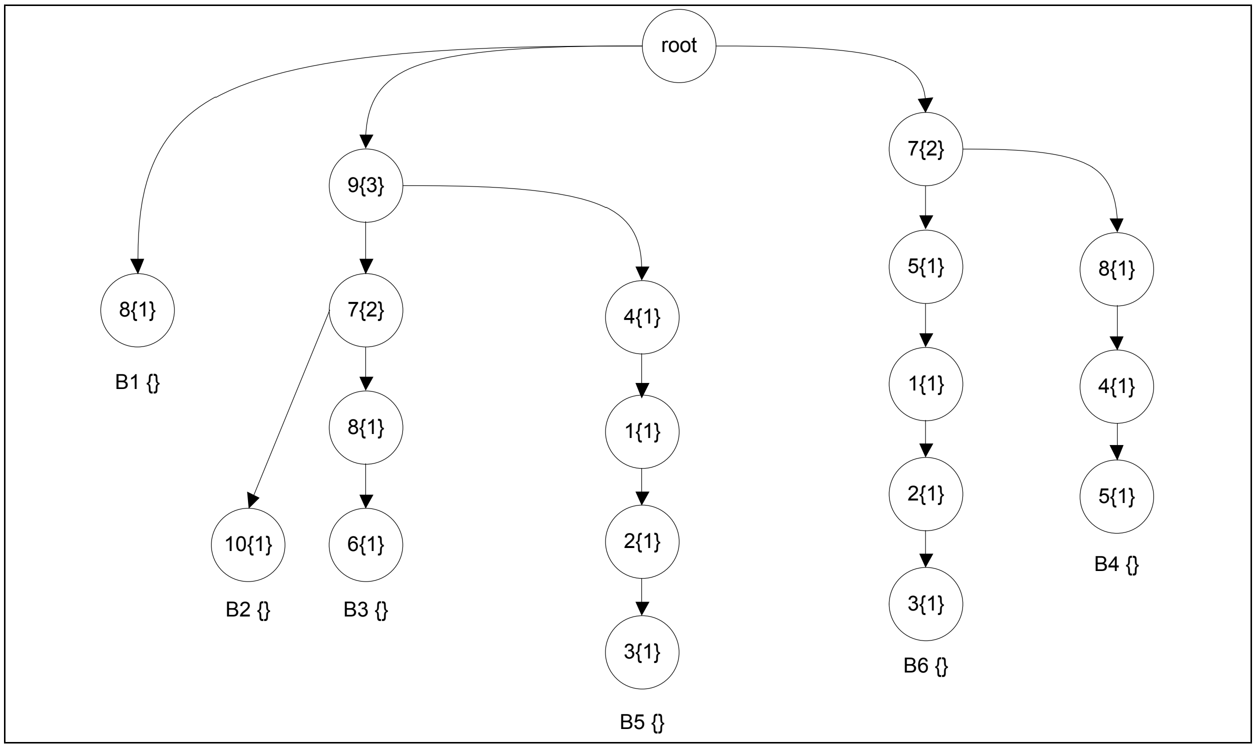

Figure 13. The final FP-Tree after sanitization.

Table 12. The Final Sanitized Database $\mathcal{D}''$

Transaction ID	**Items**				
Sanitized t2	9	7	8	6	
Sanitized t4	7	8	4	5	
Sanitized t6	9	4	1	2	3
Sanitized t8	7	5	1	2	3
Sanitized t9	8				
Sanitized t10	9	7	10		

Evaluation of the New Methodology

In order to measure its effectiveness, this new method was evaluated in comparison to other methods mentioned in extant literature, reviewed earlier in this paper. The comparative evaluation was based on the total number of items removed from the sanitized version of the database. The total number of items removed from the sanitized version of the database using the Min_Items_Removed($\mathcal{D}, \sigma_{\mathrm{min}}, \mathcal{F}^{\mathrm{C}}$) method was compared with the total number of items removed using the method described by Menon and Sarkar (2007). The method described by Menon and Sarkar (2007) is an exact approach that was used as a reference to measure the performance of the new method.

Two datasets were used for the purpose of this evaluation. The datasets are publically available by the first workshop on Frequent Itemset Mining Implementations repository. The same set of datasets was used by Menon, Sarkar and Mukherjee (2005) and was described by Goethals and Zaki (2003).

Databases differ in size or the number of transactions; they also differ in terms of the average transaction length and in the number of items. Table 13 shows the two databases that were used for evaluation (Menon, Sarkar, & Mukherjee, 2005). Each database exhibits different characteristics—size, average transaction length and number of items. This approach will highlight the advantages of the new method as well as any shortcomings.

Depending on the minimum support level used, a different number of nonsingleton itemsets frequent itemsets that support more than one item can be found for the same database (Menon, Sarkar, & Mukherjee, 2005). Menon, Sarkar and Mukerjee (2005) used different minimum support levels for every database as shown in Table 13. In this study the minimum support levels that were defined by Menon, Sarkar, and Mukherjee (2005) were used.

Menon and Sarkar (2007) also experimented with different numbers of sensitive itemsets for each database. They experimented with a set size of 5, 10, 20, and 30 sensitive itemsets. In this study, the approach that was defined by Menon and Sarkar (2007) will be followed using a set size of 10, 30, and 50 sensitive itemsets.

Table 13. Databases That Were Used for Evaluation with Their Characteristics

Database Name	**Number of Transactions**	**Number of Items**	**Average Transaction Length**	**Minimum Support Level Used**	**Number of Nonsingletone Frequent Itemsets**
kosarak	990,002	41,270	8.1	4,950 (0.50%)	1,463
retail	88,162	16,470	10.3	88 (0.10%)	5,572

In order to perform the evaluation, the sensitive itemsets that were used for each database had to be defined. Menon and Sarkar (2007) defined some thresholds to choose the sensitive itemsets. They chose itemsets that had support levels that are higher than $1.25\sigma_{min}$, $1.5\sigma_{min}$, and $2\sigma_{min}$, where σ_{min} is the minimum support level. In this study, the sensitive itemsets were chosen randomly.

Metrics Used to Quantify Performance

Several metrics were used by different studies (Verykios & Gkoulalas-Divanis, 2008). In this study, the focus was on measuring the total number of items that are removed by the new method compared to Menon and Sarkar (2007). Other metrics were collected to shed more light on the performance of the new method. For example, the number of the frequent and nonsensitive itemsets that were hidden was measured. The time for each algorithm to perform the hiding process was also measured along with the number of modified transactions.

Experiments Used to Quantify Performance

Several experiments were run during the evaluation of the new approach for each database. For each database, the first experiment attempted to use the FIH algorithm that was described by Menon and Sarkar (2007). The new FP-Tree approach was evaluated for several combinations of weight functions using the same database that was used in the first experiment. The combinations of the weight functions are described in Table 14 and Table 15. Note that the weight functions Weight_branch_2 and Weight_branch_4 were used only when the item is chosen first and that Weight_branch_1 and Weight_branch_3 were used only when the branch is chosen first.

Table 14. Different Experiments That Were Evaluated Where Item was Chosen First

Combination ID	**Weight Function for Items**	**Weight Function for Branches**
Combination 1	*Weight_item_1*	*Weight_branch_2*
Combination 2	*Weight_item_2*	*Weight_branch_2*
Combination 3	*Weight_item_1*	*Weight_branch_4*
Combination 4	*Weight_item_2*	*Weight_branch_4*

Table 15. Different Experiments That Were Evaluated Where Branch was Chosen First

Combination ID	**Weight Function for Branches**	**Weight Function for Items**
Combination 5	*Weight_branch_1*	*Weight_item_1*
Combination 6	*Weight_branch_1*	*Weight_item_2*
Combination 7	*Weight_branch_3*	*Weight_item_1*
Combination 8	*Weight_branch_3*	*Weight_item_2*

Resources

In order to replicate the work of Menon and Sarkar (2007), their programming method was implemented using Java. A linear (integer) programming solver "CPLEX" was used to replicate the integer programming approach that was followed by Menon and Sarkar (2007).

