International Journal of Data Warehousing and Mining
Volume 14 • Issue 2 • April-June 2018

Dynamic Itemset Hiding Algorithm for Multiple Sensitive Support Thresholds

Ahmet Cumhur Öztürk, İzmir Institute of Technology, İzmir, Turkey
Belgin Ergenç, İzmir Institute of Technology, İzmir, Turkey

ABSTRACT

This article describes how association rule mining is used for extracting relations between items in transactional databases and is beneficial for decision-making. However, association rule mining can pose a threat to the privacy of the knowledge when the data is shared without hiding the confidential association rules of the data owner. One of the ways hiding an association rule from the database is to conceal the itemsets (co-occurring items) from which the sensitive association rules are generated. These sensitive itemsets are sanitized by the itemset hiding processes. Most of the existing solutions consider single support thresholds and assume that the databases are static, which is not true in real life. In this article, the authors propose a novel itemset hiding algorithm designed for the dynamic database environment and consider multiple itemset support thresholds. Performance comparisons of the algorithm is done with two dynamic algorithms on six different databases. Findings show that their dynamic algorithm is more efficient in terms of execution time and information loss and guarantees to hide all sensitive itemsets.

KEYWORDS

Dynamic Itemset Hiding, Itemset Hiding, Multiple Sensitive Support Thresholds, Privacy Preserving Association Rule Mining

1. INTRODUCTION

Data mining is the process of extracting knowledge from data with the help of statistics, artificial intelligence, machine learning and database systems. Association rule mining is one of the data mining tasks. It was first proposed by (Agrawal and Srikant, 1994) and is used for discovering correlated items transactional databases. Association rule mining process has mainly two steps; the first step is called frequent itemset (co-occurring items) generation and the second step is called rule generation where meaningful rules are generated from the discovered frequent itemsets. The second step is straightforward and similar in all proposed algorithms; as a result, association rule mining algorithms focus on the first step which is computationally expensive. For this reason, terms association rule mining and itemset mining are used interchangeably.

Lately many organizations use itemset mining tasks for short or long-term planning and strategical decision making. In modern business, organizations also share data with each other or with third parties in order to provide extraction of knowledge for mutual benefit. Similarly, itemset mining tasks are applied on this shared data however this may pose security threat for strategical and sensitive information of data owners.

The importance of this threat is well explained with a scenario given in (Clifton and Marks, 1996). Suppose BigMart supermarket chain is negotiating with DedTrees Paper Company for selling their

DOI: 10.4018/IJDWM.2018040103

products, and DedTrees offers to reduce their price if BigMart agrees to share sales database. After BigMart agrees to share the sales database, DedTrees applies itemset mining task on this database and finds out that people who purchase skim milk also purchase Green paper. Dedtrees Company then runs a coupon marketing campaign that gives 50 cents off for each purchase a Dedtrees product. The campaign cuts heavily sales of Green paper and as a result Green paper has to increase prices because of the low sales amount. In the next negotiation with DedTrees, they are unwilling to reduce their prices because they reach their goal. As a result, the BigMart suffers serious losses to competitors. This scenario shows that before the data is shared with other parties, the database owner should take precautions to protect its strategical and sensitive knowledge from being discovered by itemset mining task. Privacy preserving itemset mining is the problem of preserving the sensitive itemsets from being discovered in case of data sharing.

The most popular approach for sanitizing sensitive and frequent itemsets is to decrease their frequency (support) under predefined support threshold. Such a modification operation of converting the original database D into a sanitized database D' is called frequent itemset hiding. A well designed frequent itemset hiding algorithm should hide all given sensitive itemsets while keeping the loss of non-sensitive itemsets, production of new artificial frequent itemsets and the distortion done on the database at minimum. Most proposed frequent itemset hiding approaches allow user to define a single support threshold for each sensitive itemset and assume that the databases are static (Amiri, 2007; Li et al., 2007; Weng et al., 2008; Dehkordi and Dehkordi, 2016; Verykios et al., 2004; Pontikakis et al., 2004; Hong et al., 2013; Cheng et al., 2016). Single support threshold barrier does not suit the nature of different itemsets; in transactional databases, frequency of some sensitive itemsets may be too high while some sensitive itemsets may be too low. Assigning unique single sensitive support threshold for each sensitive itemsets may result in decreasing the frequency of some itemsets more than required.

On the other hand, transactional databases are dynamic; they get updates continuously. When dynamicity of the databases is considered applying a sensitive itemset hiding algorithm from the start will result in redundant execution time and memory allocation. In (Dai and Chiang, 2010; Jadav et al., 2014) dynamic sanitization algorithms are proposed; they either do the sanitization on the whole database or on the update only. The proposed approach in (Dai and Chiang, 2010) uses a tree like data structure to speed up the execution time and does the sanitization on the whole database. This approach is good for minimizing the side effects and achieving optimum sanitization however the data structure used is inadequate for dense databases when the given set of sensitive itemsets contains a certain number of overlapping items. As the number of overlaps increase it becomes impossible to uncover all sensitive itemset supporting transactions from this data structure. These supporting transactions are called sensitive transactions and uncovering inadequate number of sensitive transactions may result in smaller search space of sensitive transactions. So the sanitized database may keep containing sensitive itemsets. The proposed approach in (Jadav et al., 2014) only modifies the transactions in the updated part and does not use any data structure to speed up the execution time. Although this approach guarantees hiding all sensitive itemsets, since the modification is always done on the incremental part, distortion on the database and loss of non-sensitive information cannot be kept at minimum.

In this paper a dynamic frequent itemset hiding algorithm *DynamicPGBS* with four major processes is proposed: Initialization, Increment Handling, Hiding and Publish Database. Hiding process is an extension of PGBS (Öztürk and Ergenç-Bostanoğlu, 2017). The *DynamicPGBS* is designed for hiding a given set of sensitive itemsets by deleting one or more items from adequate number of transactions while minimizing the execution time, memory requirement, distortion on the updated database and loss of non-sensitive knowledge. The main contributions of the dynamic algorithm are; 1) different sensitive support thresholds can be assigned to sensitive itemsets, 2) sanitization is done by considering whole transactions of the database that means large search space of sensitive transactions and less side effects on the sanitized database, 3) hiding process is designed for incremental environment, 3) it guarantees hiding all given sensitive itemsets. In the performance evaluation, *DynamicPGBS* is compared with similar counterparts SPITF (Dai and Chiang, 2010) and

RHID (Jadav et al., 2014) algorithms. The performance evaluation is conducted in order to measure execution time, distortion given to the data and knowledge and memory requirement. The experiments demonstrate that *DynamicPGBS* achieves significant improvement over both SPITF and RHID algorithms in terms of execution time and non-sensitive information loss in the sanitized database.

This paper is organized as follows. Definitions, preliminary background information related to itemset mining, itemset hiding and dynamic itemset hiding are given in the second section. In section three detailed related work is presented and discussed with different attributes of heuristic based hiding algorithms. The proposed frequent itemset hiding solution for dynamic environment is introduced in the fourth section. Performance analysis of the proposed algorithm on 6 different databases in comparison to two different algorithms is given in the fifth section. Conclusion is given in the final section.

2. BACKGROUND

Notation and concepts related to itemset mining and itemset hiding are explained in the first two subsections. As one may notice itemset mining and itemset hiding are two sides of a coin; it is not possible to understand itemset hiding without prior knowledge on itemset mining. In the last subsection dynamic itemset hiding strategies are discussed with an example.

2.1. Itemset Mining

Association rule mining aims to find unforeseen correlations among items within a dataset. Association rule mining algorithms have two consecutive phases; 1) finding frequent itemsets which enumerates itemsets and check their frequency and 2) generating frequent association rules from frequent itemsets of the previous phase. Since first phase is computationally expensive most of the algorithms focus on this phase and contain different effective approaches and methods and the terms association rule mining and itemset mining are used interchangeably. In order to find frequent itemsets and frequent association rules, algorithms use two important thresholds known as support and confidence. Support is used to measure the interestingness of itemsets whereas confidence is used to measure the interestingness of association rules. Itemset mining and association rule mining algorithms have different concerns e.g. finding association rules with high support and confidence (Agrawal and Srikant, 1994; Han et al. 2000), finding frequent itemsets with multiple support thresholds (Kiran and Reddy, 2011; Darrab and Ergenc, 2017), finding exception rules (Daly and Taniar, 2004), finding redundant association rules (Bastide et. al., 2000; Ashrafi et al., 2007), finding closed or maximal itemsets (Agarwal et. al, 2000; Pei et.al., 2000).