Specific sensitive itemsets that were used by Menon and Sarkar (2007) for the retail database were provided by Menon and Sarkar (2007). These sensitive itemsets were used to replicate Menon and Sarkar's work (2007) for the retail database. Satisfactory results were produced by this replication. This study has supported Menon and Sarkar and added to the evidence that their implementation is legitimate. The main metric that was considered in the evaluation of the replicated work was the nonsensitive frequent itemsets that were lost after sanitization. This was compared to the findings of Menon and Sarkar (2007) for the same retail database.

The next major code development was to implement the FP-Tree method that was described earlier. Several weight functions were implemented and called in order to weigh items and branches in order to hide the sensitive itemsets. The two developed codes were used to perform the experiments that were mentioned earlier.

Summary

In this chapter, the proposed methodology that was followed during the course of this dissertation was discussed. A heuristic that utilized the FP-Tree was the main contribution of this research. Specific steps followed in, in an attempt to find satisfactory solutions to the frequent itemsets hiding problem were described.

The specific databases that were used in previous studies were identified and the initial set of parameters used were presented. The metrics used to compare the outcome of the new method to the outcome of other studies were mentioned.

Chapter 4

Results

Introduction

This study has addressed the development and evaluation of a new heuristic that attempted to find satisfactory solutions for a version of the frequent itemset hiding problem, which aimed to minimize the total number of items that are removed from transactions in sanitizing the database.

The method utilized the FP-Tree to sanitize the database in order to reduce the total number of items removed during sanitization. The FP-Tree was generated for relevant transactions—the transactions that support sensitive itemsets. Eight heuristics were used to sanitize the FP-Tree (combinations 1 through 8 as listed in Table 14 and Table 15 in Chapter 3). The method suggested by Menon and Sarkar (2007) was also used. Results for both methods will be presented and compared in this chapter.

The methods were run using different problem instances. A problem instance is defined as a set of sensitive itemsets, an input database, and a minimum support level. Problem instances were grouped in several groups for each input database; each group contained fifty problem instances. The problem instances in each group contained the same number of sensitive itemsets that were generated randomly for every problem instance. Table 16 lists each group and the problem instances used for each database.

Table 16. Details of the Databases, Groups, and Problem Instances used

Database	Group	Number of Problem Instances	Number of Sensitive Itemsets per Problem Instance	Minimum Support Level
Retail	Group 1	50	10	88 (0.10%)
Retail	Group 2	50	30	88 (0.10%)
Retail	Group 3	50	50	88 (0.10%)
Kosarak	Group 1	50	10	4,950 (0.50%)

For each problem instance, several metrics were collected in order to compare all methods in terms of solution quality. Each metric was averaged for each group. The first metric is the total number of items that were removed from the database during sanitization. The second metric is the number of nonsensitive and frequent itemsets that were hidden or lost (Menon & Sarkar, 2007). The third metric, which is related to the accuracy metric defined by Menon, Sarkar, and Mukherjee (2005), is the total number of transactions that were affected by the sanitization process—i.e., the transactions in which items were removed during sanitization. The fourth metric is the total time taken to perform the sanitization process. The total time does not include the preprocessing of the data.

Each problem instance has some level of difficulty in terms of processing time and the effect processing has on the quality of the solution. The difficulty of the problem (problem size) was measured in terms of two metrics—the average support level of the sensitive itemsets for each problem instance and the normalized number of leaves in the generated FP-Tree for each problem instance. The number of leaves was normalized to the total number of transactions in the original database. It is expected that a higher average support level for the sensitive itemsets or a higher normalized number of leaves for each problem instance will yield higher difficulty for that problem instance.

A time limit was imposed for all algorithms to reach a solution for each problem instance. For the retail database a time limit of ten minutes was imposed for the sanitization of the FP-Tree and for CPLEX to solve the integer program for Menon and Sarkar (2007). For the kosarak database the time limit was chosen to be twenty four hours.

The following section will present the results of the experiments that were conducted. Results for each group will be listed separately. The relationship between the problem size and other metrics, such as the total items removed and the total time taken to perform the sanitation process will be presented.

Results

This section will present and compare the results for each group in terms of the total number of items removed, the total time taken to finish the sanitization process, the number of lost nonsensitive and frequent itemsets, and the number of modified transactions.

Number of Total Items Removed

Table 17 lists the average number of total items removed for the fifty problem instances in each group. The total number of items removed is given for each method per group.

Combination 3 removed fewer items for all groups in both databases. The method suggested by Menon and Sarkar (2007) removed more items compared to all other combinations. This is expected since the FP-Tree method is designed to minimize the total number of items removed while the method suggested by Menon and Sarkar (2007) was designed to minimize the total number of nonsensitive and frequent itemsets that are lost.

Table 17. The Total Number of Items Removed for Each Group and Method

Algorithm	**Retail**			**Kosarak**
	Group 1	**Group 2**	**Group 3**	**Group 1**
Menon and Sarkar (2007)	36518	97367	154331	2575069
FIH Combination 1	1584	4603	8188	52614
FIH Combination 2	2726	8256	13249	89101
FIH Combination 3	**1466**	**4236**	**7691**	**50097**
FIH Combination 4	2781	8093	12824	83753
FIH Combination 5	2172	6789	11858	76424
FIH Combination 6	2511	7431	12433	99116
FIH Combination 7	1920	5883	10333	65124
FIH Combination 8	2026	6145	10656	73086

Table 18 lists the number of problem instances in which each algorithm removed fewer items than the other algorithms for all problem instances in every database. A blank entry indicates zero problem instances for that algorithm. For most of the problem instances, in both databases, combination 3 removed fewer items. For the kosarak database, all the algorithms did not reach a solution for one problem instance within the allotted time. The method by Menon and Sarkar (2007) did not reach a solution for three problem instances in the kosarak database within the allotted time.

Table 18. Number of Problem Instances Where Each Algorithm Removed Fewer Items

Algorithm	**Number of Problem Instances Where Each Algorithm Removed Fewer Items**	
	Retail	**Kosarak**
FIH Combination 3	142	36
FIH Combination 1	6	13
FIH Combination 7	2	
FIH Combination 2		
FIH Combination 4		
FIH Combination 5		
FIH Combination 6		
FIH Combination 8		
Menon and Sarkar (2007)		

Number of Total NonSensitive and Frequent Itemsets That Were Lost

Table 19 lists the average number of nonsensitive and frequent itemsets that were lost for the fifty problem instances in each group. The number of nonsensitive and frequent itemsets that were lost is given for each method per group.

The method that was suggested by Menon and Sarkar (2007) resulted in fewer lost itemsets. This is expected since the method suggested by Menon and Sarkar (2007) was designed to minimize the total number of nonsensitive and frequent itemsets lost. On the other hand, the FP-Tree method is designed to minimize the total number of items removed. Combination 3 resulted in fewer itemsets lost compared to all other methods for both databases.