Formally, an itemset is defined as follows: Let $I = \{i_1, \ldots, i_n\}$ be a set of literals called items, an itemset X is a non-empty subset of I such that $X \subset I$. A transaction t is an ordered pair of items, i.e. $t \subset I$. A transactional database D is a database of transactions and total number of transactions in D is denoted as |D|. Support count of X is the number of transactions supporting X in D and it is denoted as scount(X), support of X is calculated as the ratio of scount(X) to |D| and denoted as supp(X). An itemset X is frequent if $supp(X) \geq \sigma$, where σ is the user specified minimum support threshold and the set of frequent itemsets is denoted as FI. Association rule on the other hand is an implication of the form X=>Y, where X and Y are database itemsets. The rule X => Y has support s, if s% of all transactions contain both X and Y. The rule X => Y has confidence c, if c% of transactions that contain X, also contain Y.

2.2. Itemset Hiding

Itemset hiding is the process of converting a given database into a sanitized database which does not contain sensitive itemsets of the data owner. A sensitive itemset, is a frequent itemset to be hidden from database D based on some privacy concerns of the database owner. SI is the set of sensitive itemsets, i.e. $SI \subset FI$. The sanitization process is transforming D into a new database D' from which

none of the sensitive itemsets can be extracted with the sensitive support threshold defined by the database owner (Atallah et al., 1999). Support of a sensitive itemset is decreased by removing items of sensitive itemsets from transactions during transformation. This type of sanitization is called distortion based sanitization and the main objective of distortion based sanitization is to hide all itemsets in SI while keeping the change on data and loss of non-sensitive knowledge at minimum.

Distortion based sanitization has two main challenges as identifying the right transactions for modification and selecting the right items to be removed from identified transactions. The first challenge is discovering all transactions containing sensitive itemsets called sensitive transactions. Each different sensitive transaction set for modification gives different side effects to the database. Second challenge is selecting the items to be removed from selected sensitive transactions. Usually the items having the maximum cover degree are selected; cover degree of an item i, is the number of sensitive itemsets that contain item i. Selecting the item with highest cover degree makes sense since if more than one sensitive itemset have a common item then removing this common item may sanitize more than one sensitive itemset at once (Pontikakis et al, 2004).

Distortion based frequent itemset hiding algorithms prevent the disclosure of private information, while preserving the utility of non-sensitive information as much as possible by minimum modification on the database. However, manipulating the database in order to protect the confidentiality of sensitive information brings out some side effects. These side effects are:

- **Hiding Failure (HF):** Measure of the amount of sensitive itemsets that are not hidden by the sanitization process. HF = |SI'| / |SI|, where |SI'| is the total number of sensitive itemsets in D' and |SI| is the total number of sensitive itemsets in D. The objective of majority of itemset hiding algorithms is to measure zero hiding failure whereas some of them allow tuning of hiding disclosure.
- **Information Loss (IL):** Measure of the amount of non-sensitive frequent itemsets (frequent itemsets except sensitive itemsets) lost by the sanitization process. IL = ((|FI| - |SI|) - (|FI'| - |SI'|)) / (|FI| - |SI|) where |FI| is the number of frequent itemsets and |SI| is the number of sensitive itemsets in D, |FI'| is the number of frequent itemsets in D' and |SI'| is the number of sensitive itemsets in D'. This metric mainly indicate the ratio of unintentionally removed itemsets by sanitization process.
- **Distance:** Measure of the total number of items removed by the sanitization process. Distance = (total number of items in D) – (total number of items in D'). This metric gives idea about the change on the original database caused by the sanitization process.

Besides these three metrics there is another metric named as "ghost rules" which shows the number of unintentionally created itemsets by the sanitization process. In other words ghost rules are the ones that do not exist in the original database but exist in the sanitized database. In this work, this side effect is not measured since proposed algorithm and the competitor algorithms do not create any ghost rules. They are mostly produced by reconstruction based itemset hiding algorithms where sanitization includes addition of non-sensitive itemsets in order to decrease the support of sensitive itemsets.

2.3. Dynamic Itemset Hiding

In dynamic environment transactional databases are continuously updated by receiving increments. The objective of dynamic itemset algorithms is to establish mechanisms that allow arrival of increments and generation of sanitized database whenever needed without requiring the repeat of sanitization process from scratch. Dynamic sanitization mechanisms face the challenge of maintaining large enough number of sensitive transaction search space in order to keep the potential side effects at minimum.

Dynamic itemset hiding algorithms hide sensitive itemsets by modifying only the incremental part or modifying the whole updated database. In the first strategy, original database D is already sanitized, then sanitizing only the incremental part d and combining it with D will result a sanitized

updated database. Although this strategy is more efficient in terms of execution time and resource allocation, it gives more side effects such as information loss and distance to the sanitized database because all potential sensitive transactions will not be in the search space. In the second strategy when incremental part d arrives it is combined with the original part D and then the sanitization operation is performed on the whole updated database $\left(D \cup d\right)$. This strategy is more efficient in terms of side effects because the sensitive transaction search space with all potential transactions for modifications is greater than that of the first strategy. This strategy is inefficient in terms of execution time and resource allocation if necessary precautions are not taken.

Let us try to make the strategies clear with an example. Suppose a dynamic database as in Figure 1 (a) with original part D and incremental part d is given. Suppose again three sensitive itemsets with multiple support thresholds as shown in Figure 1 (b) is given, meaning the third party should not find their supports more than predefined sensitive thresholds.

Actual support of sensitive itemsets "AD", "CD" and "BD" are 28.6%, 14.3% and 0.0% respectively in the original part of the database. All these three sensitive itemsets are infrequent or already hidden in D that is their supports are less than the given sensitive thresholds. Assume after a certain time, a new batch of transactions called incremental part d is attached to the original part. The new support of sensitive itemsets "AD","CD" and "BD" in the updated database is now 50.0%, 30.0% and 10.0% respectively where all sensitive itemsets become frequent and need to be hidden.

Sanitizing only the incremental part of the database is dealing with small number of transactions (TID8, TID9 and TID10) and this will keep the execution time and memory allocation at minimum. A possible approach may be removing the item "d" from transactions TID8, TID9 and TID10. As a result, 3 items will be removed from the new sanitized database and if the resulting database is mined with a frequent itemset mining algorithm with 10% minimum threshold then there are 28 frequent itemsets. This strategy cannot consider all potential modification operations and might not choose the optimum solution in terms of side effects (e.g. information loss) on the released database. Another disadvantage of this approach is third party might discover what is hidden by analyzing the overall database and notice unexpected support changes in different portions of the database.

If the whole updated database is sanitized, then the search space of sensitive transactions will include all ten transactions in Figure 1 (a). A possible approach may be removing item "d" from TID7, TID8 and TID10. This will result in 3 item removal again and if the resulting database is mined with a frequent itemset mining algorithm with 10% minimum threshold then there 32 frequent itemsets are found; that means information loss is less. Although sanitizing the whole updated database reduces the side effect on the sanitized database, it increases the execution time and resource allocation.

Figure 1. Motivating example; (a) dynamic transaction database and (b) sensitive itemsets and sensitive thresholds

TID	Transactions	
1	cf	Original Part (D)
2	abe	
3	de	
4	afg	
5	ade	
6	ae	
7	acd	
8	abcde	Incremental Part (d)
9	adfg	
10	acd	

(a)

Sensitive Itemset	Sensitive Threshold
ad	30%
cd	15%
bd	10%

(b)

3. RELATED WORK

Association rule hiding problem was first introduced in (Attalah et al, 1999), the authors proposed heuristic algorithm to conceal the sensitive association rules by decreasing the support of frequent itemsets and also, they proved the NP-hardness of the problem. Since then many different solutions are proposed and these can be classified into 4 categories; 1) Border Based Approaches (Moustakides and Verykios, 2008; Stavropoulos et al, 2016; Sun and Yu, 2005; Sun and Yu, 2007) separate the frequent and non-frequent itemsets with a border and then revise the border to hide the frequent itemsets. During the sanitization process they give less distortion to non-sensitive information when compared to heuristic and reconstruction based approaches but they are unable to identify optimal hiding solutions for some cases although there exists a solution. 2) Exact Approaches (Ayav and Ergenç, 2015; Gkoulalas-Divanis and Verykios, 2006; Gkoulalas-Divanis and Verykios, 2008; Gkoulalas-Divanis and Verykios, 2009; Menon et al., 2005) first formulate the sanitization problem as constraint based satisfaction problem and then apply a linear programming approach to find an optimal solution. Exact approaches manage to find a solution that least modifies the database but because of the nature of linear programming the execution time of exact approaches are seriously higher than other approaches. 3) Heuristic Based Approaches (Keer and Singh, 2012; Oliveria and Zaiane, 2002; Oliveria and Zaiane, 2003; Verykios et al., 2004; Pontikakis et al., 2004; Wu et al., 2007) rely on heuristics and data structures while performing the sanitization process but they may give more side effects when compared to border based and exact approaches as non-sensitive itemsets unintentionally hidden or artificial frequent itemsets unintentionally generated. 4) Reconstruction Based Approches (Boora et al., 2009; Guo, 2007; Lin and Liu, 2007; Mohaisen et al., 2010) first mine the given database to generate the set of frequent itemsets next remove the sensitive itemsets and their supersets from this set and then using this set they generate the sanitized database from scratch. The biggest problem in reconstruction based approaches is putting the non-frequent itemsets into the sanitized database and as a result there may be a big difference between the sanitized and the original database.

Majority of the research focus on heuristic approaches because they are efficient, scalable and have less response time. Table 1 shows the basic attributes of distortion based heuristic association rule hiding or itemset hiding algorithms. The "Algorithm" column of the table gives the name of the algorithm. "Hiding" column gives whether the algorithm is designed for sanitizing sensitive itemset or sensitive association rule. The "Victim Item Selection" column shows the victim item selection criteria of the algorithm where the victim item is the item to be deleted from selected transactions; "Cover" shows the selection of the item is done depending on its cover which is the number of occurrences of the item in different sensitive itemsets, "Support" shows the selection of the item is done depending on its support," Greedy" shows the selection of the item is done in trial and error and "None" shows the algorithm does not select any victim item and deletes the sensitive transaction completely from the database. The "Transaction Selection" column shows the approach of the algorithm in selecting the transaction for modification; "Length" indicates that the algorithm selects the transaction according to its length, "Degree" indicates that the selection is done by considering the number of sensitive itemsets or association rules that are contained by the transaction, "Greedy" indicates that the selection is done in trial and error style. The "Environment" columns show whether the algorithm is designed for incremental or static database. The Sensitive Support threshold indicates if the algorithm allows user to assign multiple sensitive support thresholds or not. "Itemsets/Rules" column shows whether the algorithm is designed for hiding more than one itemset or association rule at each iteration of the algorithm; "Overlap" indicates that the algorithm is designed for sensitive itemsets or sensitive rules sharing common item and "Disjoint" indicates that the algorithm is designed for sensitive itemsets or association rules that do not share any common item.

As it can be seen from Table 1, number of itemset hiding algorithms is more than association rule hiding algorithms. Many algorithms use different heuristics or assigns weights to select victim

Table 1. Summary of existing heuristic based hiding algorithms

Algorithm	Hiding	Victim Item Selection	Transaction Selection	Overlap/ Disjoint	Sensitive Support Thresh.	Environ.
RHID (Jadav et al., 2014)	Rule	Weight	Weight	Overlap	Multiple	Incremental
SPITF (Dai and Chiang, 2010)	Itemset	Degree	Degree			
TTBS (Kuo et al., 2008)	Itemset	Degree	Degree			Static
SWA (Oliveria and Zaiane, 2003)	Itemset	Support	Length			
MDSRRC (Oliveria and Zaiane, 2002)	Rule	Weight	Weight			
HSARWI (Dehkordi and Dehkordi, 2016)	Rule	Weight	Weight		Single	
FHSAR (Weng et al., 2008)	Rule	Degree	Weight			
MICF (Li et al., 2007)	Itemset	Degree	Weight			
Aggregate (Amiri, 2007)	Itemset	None	Greedy			
Disaggregate (Amiri, 2007)	Itemset	Greedy	Greedy			
Hybrid (Amiri, 2007)	Itemset	Greedy	Greedy			
IGA (Oliveria and Zaiane, 2002)	Itemset	Degree	Degree			
RelevanceSorting (Cheng et al., 2016)	Rule	Support	Weight	Disjoint		
EDSR (Norafkan et al., 2015)	Itemset	None	Length			
HRR (Garg et al, 2014)	Rule	Support	All			
SIF-IDF (Hong et al., 2013)	Itemset	Support	Weight			
Algorithm 2.b (Verykios et al., 2004)	Itemset	Support	Length			
Algorithm 2.c (Verykios et al., 2004)	Itemset	None	Length			
PDA (Pontikakis et al., 2004)	Rule	Greedy	Weight			
WDA (Pontikakis et al., 2004)	Rule	None	Weight			
Naïve (Oliveria and Zaiane, 2002)	Rule	All	Degree			
MaxFIA (Oliveria, and Zaiane, 2002)	Itemset	Support	Degree			
MinFIA (Oliveria, and Zaiane, 2002)	Itemset	Support	Degree			

item while some do not select any victim item and remove the whole sensitive transaction from the database. Most of the algorithms assign weight to sensitive transactions, where the weight of a sensitive transaction is calculated depending on some heuristics and these weights are generally calculated by considering the number of sensitive itemsets or rules a transaction contains. Also, most of the studies are designed with the overlapping sensitive itemset or sensitive rule assumption where the terminology overlapping is used for sensitive itemsets or sensitive rules sharing common item.

After a throughout survey of heuristic based itemset/association rule hiding algorithms it is observed that there are only two algorithms which are multiple support threshold based and consider updates on the database. These are SPITF (Dai and Chiang, 2010) and RHID (Jadav et al., 2014) algorithms. SPITF considers all the transactions of the database for sanitization after the updates. This approach reduces side effects as distance and information loss since the search space of victim sensitive transactions is large. However, this algorithm becomes inadequate when the database is dense; it cannot guarantee zero hiding failure. RHID algorithm on the other hand does the sanitization on the update each time. Although this approach is practical, less resource consuming and guarantees zero

hiding failure on all types of databases, the side effects are high since the search space of sensitive transactions is small. Also, this approach can cause unexpected support changes of items in different portions of the database. Therefore, novel and efficient algorithms that provide dynamic itemset hiding based on multiple support thresholds are required.

4. A NOVEL DYNAMIC ITEMSET HIDING ALGORITHM

This section describes the proposed *DynamicPGBS* (Dynamic Pseudo Graph based Sanitization) algorithm and illustrates it by an example. *DynamicPGBS* uses the similar concepts and data structures as PGBS algorithm (Öztürk and Ergenç-Bostanoğlu, 2017). Main difference is the environment; the PGBS algorithm is designed for static database whereas *DynamicPGBS* algorithm is designed for incremental database. Another difference is PGBS puts all transactions into pseudo graph while *DynamicPGBS* puts only the sensitive transactions into pseudo graph in order to reduce the memory allocation.

The first subsection explains how to convert a given transactional database into pseudo graph, the second subsection explains our hiding strategy and the last subsection illustrates our sanitization approach with an example.