Table 19. The Number of Nonsensitive and Frequent Itemsets That Were Lost for Each Group and Method

Algorithm	**Retail (5572 frequent itemsets)**			**Kosarak (1463 frequent itemsets)**
	Group 1	**Group 2**	**Group 3**	**Group 1**
Menon and Sarkar (2007)	**9**	**12**	**32**	**25**
FIH Combination 1	23	55	115	142
FIH Combination 2	28	71	147	199
FIH Combination 3	23	53	108	110
FIH Combination 4	30	69	140	185
FIH Combination 5	25	63	139	181
FIH Combination 6	25	63	140	213
FIH Combination 7	23	59	116	121
FIH Combination 8	25	58	117	136

Table 20 lists the number of problem instances in which each algorithm lost fewer itemsets when compared to the other algorithms for all problem instances in every database. For most of the problem instances in both databases, the method suggested by Menon and Sarkar (2007) yielded fewer lost itemsets.

Table 20. Number of Problem Instances Where Each Algorithm Lost Fewer Itemsets

Algorithm	**Number of Problem Instances Where Each Algorithm Lost Fewer Itemsets**	
	Retail	**Kosarak**
Menon and Sarkar (2007)	145	47
FIH Combination 4	3	
FIH Combination 7	1	
FIH Combination 8	1	
FIH Combination 1		
FIH Combination 2		
FIH Combination 3		2
FIH Combination 5		
FIH Combination 6		

Number of Transactions That Were Modified

Table 21 lists the average number of transactions that were modified for the fifty problem instances in each group. The number of transactions is given for each method per group.

Combinations 3 and 1 modified fewer transactions for all groups in the retail database. The method suggested by Menon and Sarkar (2007) modified more transactions for the retail database in general.

Table 21. The Number of Transactions That Were Modified for Each Group and Method

Algorithm	**Retail**			**Kosarak**
	Group 1	**Group 2**	**Group 3**	**Group 1**
Menon and Sarkar (2007)	237	510	857	4891
FIH Combination 1	171	340	**525**	3731
FIH Combination 2	220	479	725	5089
FIH Combination 3	**150**	**336**	535	**3704**
FIH Combination 4	233	489	738	4949
FIH Combination 5	203	425	674	4521
FIH Combination 6	217	451	692	5161
FIH Combination 7	188	430	673	4249
FIH Combination 8	190	439	680	4316

Table 22 lists the number of problem instances in which each algorithm modified fewer transactions when compared to other algorithms for all problem instances in all groups. Combinations 3 and 1 modified fewer transactions than all other combinations.

Table 22. Number of Problem Instances Where Each Algorithm Modified Fewer Transactions

Algorithm	**Number of Problem Instances Where Each Algorithm Modified Fewer Transactions**	
	Retail	**Kosarak**
FIH Combination 1	68	1
FIH Combination 3	60	36
FIH Combination 4	18	12
FIH Combination 8	3	
FIH Combination 5	1	
FIH Combination 2		
FIH Combination 6		
FIH Combination 7		
Menon and Sarkar (2007)		

Time Taken to Sanitize the Database

Table 23 lists the average time taken to sanitize the database for the fifty problem instances in each group. The time taken is given for each method per group.

Table 23. Time to Sanitize the Database for Each Group and Method

Algorithm	**Retail (seconds)**			**Kosarak (seconds)**
	Group 1	**Group 2**	**Group 3**	**Group 1**
Menon and Sarkar (2007)	24	18	300	10224
FIH Combination 1	5	15	37	4527
FIH Combination 2	8	8	8	7810
FIH Combination 3	**5**	**5**	**5**	**2572**
FIH Combination 4	8	8	8	6536
FIH Combination 5	**5**	**5**	**5**	6705
FIH Combination 6	**5**	**5**	**5**	8794
FIH Combination 7	12	12	12	8020
FIH Combination 8	12	12	12	9271

Combination 3 took less time on average to sanitize the database for the fifty problem instances in each group, while the method suggested by Menon and Sarkar (2007) took more time. Table 24 lists the number of instances in which each algorithm took less time to sanitize the database when compared to other methods. Menon and Sarkar's (2007) method took less time for the majority of the problem instances, which are considered easy problem instances. For large problem instances, their method took much more time to find a good solution. Combinations 5 and 3 took less time to sanitize large problem instances.

Table 24. Number of Problem Instances Where Each Algorithm Took Less Time to Sanitize the Database

Algorithm	**Number of Problem Instances Where Each Algorithm Took Less Time to Sanitize the Database**	
	Retail	**Kosarak**
Menon and Sarkar (2007)	135	38
FIH Combination 5	14	1
FIH Combination 8	1	
FIH Combination 1		
FIH Combination 2		
FIH Combination 3		10
FIH Combination 4		
FIH Combination 6		
FIH Combination 7		

Analysis of Problem Instances

The problem instances that were used to compare the different algorithms had different characteristics. Figure 14 shows the Cumulative Distribution Function (CDF) for all problem instances in each group in terms of average support level of the sensitive itemsets in each problem instance.

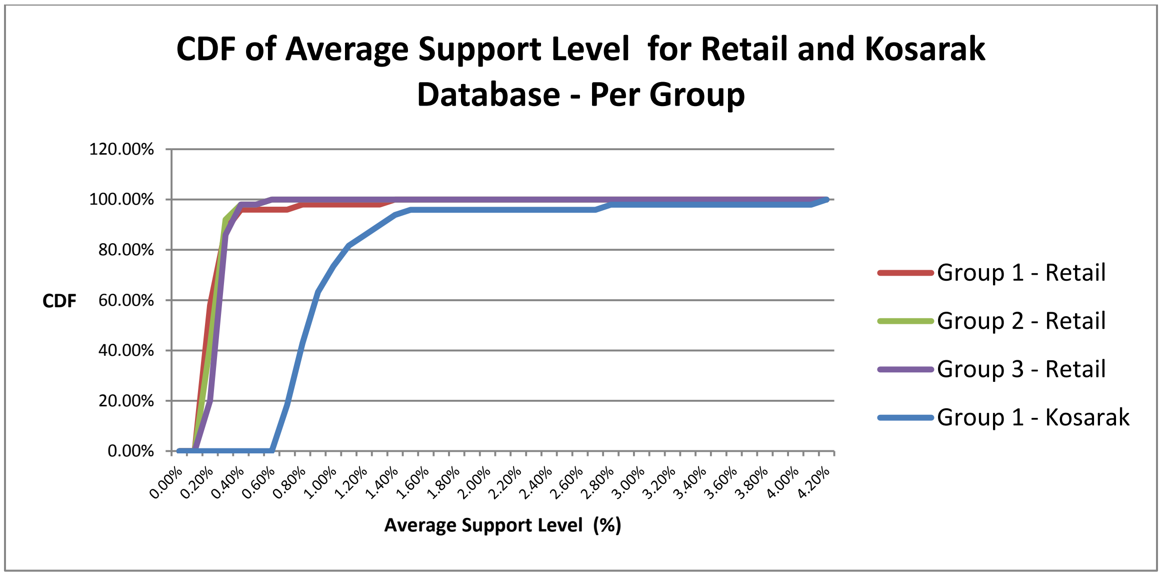

Figure 14. CDF of the average support level for the retail and kosarak databases.

For the retail database, the majority of the problem instances had an average support for the sensitive itemsets below 0.4%. For the kosarak database, the majority of the problem instances had an average support for the sensitive itemsets below 1.5%. The generated random sensitive itemsets for the problem instances did not yield very large problems for the retail database compared to the kosarak database.

Figure 15 shows the CDF for the normalized number of leaves for the generated FP-Tree for the retail and kosarak databases. The number of leaves was normalized to the total number of transactions in the retail database. The number of leaves increases when the number of sensitive itemsets in each problem instance is higher. It is expected that the difficulty of the problem will increase with higher number of sensitive itemsets.