4.1. Pseudo Graph

In order to speed up the execution time and minimize the resource allocation Pseudo Graph data structure is used (Öztürk and Ergenç-Bostanoğlu, 2017). A Pseudo Graph (PG) can be represented as PG = (V, E) where V is a set of vertices and E is an ordered set of labelled edges. PG allows both graph loops and multiple edges. Each vertex in PG contains item name (or item number) and each edge represents list of transaction ids that contain the items on the path between starting vertex and its direct successor.

The algorithm for creating pseudo graph of a given transactional database is depicted in Algorithm 1. First each transaction is read one by one from the database D, if the transaction contains any sensitive itemset then it is inserted into the PG.

4.2. DynamicPGBS Algorithm

When a new batch of transaction arrives to the database, the state of a sensitive itemset may automatically change i.e. it may become infrequent, infrequent sensitive itemset may become frequent, or the state of the sensitive itemset may remain the same. The main challenge is hiding sensitive itemsets with minimum execution time, resource allocation and side effects while their states vary. As already explained, in order to speed up the execution time and minimize the resource allocation Pseudo Graph data structure is used as proposed in (Öztürk and Ergenç-Bostanoğlu, 2017). This time, only sensitive transactions are put into the Pseudo Graph structure because the modification of non-sensitive transactions does not affect the support of any sensitive itemset. Another reason for putting only sensitive transactions into PG is continuously increasing database size.

The main properties of our proposed *DynamicPGBS* algorithm are listed as follows i) it is designed for incremental environment and assumes that the sanitization operation is continuous, ii) it handles incremental arrivals, iii) it creates the sanitized database by using an internal data structures, iv) it allows database owner to assign multiple sensitive support thresholds to sensitive itemsets.

DynamicPGBS algorithm keeps the sensitive transactions in a pseudo graph; each increment is added on this data structure. Before sharing the database, our proposed algorithm modifies transactions for hiding the sensitive itemsets on this pseudo graph. This modification operation consists of deleting some items from sensitive transactions until support of every sensitive itemset falls below the corresponding sensitive support threshold by causing as minimum side effect as possible. *DynamicPGBS* algorithm does not modify any transaction in the actual database, it performs the modification operations on the internal data structure, because in an incremental environment the

Algorithm 1. Initialization Process

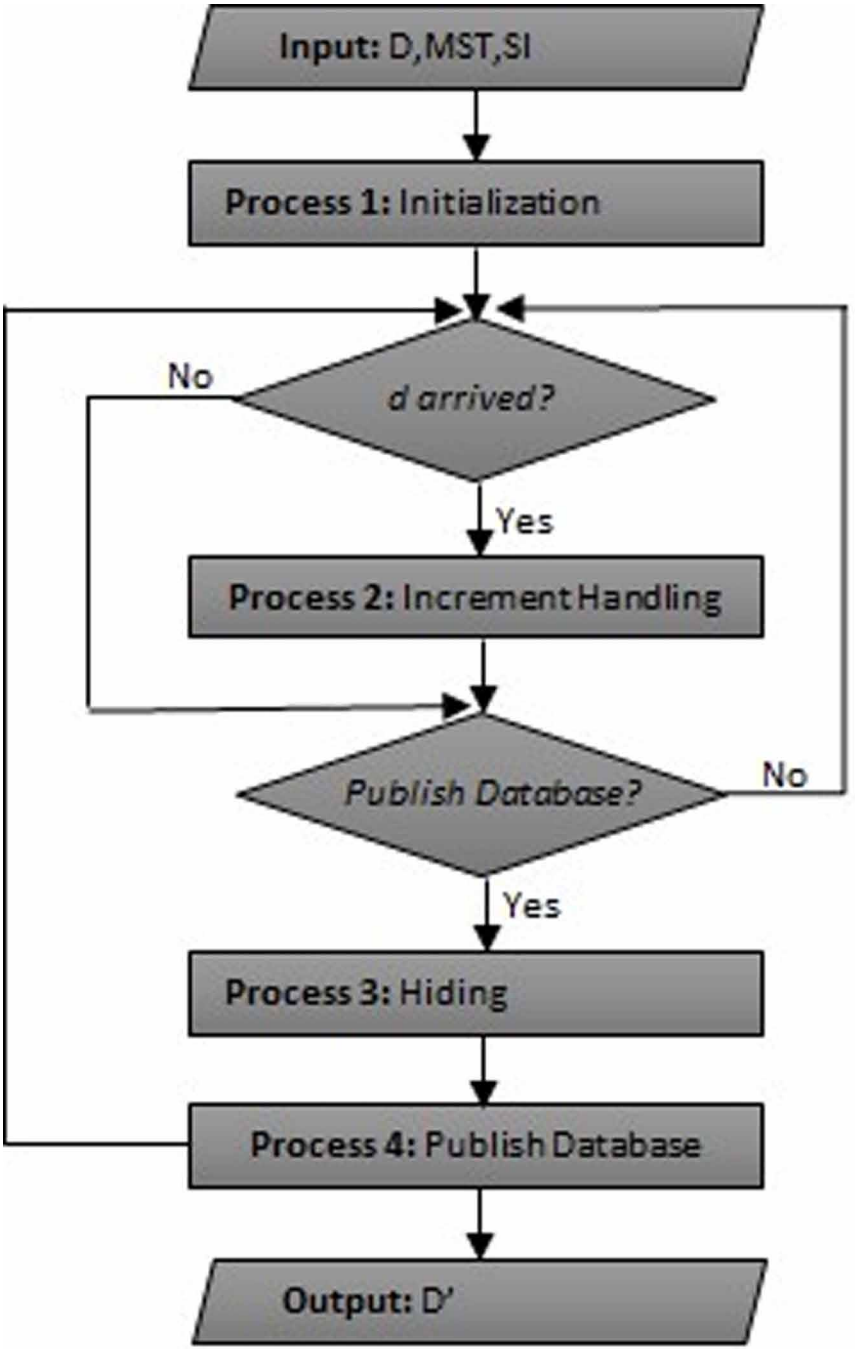
```
Input: Transactional Database D
Output: Pseudo Graph PG
foreach sensitive transaction tr ∈ D do
    foreach item j ∈ tr do
        if PG does not contain v_j then
            create v_j
        end
    end
    foreach item i ∈ tr do
        // if there is more than one item in transaction tr and
        there exist a successor of the item at position i
        if tr[i+1]≠null then
            put an outgoing edge from v_tr[i] to v_tr[i+1] labeled with
             transaction id of tr
        else
              // create a self loop if there is only one item in
            transaction tr
            put an outgoing edge from v_tr[i] to v_tr[i] labeled with
             transaction id of tr
        end
    end
end
```

database is continuously updated and shared many times; original database should be kept as it is. When sharing the database is required, hiding operation is applied on the pseudo graph and summary of delete operations is prepared. With the help of this summary, sanitized database is generated as a copy of the actual database by applying necessary modifications.

The flowchart of *DynamicPGBS* is given in Figure 2. The algorithm takes database D, minimum support thresholds of sensitive itemsets (MST) and sensitive itemsets (SI) as input where SI and the MST are assumed to be defined by the preferences or privacy policies of the database owner. Initialization Process prepares internal data structure. After the initialization process, if a new batch of transactions (d) arrives then the algorithm performs the Increment Handling process which updates the database and the internal data structure. Either after the Initialization process or Increment Handling process if the database owner wants to release the database, the Hiding Process is performed. Hiding process defines the transactions to be modified and items to be deleted from these transactions. The objective here is to prepare summary of delete operations. After the Hiding Process is finished the database owner can share the database with the Publish Database process. This process creates a copy of the original database by making required modifications defined by Hiding Process. This copy is the sanitized database D' in which sensitive itemsets are infrequent, in other words they are hidden. Another operation performed by this process is to restore the internal data structure to the state prior the sanitization process in order to be ready to accept a new increment and to be compliant with the original database.