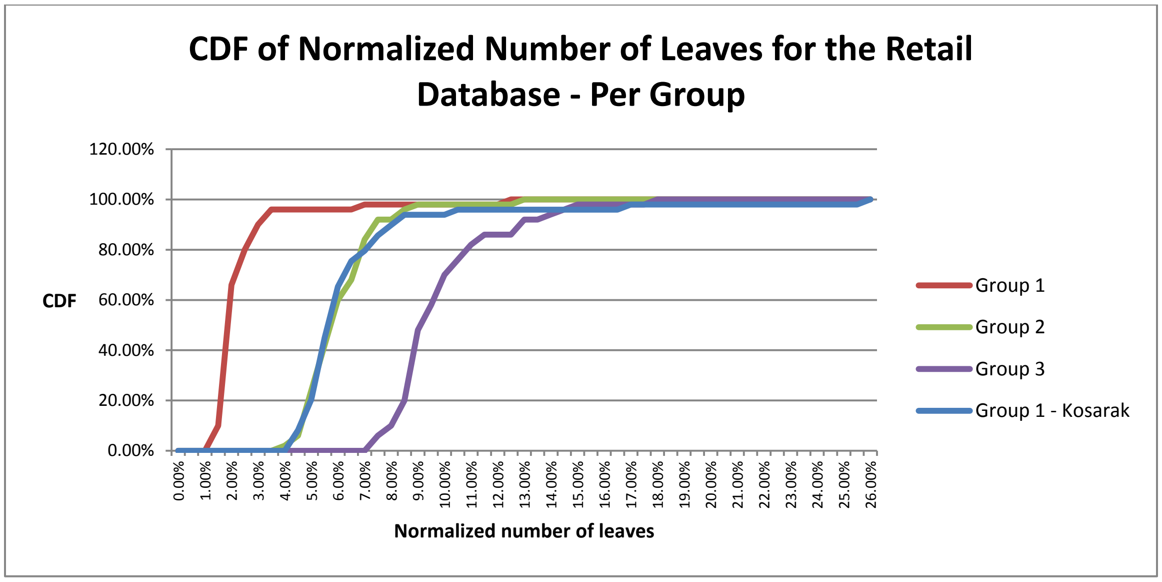

Figure 15. CDF of the normalized number of leaves for the retail and kosarak databases.

Problem Difficulty and Total Number of Items Removed

Figure 16 displays the total number of items removed during sanitization as a function of the average support level for the sensitive itemsets in each problem instance for the retail database using combination 3 and the method by Menon and Sarkar (2007). The figure illustrates that the total number of items removed increases with a higher average support level and that a greater number of items are removed in groups with a higher number of sensitive itemsets (e.g., group 3). The number of items removed increases linearly with the average support level per problem instance within each group for both algorithms.

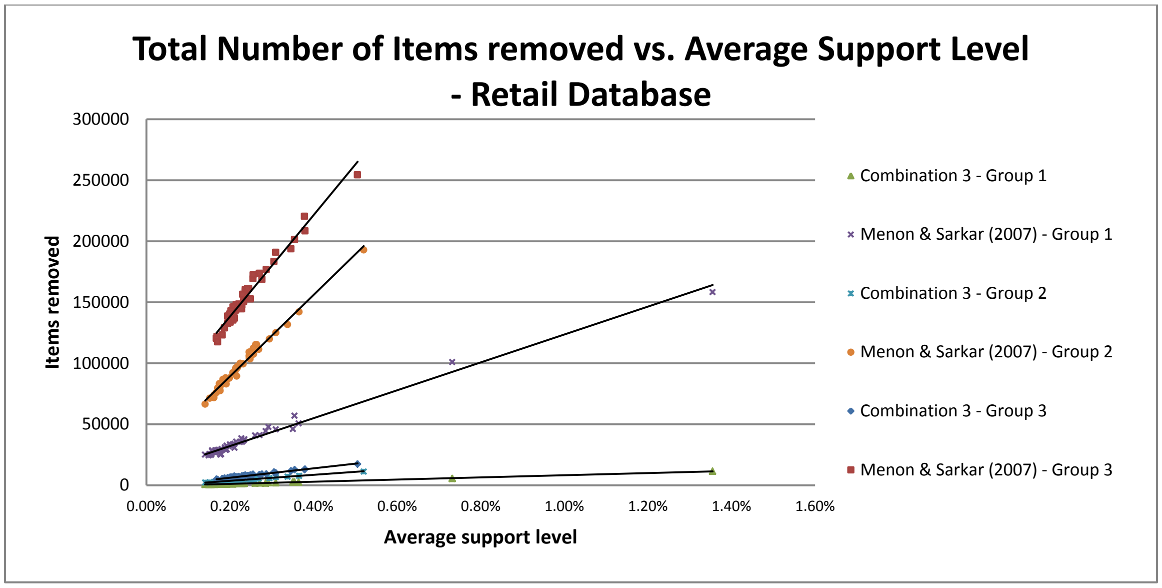

Figure 16. Total number of items removed vs. average support level for the retail database.

Figure 17 shows the total number of items removed during sanitization as a function of the average support level for the sensitive itemsets in each problem instance for the kosarak database using combination 3 and the method by Menon and Sarkar (2007). The total number of items removed increases with higher average support level and the number of items removed increases linearly with the average support level per problem instance for both algorithms.

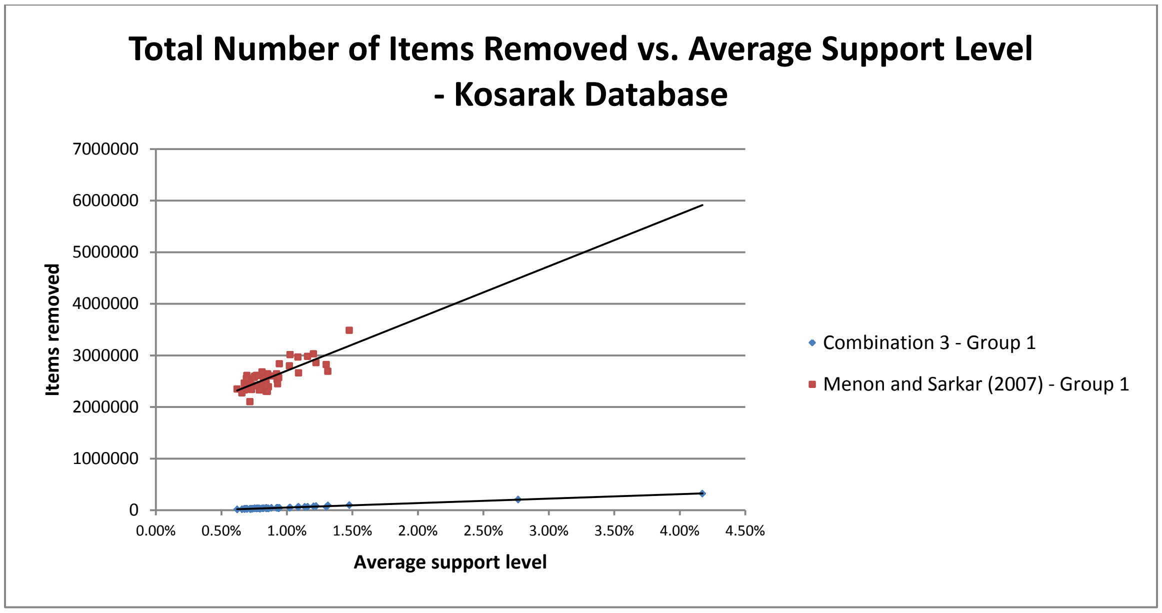

Figure 17. Total number of items removed vs. average support level for the kosarak database.