There are three important internal data structures used by *DynamicPGBS* algorithm; Pseudo Graph (PG), the Sensitive Count Table (SCT) and the Sanitization Table (ST). PG keeps all sensitive

Figure 2. Flowchart of DynamicPGBS



transactions. Second data structure SCT keeps the number of necessary modifications (N_{Modify}) related with the sensitive itemsets in order to reduce their actual supports below the given user defined sensitive thresholds. Equation (1) shows how to calculate N_{Modify} related with each sensitive itemset

in SCT. Base floor is used for the difference in the equation since support count calculations derived from support percentages may yield decimal numbers.

$$N_{Modify} = \lfloor actual\ support\ count\ of\ itemset - sensitive\ threshold\ count + 1 \rfloor \quad (1)$$

As an example, in Table 2, N_{Modify} for sensitive itemset "ad" is 3; it means that if we delete either item "a" or item "d" 3 times, we reduce actual support count of itemset "ad" below the given sensitive threshold. If N_{Modity} value of any sensitive itemset is zero or less than zero, this implies that the given sensitive itemset is hidden in the database.

The third data structure ST keeps the modification information which will be applied to the database before it is released. The ST keeps the pairs of victim items and transactions, where the victim field shows which item will be deleted and the transactions field shows the ids of transactions that will be modified. As an example, if victim item "d" in Table 3 is removed from transactions 8, 7, 10; it means all sensitive itemsets will be hidden. The objective of the Hiding Process is to prepare ST.

The DynamicPGBS consists of four major processes as seen in Figure 2; Initialization, Increment Handling, Hiding and Publish Database. Initialization process puts all sensitive transactions in dataset D to PG and Algorithm 1 presents this process. Increment handling process adds all transactions in increment d to D and then adds all sensitive transactions in d to PG. The Hiding process creates the Sanitization Table and it is depicted in Algorithm 2. The Publish Database process first prepares D' as a copy of D by deleting each item from its corresponding transaction given in ST and then restores all removed items on PG. The restore operation is performed by putting each victim and transactions pairs stored in ST into the PG again.

In Algorithm 2 through Steps 1 to 2 first SCT is created by calculating N_{Modify} of each sensitive itemset and then sorted in decreasing order of N_{Modify}. In Step 3 cover degree of each item is calculated by counting occurrences of items in all sensitive itemsets whose N_{Modify} value is greater than zero. Whenever N_{Modify} of a record in SCT becomes less than or equal to zero, it indicates that the corresponding sensitive itemset in SCT is already sanitized. In Step 4 the sanitization operation starts from the first row r1 of SCT and keeps selecting victim items and corresponding transactions till N_{Modify} of all records in SCT become less than or equal zero. The first row r1 of SCT is selected to be sanitized first, because it keeps the sensitive itemset that needs more support decrease than others. In Step 5 the victim item is selected among items in r1.SI that has the maximum cover degree, if there is more than one victim item having the same cover degree the victim item is selected with the highest support in PG. Finally, if there is still more than one victim item, a random item is selected. The variable USI (Unified Sensitive Itemsets) is the unification of all sensitive itemsets of SCT whose

Table 2. Sensitive count table (SCT)

SI	Actual Support Count	Sensitive Threshold	Sensitive Threshold Count	N_{Modify}
ad	5	30%	3	3
cd	3	15%	1	2
bd	1	10%	1	1

Table 3. Sanitization table (ST)

Victim	Transactions
d	**8, 7, 10**

N_{Modify} value is greater than zero and contain the victim item. The USI is generated in Step 6 and then maximum r1.N_{Modify} number of transaction ids containing USI is uncovered in Step 7. In Step 8 the victim is deleted from these transactions on PG and then in Step 9 these victim and transaction id pairs are added to the Sanitization Table (ST). Through Steps 10 to 18, if any sensitive itemset is a subset of USI then its N_{Modify} is decreased by the number of transaction ids uncovered and then sensitive itemsets whose N_{Modify} became less than or equal to zero are removed from the USI. In Step 19 if the sensitive itemset in the first row of SCT is still not sanitized then the algorithm tries to find different transactions by changing the USI with removing the sensitive itemset having the least $N_{Modify.}$

When computational complexity of the Initialization process is analyzed it is seen that first the database is scanned in order to find sensitive transactions, time complexity of this Step is O(|D|) where |D| is the database size. During this scan each sensitive transaction is put into PG, the cost of this O(|V|) where |V| is the total number of vertices, in other word number of distinct items in the database. As a result, scanning the database to find sensitive transactions and putting them into PG brings total O(|D| * |V|) computational complexity. Hiding process on the other hand, uncovers transactions from PG in O(|V|) time. That is repeated as the number of victim items so at worst case computational complexity is O(|SI|*|V|) where |SI| is the number of items in all sensitive items assuming that there is no overlapping item.

4.3. Illustrating Example

Suppose the database given in Figure 1 (a) and sensitive itemsets with their sensitive thresholds as in Figure 1 (b) are given. First the Initialization Process puts all sensitive transactions into PG as shown in Figure 3 (a). Whenever an increment arrives, Increment Handling process updates D and PG, with the example the updated PG is shown in Figure 3 (b).

Let us continue by explaining Hiding Process; the algorithm uses the PG to create the Sensitive Count Table (SCT) with calculated N_{Modify} values (Step 1) as shown in Table 2, Cover degree of each item stored in SCT is calculated as a:1, b:1, c:1 and d:3 (Step 3; a, b and c is included in 1 sensitive itemset whereas d is included by 3 of the sensitive itemsets). The first row in SCT stores the sensitive itemset "ad" so the victim is selected among candidate victims "a" and "d" where "d" is selected as victim item because it has the maximum cover degree (Step 5). Then the algorithm unifies 3 of the sensitive itemsets because each of them contains the victim item "d". The unified sensitive itemset is "abcd"(Step 6) and according to Figure 3 (b) only the transaction 8 contains "abcd" (Step 7). So the item "d" is removed from 8^{th} transaction in PG as shown in Figure 3 (c) (Step 8), the victim item "d" and transaction 8 is added into the ST (Step 9) and N_{Modify} value of each sensitive itemset in SCT is decreased by 1. N_{Modify} value of the sensitive itemset "bd" becomes zero so the cover degree of items are updated as a:1, c:1 and d:2 (Steps 10-18). The algorithm selects new victim item, again the item "d" is selected because it still has the maximum cover degree, also the sensitive itemsets "ad" and "cd" are unifed because they both contain the item "d". According to Figure 3 (c) transactions 7 and 10 contain the unified itemset "acd". The victim "d" is removed from transactions 7 and 10 as shown in Figure 3 (d) and these victim item and transaction pairs are added to the ST. The new N_{Modify} value of "ad" becomes 0 and "cd" becomes -1 so the hiding process terminates.

After the Hiding Process is finished the Publish Database Process prepares D' by deleting each victim item and transactions pair stored in ST from D. The resulting ST for this example is given in Table 3 and according to this table the sanitized database D' is given in Figure 4. After having sanitized dataset D' ready, PG is recovered by again using the ST and then all records in ST are deleted. Now the system is ready to accept new increments.

5. PERFORMANCE EVALUATION

In this section the performance of *DynamicPGBS* is compared with SPITF (Dai and Chiang, 2010) and RHID (Jadav et al., 2014) algorithms. As *DynamicPGBS* both SPITF and RHID algorithms are