Figure 18 and Figure 19 show the total number of items removed during sanitization as a function of the normalized number of leaves for the generated FP-Tree in each problem instance for the retail and kosarak databases using combination 3 and the method by Menon and Sarkar (2007). The total number of items removed increases with a higher normalized number of leaves. The number of items removed increases linearly with the normalized number of leaves per problem instance for both algorithms. The FP-Tree approach removes fewer items than the method by Menon and Sarkar (2007).

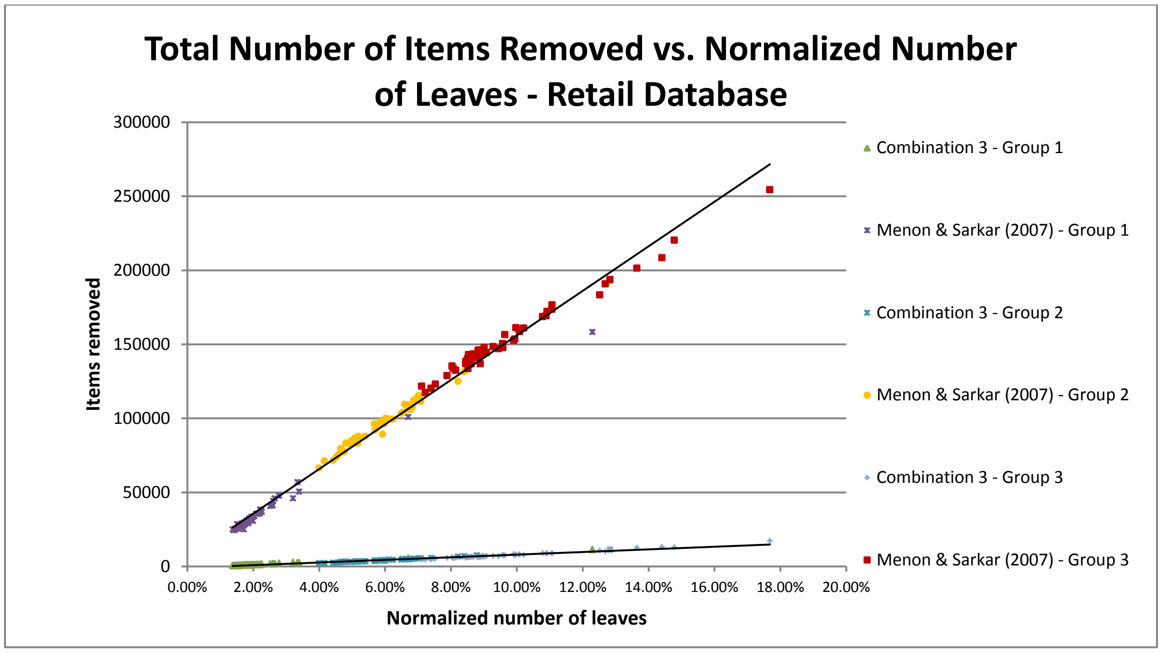

Figure 18. Total number of items removed vs. normalized number of leaves for the retail database.

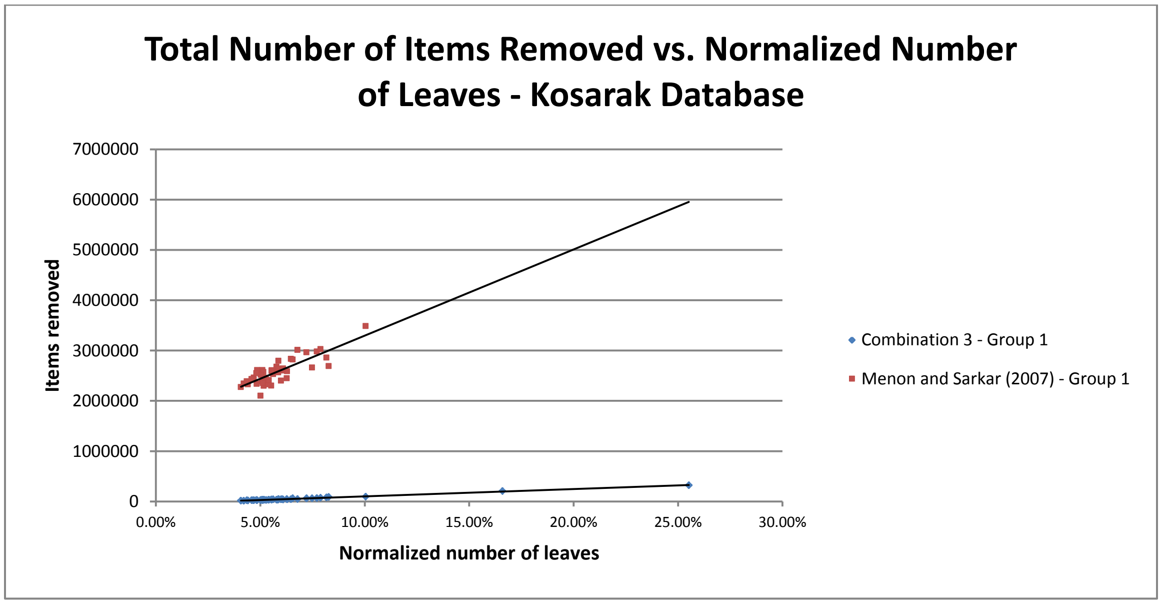

Figure 19. Total number of items removed vs. normalized number of leaves for the kosarak database.

Problem Difficulty and Total Time Needed

Figure 20 shows the total time taken during sanitization as a function of the average support level for the sensitive itemsets in each problem instance for the retail database using combination 3 and the method by Menon and Sarkar (2007). The total time taken increases with higher average support level; the total time also increases for groups with a higher number of sensitive itemsets (e.g., group 3). Menon and Sarkar's (2007) method took less time for the majority of the problem instances that are considered easy problem instances. However, for large problem instances, their method took much more time to find a good solution.

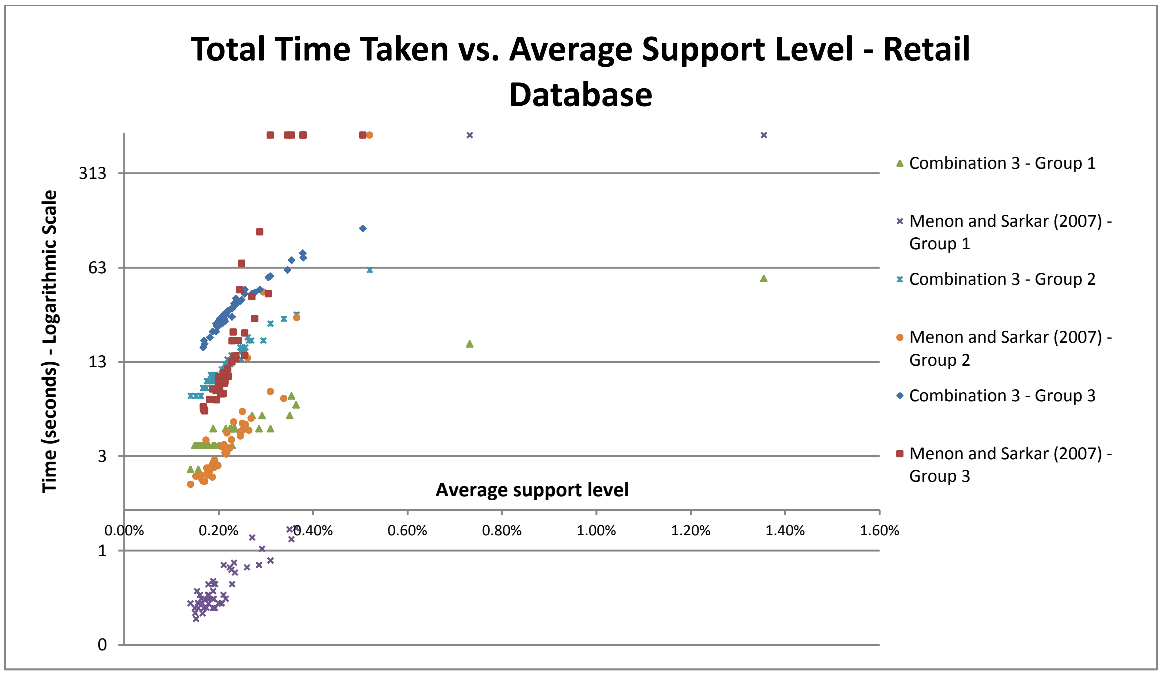

Figure 20. Total time taken vs. average support level for the retail database.