Algorithm 2. Hiding Process

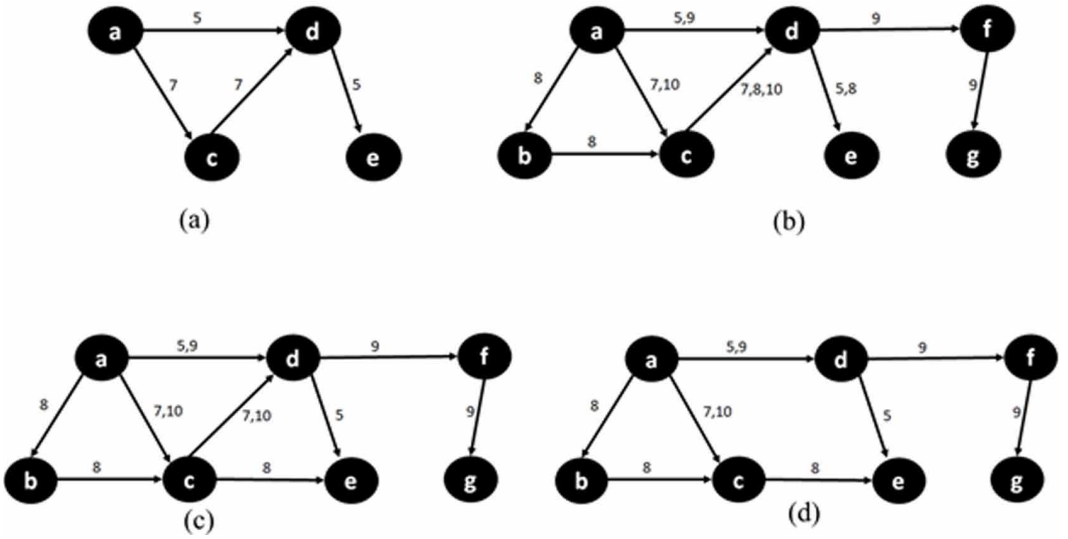
Input: Pseudo Graph PG, Sensitive Itemsets SI, Sensitive Thresholds
Output: Sanitization Table ST, Support Count Table SCT
1 Calculate N_{Modify} for each si ∈ SI and put them into SCT
2 Sort SCT in decreasing order of N_{Modify}
3 Calculate cover degree of items in SI of SCT
4 **while** *N_{Modify} value of the first row r1 > 0* **do**
5 **victim**←item that has the maximum cover degree in r1.SI or has the maximum support or selected randomly
6 **USI**← USI ∪ SI for all SI ∈ SCT where $N_{Modify} > 0$ and victim ⊆ SI
7 **transactions**←transaction ids containing USI in PG *// number of transactions < r1.N_{Modify}*
8 Update PG by removing *victim* from **transactions**
9 Insert **victim** and **transactions** pairs into ST
10 **foreach** *row r ∈ SCT* **do**
11 **if** *r.SI ⊆ USI* **then**
12 r.N_{Modify}←r.N_{Modify} - |transactions|
13 **end**
14 **if** *r.N_{Modify}≤0* **then**
15 USI ← USI - r.SI
16 Recalculate cover degree of items
17 **end**
18 **end**
19 **if** *r1.N_{Modify} > 0* **then**
20 Remove the sensitive itemset from USI which has the least N_{Modify} and go back to step 7
21 **else**
22 Sort SCT in decreasing order of N_{Modify}
23 **end**
24 **end**

designed for incremental environment. They differ in their sanitization methodologies. The SPITF performs the sanitization operation on the whole database after the incremental part has been added whereas the RHID algorithm performs the sanitization process on the incremental part and then combines it with the sanitized original database. Both SPITF and RHID enable the user to assign different sensitive thresholds to each sensitive itemset as *DynamicPGBS*.

All the experiments are conducted on a computer with Intel core i7-5500 2.4 GHZ processor and 8GB of RAM running on a Windows 10 operating system. Execution time includes I/O and CPU time. During performance evaluation, it is ensured that the system state is similar for all test runs and they give close results when repeated.

Our experiments are performed on 4 real databases; Chess, Connect and Mushroom and Retail where first three is obtained from UCI Repository (Blake and Merz, 1998) and last one is retrieved from (Brijs et al., 1999). In addition to 4 real databases 2 synthetic databases are used. The synthetic

Figure 3. Pseudo Graph (PG) with the motivating example; (a) PG of the sensitive transactions in the original part of the database; (b) PG of the sensitive transactions in the whole updated database; (c) PG after item "d" is removed from transaction 8; (d) PG after item "d" is removed from transactions 7 and 10



databases with different characteristics are generated by using the IBM quest dataset generator (Bhalodiya, 2014). The characteristics of all databases in terms of number of transactions, number of distinct items, average transaction length, shortest and longest transaction length and density are given in Table 4. The density of a database is the average transaction length divided by number of distinct items. Density of a database shows whether it is dense or sparse. As indicated in (Bayardo et al., 1999; Gkoulalas and Divanis, 2009; Han et al., 2000; Pei et al., 2000) frequent itemsets generated from dense databases may be long. Databases with different densities allow the observation of the performance of the algorithm on dense and sparse databases and with short and long frequent itemsets.

To assess the performance of *DynamicPGBS* 10 sensitive itemsets are randomly selected and assigned them multiple sensitive thresholds for each database. To select the sensitive itemsets and assign them different sensitive support thresholds the databases are partitioned into 5 bins where the

Figure 4. Sanitized database D'

TID	Transactions
1	cf
2	abe
3	de
4	afg
5	ade
6	ae
7	ac
8	abce
9	adfg
10	ac

Table 4. Characteristics of experimental databases

Database Name	Number of Transactions	Distinct Items	Average Length	Shortest Length	Longest Length	Density (%)
Chess	3,196	75	37	38	38	49.4
Connect	67,557	129	43	43	43	33.4
Mushroom	8,124	119	23	20	24	19.4
Retail	88,162	16,470	10.3	2	77	0.0625
SyntheticDense	29,166	99	43.09	2	44	43.5
SyntheticSparse	28,417	9,479	11.48	2	11	0.1212

bins contain nearly equal number of itemsets. Then 2 itemsets are randomly selected from each bin and assigned the minimum support threshold as the minimum support given in the support range of the corresponding bin. The support ranges of the bins for each database are given in Table 5.

Main objective of the performance evaluation is to observe the impact of the increment size on the performance of the algorithms. The algorithms are compared for 10 increment sizes in the range of 10% to 100%. The performance of all algorithms is measured with respect to execution time, side effects and total memory consumption. As execution time, the time needed by each algorithm to perform the sanitization, as side effects hiding failure, distance and information loss caused by the sanitization algorithm and as total memory consumption the total memory allocated by each algorithm to perform the sanitization process are considered.

During the tests it is noticed that SPITF algorithm cannot guarantee zero hiding failure on dense databases. SPITF algorithm gives zero hiding failure in sparse databases like Retail and SyntheticSparse whereas it fails to hide some sensitive itemsets in dense databases as Chess, Connect, Mushroom and SyntheticDense. The hiding failure of *DynamicPGBS* and RHID algorithms is zero in all given six databases, which means these two algorithms successfully hide all given sensitive itemsets. Hiding failure results of SPITF algorithm for dense databases are given in Figure 5.

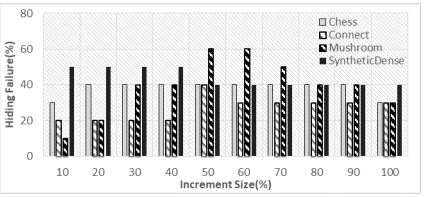
5.1. Execution Time

In Figure 6 execution time of *DynamicPGBS*, SPITF and RHID are demonstrated. Execution time of *DynamicPGBS* and SPITF does not change explicitly when new transactions are added on sparse databases as in Figure 6 (c) and (e) which is not the case in RHID algorithm. This is reasonable because sparse databases produce many short frequent itemsets and the sensitive itemsets for each database are selected randomly from the frequent itemsets, so this makes the size of the sensitive

Table 5. Support ranges of the databases

	Chess	Connect	Mushroom	Retail	Synthetic Dense	Synthetic Sparse
Bin	**Support Range (%)**	**Support Range (%)**	**Support Range (%)**	**Support Range (%)**	**Support Range (%)**	**Support Range (%)**
1	(60.01, 61.36]	(85, 85.7]	(11, 12.18]	(0.1, 0.118]	(30, 30.8]	(0.5, 0.544]
2	(61.36, 63.08]	(85.76, 86.72]	(12.18, 13.88]	(0.12, 0.142]	(30.8, 31.85]	(0.544, 0.6]
3	(63.08, 65.55]	(86.73, 87.92]	(13.88, 15.40]	(0.144, 0.185]	(31.85, 33.39]	(0.6, 0.709]
4	(65.55, 69.74]	(87.93, 89.85]	(15.40, 20.53]	(0.186, 0.287]	(33.39, 36.19]	(0.709, 0.935]
5	(69.74, 99.62]	(89.86, 99.87]	(20.53, 100]	(0.288, 5.072]	(36.19, 95.46]	(0.935, 3.8]

Figure 5. Hiding failure of SPITF algorithm



itemsets chosen for sparse databases shorter than dense databases. As the size of a sensitive itemset decreases, the execution time to search and modify a given sensitive itemset in the data structures of both *DynamicPGBS* and SPITF decreases. The execution time of *DynamicPBGS* is less than SPITF and RHID in all databases and the RHID algorithm has the worst running time on Connect, Retail, SytheticSparse and SyntheticDense databases although it only sanitizes the incremental part, so it can be seen that the data structure of both SPITF and *DynamicPGBS* decreases the execution time. The execution time of RHID algorithm is better than SPITF when the database is dense and contains small number of transactions where this situation can easily be seen on Chess database (Figure 6 (d)).