Figure 21 shows the total time taken during sanitization as a function of the average support level for the sensitive itemsets in each problem instance for the kosarak database using combination 3 and the method by Menon and Sarkar (2007). The total time taken increases with higher average support level. Menon and Sarkar’s (2007) method took less time for the majority of the problem instances that are considered easy problem instances. For large problem instances, their method took much more time to find a good solution.

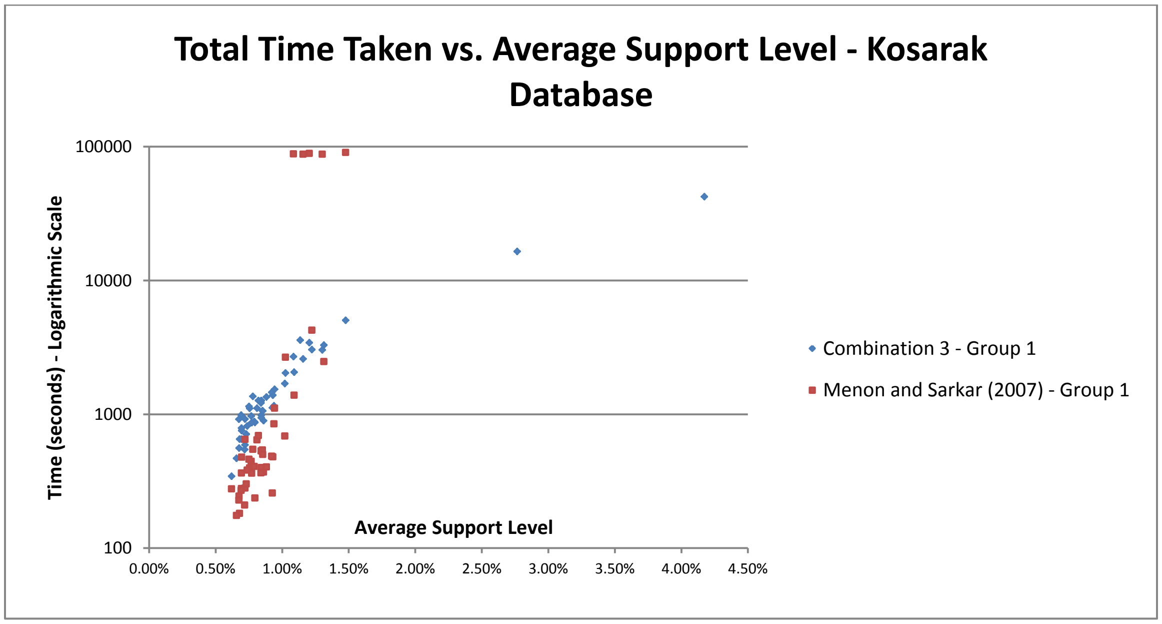

Figure 21. Total time taken vs. average support level for the kosarak database.

Figure 22 shows the total time taken during sanitization as a function of the normalized number of leaves for the generated FP-Tree in each problem instance for the retail database using combination 3 and the method by Menon and Sarkar (2007). The total time taken increases with the normalized number of leaves. The total time is a quadratic function of the normalized number of leaves per problem instance for combination 3 as shown in Figure 22.

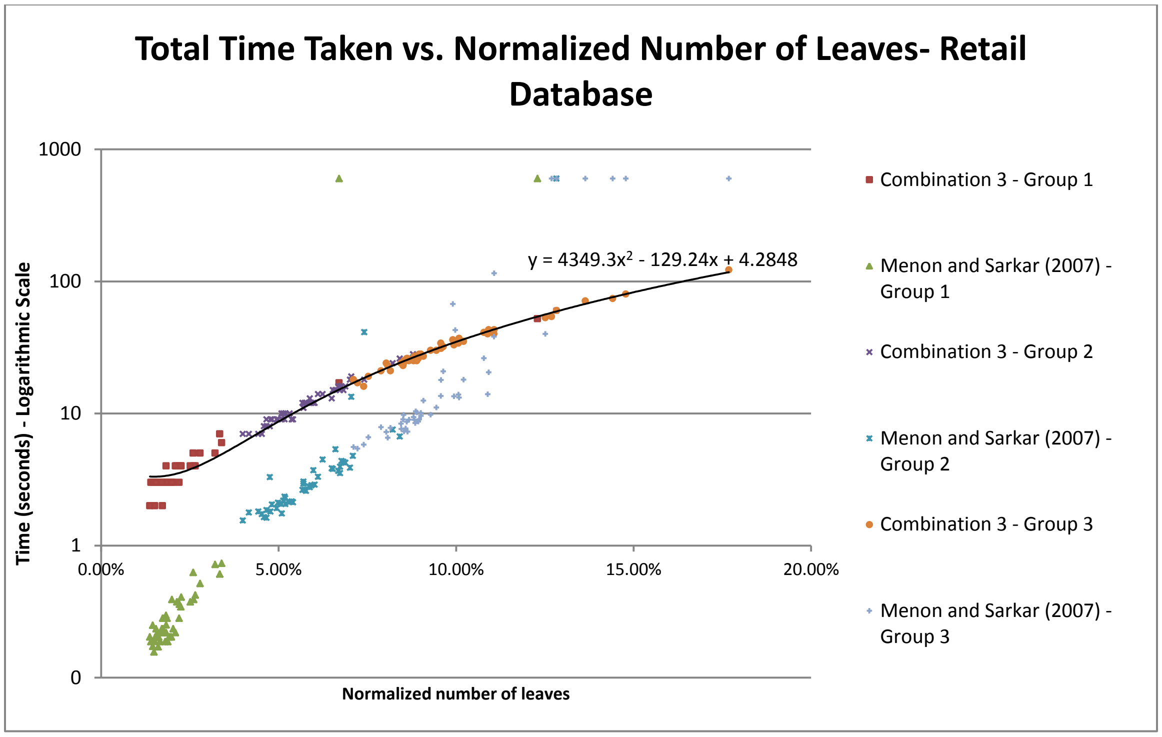

Figure 22. Total time taken vs. normalized number of leaves for the retail database.

Figure 23 shows the total time taken during sanitization as a function of the normalized number of leaves for the generated FP-Tree in each problem instance for the kosarak database using combination 3 and the method by Menon and Sarkar (2007). The total time taken increases with the normalized number of leaves. The total time is a quadratic function of the normalized number of leaves per problem instance for combination 3 as shown in Figure 23.

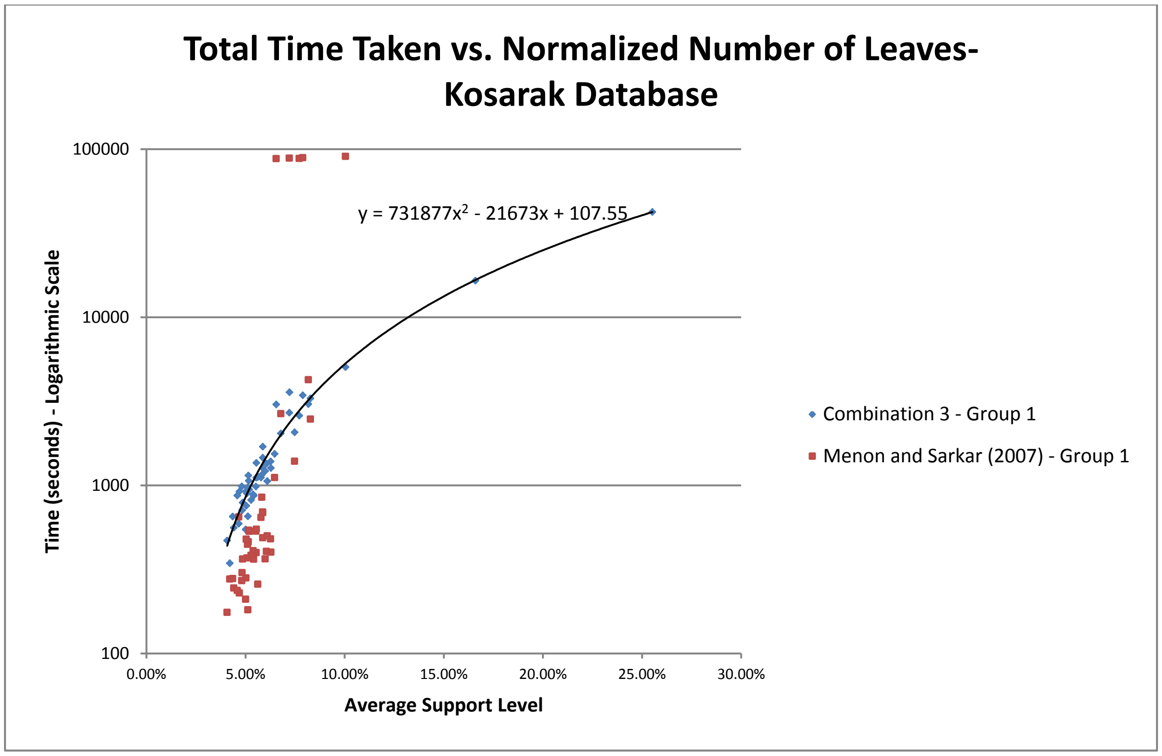

Figure 23. Total time taken vs. normalized number of leaves for the kosarak database.

Chapter 5

Conclusions, Implications, Recommendations, and Summary

Summary and Conclusions

A problem that has been the focus of much recent research in privacy preserving data-mining is the frequent itemset hiding (FIH) problem. Identifying itemsets that appear together frequently in customer transactions is a common task in association rule mining. Organizations that share data with business partners may consider some of the frequent itemsets sensitive and aim to hide such sensitive itemsets by removing items from certain transactions. Such modifications adversely affect the utility of the database for data mining applications. Several techniques were suggested in the literature to increase the utility of the shared database. Since the frequent itemset hiding problem is NP-hard and practical instances of this problem are too large to be solved optimally, there is a need for heuristic-based methods that provide good solutions. This is an active area of research and many have attempted to find good solutions. Chapter 2 presented a comprehensive review of the many prominent methods and techniques that were proposed in the literature to tackle this NP-hard problem. Heuristic-based methods included support-based and confidence-based distortion schemes (Attallah, Bertino, Elmagarmid, Ibrahim, & Verykios, 1999; Dasseni, Verykios, Elmagarmid, & Bertino, 2001; Oliveira & Zaïane, 2002; Pontikakis, Tsitsonis, & Verykios, 2004; Amiri, 2007; Wu, Chiang , & Chen, 2007; Wang & Hong, 2007; Modi, Rao, & Patel, 2010; Yildiz & Ergenc, 2011). Other techniques used support-based and confidence-based blocking schemes (Saygin, Verykios, & Clifton, 2001; Wang & Jafari, 2005). Border-based approaches, exact approaches and reconstruction-based approaches were also suggested among others (Sun & Yu, 2005; Moustakides & Verykios, 2006; Menon, Sarkar, & Mukherjee, 2005; Menon & Sarkar, 2007; Gkoulalas-Divanis & Verykios, 2006; Guo, 2007).

Methods suggested in the literature aimed to maximize the utility of the shared database using different goals, e.g., loss of nonsensitive itemsets, reduction in total number of items removed, and the production of new association rules. Reduction in the total number of removed items was not addressed enough in the literature. This dissertation built an FP-Tree-based algorithm that minimizes the total number of removed items and produces good results that are better than those obtained using extant methods in the literature. Chapter 3 introduced the Min_Items_Removed algorithm (refer to Figure 4) and provided its details. Nine experiments were conducted using different methods in order to measure the effectiveness of the Min_Items_Removed algorithm. Results were presented in Chapter 4. Observations that were made in Chapter 4 based on the results of the experiments addressed the research goal that was presented in Chapter 1.

The first experiment reproduced the work of Menon and Sarkar (2007). Their method is an exact method that aims to minimize the total number of the nonsensitive itemsets that are lost. Specific sensitive itemsets that were used by Menon and Sarkar (2007) for the retail database were provided by Menon and Sarkar (2007). These sensitive itemsets were used to replicate Menon and Sarkar's work (2007) for the retail database.

Satisfactory results were produced by this replication. This dissertation has supported Menon and Sarkar and added to the evidence that their implementation is legitimate.

The next eight experiments utilized the Min_Items_Removed algorithm. Each experiment used different weight functions (refer to Table 14 and Table 15). For every experiment, these weight functions identified the node that was chosen to be modified in the FP-Tree per iteration.

Observations and conclusions made in Chapter 4 showed the effectiveness of the Min_Items_Removed algorithm in finding better-quality solutions than existing methods in terms of minimizing the total number of removed items. Combination 3 was the algorithm that removed a smaller number of items compared to all algorithms. However, the results also showed the effectiveness of the method by Menon and Sarkar (2007) in terms of reducing the number of lost nonsensitive and frequent itemsets. This is expected, since the FP-Tree method is designed to minimize the total number of items removed, while the method suggested by Menon and Sarkar (2007) was designed to minimize the total number of nonsensitive and frequent itemsets that are lost.

Observations and conclusions made in Chapter 4 showed the effectiveness of the Min_Items_Removed algorithm in finding better-quality solutions than existing methods in terms of minimizing the total number of transactions that are modified. Combinations 1 and 3 were the algorithms that modified a fewer number of transactions compared to all algorithms. Also observations and conclusions made in Chapter 4 showed that the method by Menon and Sarkar (2007) took less time to sanitize easy problem instances, while combination 3 took less time to sanitize large problem instances.

Observations and conclusions made in Chapter 4 showed that the newly introduced metric (normalized number of leaves) is a very good indicator of the difficulty of the problem. The total number of removed items has a linear relation to this metric that is independent of the number of sensitive itemsets. In addition, the time taken to sanitize the database has a direct relation to this metric that is independent of the number of sensitive itemsets.

Observations and conclusions made in Chapter 4 showed that the newly introduced metric (average support level) is another good indicator of the difficulty of the problem. The total number of removed items has a linear relation to this metric, but is dependent on the number of sensitive itemsets. In addition, the time taken to sanitize the database has a direct relation to this metric that is also dependent on the number of sensitive itemsets.