5.2. Distance

The experimental results of the distance experiment show that the best distance result achieved by SPITF algorithm is in only dense and large database Connect (Figure 7). This may be due to fact that SPITF is unable to hide all sensitive itemsets on dense databases and as a result less than necessary number of transaction modification causes less item removal. *DynamicPGBS* achieves the best distance result in sparse databases Retail and SyntheticSparse and the RHID algorithm achieves the best distance in dense databases Chess and SyntheticDense databases.

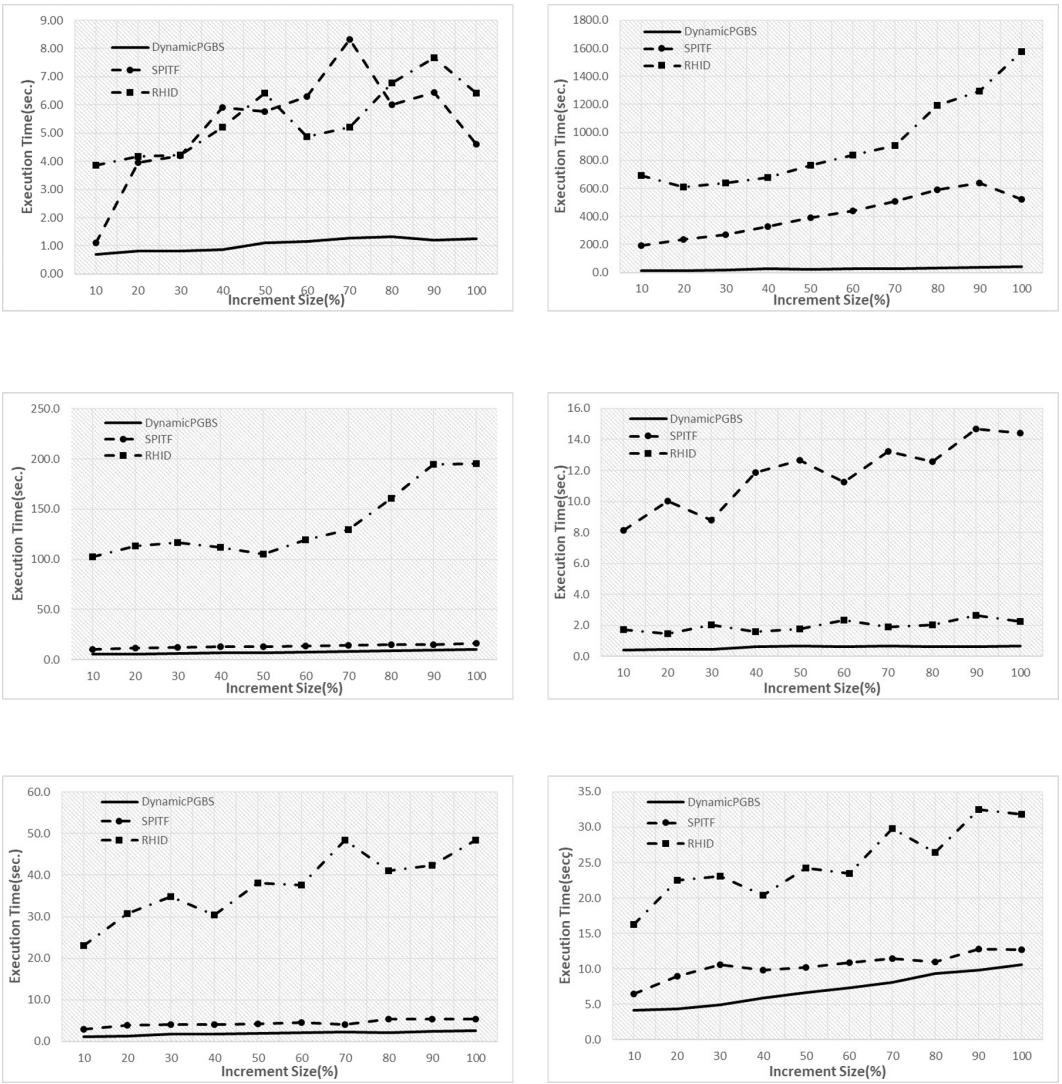
5.3. Information Loss

The experiment results of Information Loss are given in Figure 8. The results show that SPITF causes less Information Loss than *DynamicPGBS* and RHID on dense databases like Chess, Connect and SyntheticDense because it is unable to hide 2 of the sensitive itemsets on these databases. On Mushroom database the *DynamicPGBS* causes minimum information loss when the increment sizes are 10%, 20%, 30%, 90% and 100% and between 40% and 80% the SPITF algorithm has the minimum value. The results for Mushroom depend on the increment size because the density of this database smaller than other dense databases. On sparse databases like Retail and SyntheticSparse the *DynamicPGBS* algorithm gives less information loss than the SPITF and RHID algorithms.

5.4. Memory Requirement

Figure 9 shows the total memory consumption in megabytes (MB). The RHID algorithm consumes the minimum amount of memory on dense databases and also its total memory consumption increases linearly with the increment size for all databases. This is because the average transaction lengths of dense databases (Chess, Connect, Mushroom and SyntheticDense) are greater than sparse databases (Retail and SyntheticSparse). As the average transaction length increases the memory consumption of data structures of both SPITF and *DynamicPGBS* algorithms increase if the arriving batch of transactions are different than the transactions previously added, whereas the RHID algorithm does not use any data structure to store the transactions the similarity of transactions in the original part and incremental part does not affect the memory consumption. The *DynamicPGBS* algorithm consumes the minimum amount of memory as in Figure 9 (c) and (e) when the databases are sparse, this is due to small average transaction length. Also in all databases the SPITF algorithm consumes the highest amount of memory because of its tree like data structure. In this data structure an item can be represented as node many times but this is not the case in *DynamicPGBS*'s graph data structure.

Figure 6. Execution time of sanitization process; (a) Mushroom Database; (b) Connect Database; (c) Retail Database; (d) Chess Database; (e) SyntheticSparse Database; (f) SyntheticDense Database