Future Work

Findings that were presented in Chapter 4 clearly indicate that the Min_Items_Removed algorithm found good solutions to the FIH problem in terms of minimizing the removed items from the shared database after sanitization. On the other hand this dissertation revealed a limitation to the Min_Items_Removed algorithm: the higher processing time for easy problem instances. This limitation can be addressed by improving the algorithm to remove more than one item per iteration. Several items can be identified in every iteration, which can eventually lead to modifying more than one node. This will reduce the amount of time needed to sanitize the FP-Tree.

References

Amiri, A. (2007). Dare to share: Protecting sensitive knowledge with data sanitization. *Decision Support Systems, 43*(1), 181-191.

Atallah, M., Bertino, E., Elmagarmid, A., Ibrahim, M., & Verykios, V. (1999). Disclosure limitation of sensitive rules. *Workshop Knowledge Data Engrg* (pp. 45-52). Washington, D.C.: IEEE Computer Society.

Chen, X., Orlowska, M., & Li, X. (2004). A new framework of privacy preserving data sharing. *Proceedings of the 4th IEEE ICDM Workshop on Privacy and Security Aspects of Data Mining* (pp. 47-56). IEEE Computer Society.

Clifton, C., & Marks, D. (1996). Security and privacy implications of data mining. *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data (SIGMOD'96)*, (pp. 15-19).

Dasseni, E., Verykios, V., Elmagarmid, A., & Bertino, E. (2001). Hiding association rules by using confidence and support. *4th International Information Hiding Workshop (IHW)* (pp. 363-383). London, UK: Springer-Verlag.

Gkoulalas-Divanis, A., & Verykios, V. S. (2006). An integer programming approach for frequent itemset hiding. *Proceedings of the 15th ACM International Conference on Information and Knowledge Management* (pp. 748-757). Arlington, Virginia USA: ACM.

Gkoulalas-Divanis, A., & Verykios, V. S. (2009, May). Exact knowledge hiding through database extension. *IEEE Transactions on Knowledge and Data Engineering, 21*(5), 699-713.

Goethals, B. (2002). *Efficient frequent pattern mining.* Diepenbeek, Belgium: Ph.D. thesis, Universitiet Limburg.

Goethals, B., & Zaki, M. J. (2003, November). Advances in frequent itemset mining implementations: Report on FIMI'03. Melbourne, FL: Proc. Workshop Frequent Itemset Mining Implementations (FIMI'03).

Guo, Y. (2007). Reconstruction-based association rule hiding. *In Proc. Of SIGMOD2007 Ph.D. Workshop on Innovative Database Research 2007(IDAR2007).* Beijing, China.

Han, J., & Kamber, M. (2006). *Data Mining: Concepts and Techniques* (2nd ed.). Morgan Kaufmann.

Han, J., Pei, J., & Yin, Y. (2000). Mining frequent patterns without candidate generation. *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, 29*(2), 1-12.

Kantarcioglu, M., & Clifton, C. (2004). Privacy-preserving distributed mining of association rules on horizontally partitioned data. *Knowledge and Data Engineering, IEEE Transactions on, 16*(9), 1026 - 1037.

Menon, S., & Sarkar, S. (2007). Minimizing information loss and preserving privacy. *Management Science, 53*(1), 102-116.

Menon, S., Sarkar, S., & Mukherjee, S. (2005). Maximizing accuracy of shared databases when concealing sensitive patterns. *Information Systems Research, 16*(3), 256-270.

Modi, C. N., Rao, U. P., & Patel, D. R. (2010, July). Maintaining privacy and data quality in privacy preserving association rule mining. *International Conference on Computing Communication and Networking Technologies (ICCCNT), 2010*, (pp. 1-6). Karur, India.

Moustakides, G. V., & Verykios, V. S. (2006). A max-min approach for hiding frequent itemsets. *Workshops Proceedings of the Sixth IEEE International Conference on Data Mining (ICDM 2006)*, (pp. 502-506).

O'Leary, D. E. (1991). Knowledge discovery as a threat to database security. *Proceedings of the 1st International Conference on Knowledge Discovery in Databases*, (pp. 507-516).

Oliveira, S., & Zaïane, O. (2002). Privacy preserving frequent itemset mining. *IEEE ICDM Workshop Privacy, Security Data Mining* (pp. 43–54). Darlinghurst, Australia: Australian Computer Society.

Oliveira, S. R., & Zaïane, O. R. (2003). Protecting Sensitive Knowledge By Data Sanitization. *Proceeding ICDM '03 Proceedings of the Third IEEE International Conference on Data Mining* (pp. 211-218). Washington, DC, USA: IEEE Computer Society .

Pontikakis, E. D., Theodoridis, Y., Tsitsonis, A. A., Chang, L., & Verykios, V. S. (2004). A quantitative and qualitative analysis of blocking in association rule hiding. *Proceedings of the 2004 ACM Workshop on Privacy in the Electronic Society (WPES 2004)*, (pp. 29-30).

Pontikakis, E. D., Tsitsonis, A. A., & Verykios, V. S. (2004). An experimental study of distortion-based techniques for association rule hiding. In C. Farkas, & P. Samarati (Ed.), *Proceedings of the 18th Conference on Database Security (DBSEC 2004). 144*, pp. 325-339. Boston, USA: Springer.

Sarathy, R., & Muralidhar, K. (2002). The security of confidential numerical data in databases. *Information Systems Research, 13*(4), 389-403.

Saygin, Y., Verykios, V., & Clifton, C. (2001). Using unknowns to prevent discovery of association rules. *SIGMOD Rec., 30*(4), 45-54.

Sun, X., & Yu, P. (2005). A border-based approach for hiding sensitive frequent itemsets. *In Proceedings of the Fifth IEEE International Conference on Data Mining (ICDM 2005)*, (pp. 426-433).

Verykios, V. S., Bertino, E., Fovino, I. N., Provenza, L. P., Saygin, Y., & Theodoridis, Y. (2004). State of the art in privacy preserving data mining. *ACM SIGMOD Record, 33*(1), 50-57.

Verykios, V., Elmagarmid, A., Bertino, E., Saygin, Y., & Dasseni, E. (2004). Association rule hiding. *IEEE Trans. Knowledge Data Engrg, 16*(4), 434-447.

Verykios, V. S., & Gkoulalas-Divanis, A. (2008). A Survey of Association Rule Hiding Methods for Privacy. In C. C. Aggarwal, & P. S. Yu, *Privacy-Preserving Data Mining: Models and Algorithms* (Vol. 34, pp. 267-289). Springer, US.

Wang, S. -L., & Jafari, A. (2005). Using unknowns for hiding sensitive predictive association rules. *Proceedings of the 2005 IEEE International Conference on Information Reuse and Integration (IRI 2005)*, (pp. 223-228).

Wang, S.-L., & Hong, T.-P. (2007). One-scan sanitization of collaborative recommendation association rules. *Proceedings of National Computer Symposium*, (pp. 170-176). Taichun, Taiwan.

Wu, Y.-H., Chiang , C.-M., & Chen, A. L. (2007). Hiding eensitive association rules with limited side effects. *IEEE Transactions on Knowledge and Data Engineering, 19*(1), 29-42.

Yildiz, B., & Ergenc, B. (2011). Hiding sensitive predictive frequent itemsets. *Proceedings of the International MultiConference of Engineers and Computer Scientists*, (pp. 339-445). Hong Kong.

Zhu, Z., & Du , W. (2010). K-anonymous association rule hiding. *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security* (pp. 305-309). Beijing, China: ACM.