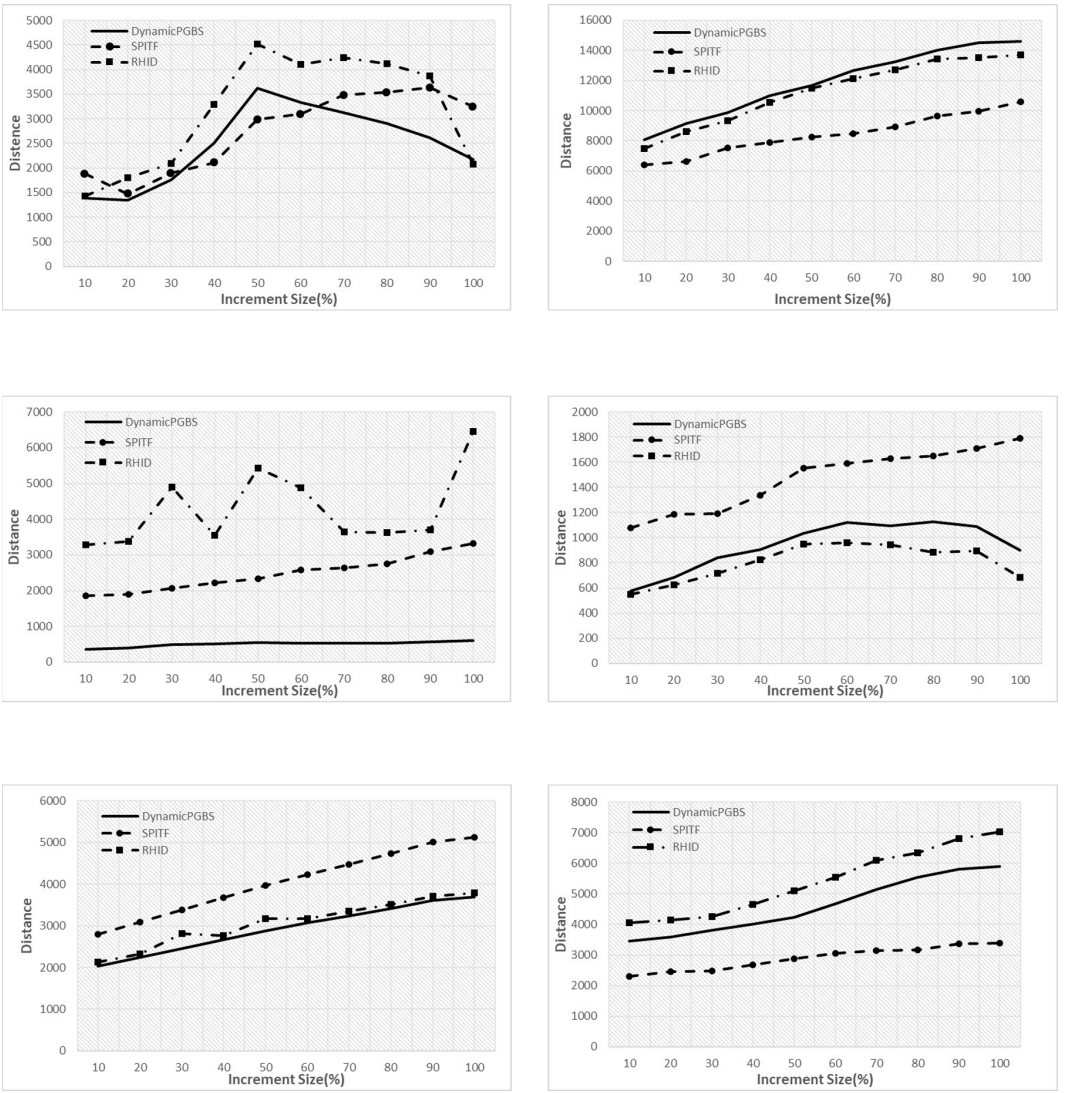


5.5. Discussion on the Results

The Pseudo Graph data structure used in *DynamicPGBS* has significant improvement on the execution time for each database used in the performance evaluation. On sparse databases the *DynamicPGBS* has the minimum distance and information loss value whereas it has the second-best results on dense databases. The SPITF algorithm seems to be good in distance and information loss on dense databases but the most important drawback of this algorithm is, it causes hiding failure on the resulting sanitized databases when the density of the databases is high.

The memory consumption of *DynamicPGBS* is proportional to density and the size of the given database. The total memory consumption of RHID algorithm is better than *DynamicPGBS* but this memory allocation comes with a tradeoff of higher execution time.

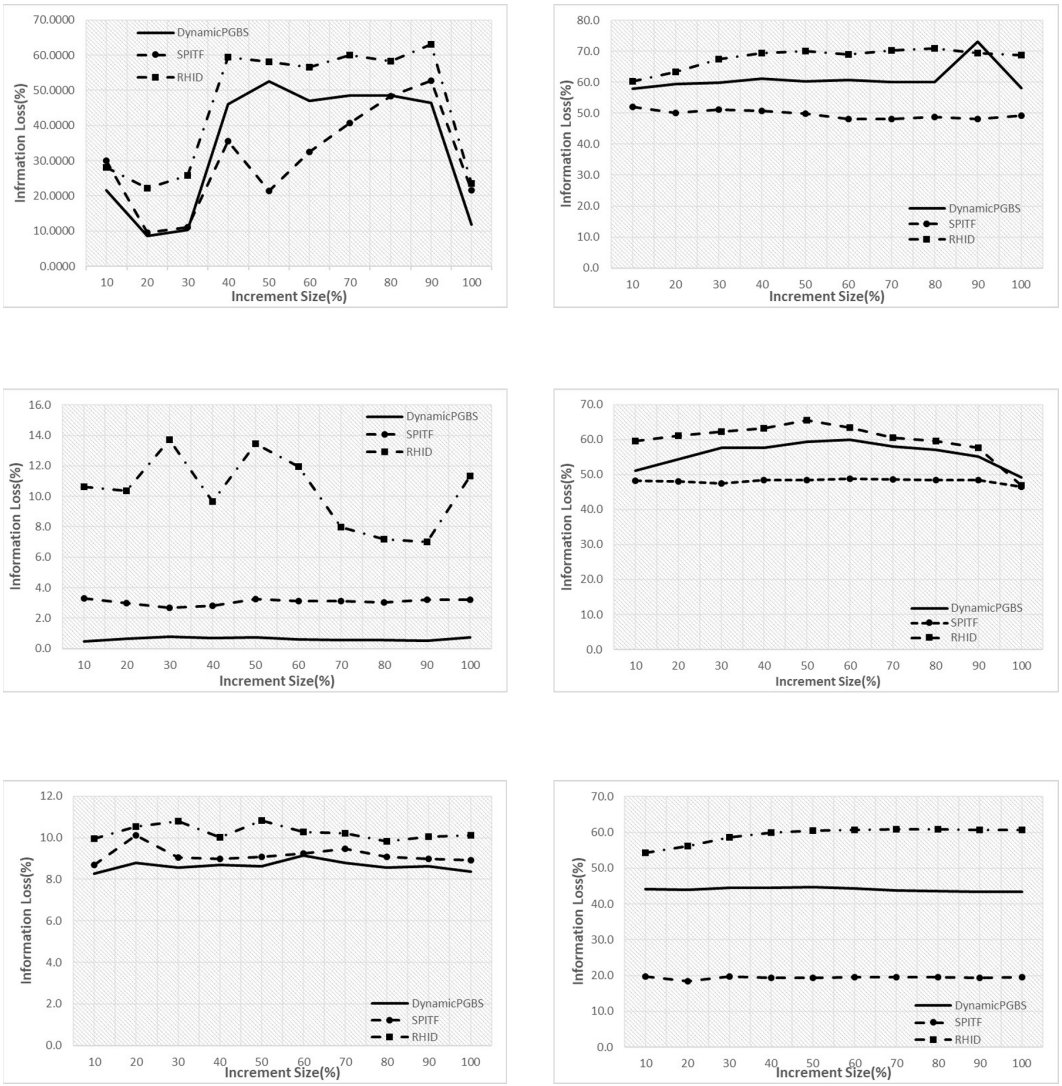
Figure 7. Number of items removed during sanitization process; (a) Mushroom Database; (b) Connect Database; (c) Retail Database; (d) Chess Database; (e) SyntheticSparse Database; (f) SyntheticDense Database



6. CONCLUSION

In this paper a new distortion based dynamic heuristic itemset hiding algorithm called *DynamicPGBS* for the problem of itemset hiding on updated databases is proposed. The *DynamicPGBS* algorithm is based on the hiding algorithm in (Öztürk and Ergenç-Bostanoğlu, 2017). Besides concealing the sensitive itemsets, the *DynamicPGBS* is designed for the challenges such as hiding sensitive itemsets under multiple sensitive support thresholds and dealing with the incremental environment. All sensitive transactions in the whole updated database are represented as Pseudo Graph structure thus scanning the sensitive transactions and modifying them become much easier than performing these operations on the actual database. After a sanitization operation *DynamicPGBS* restores all transaction modifications on this pseudo graph. Thus, it still utilizes all possible sensitive transaction modifications in the next sanitization operation.

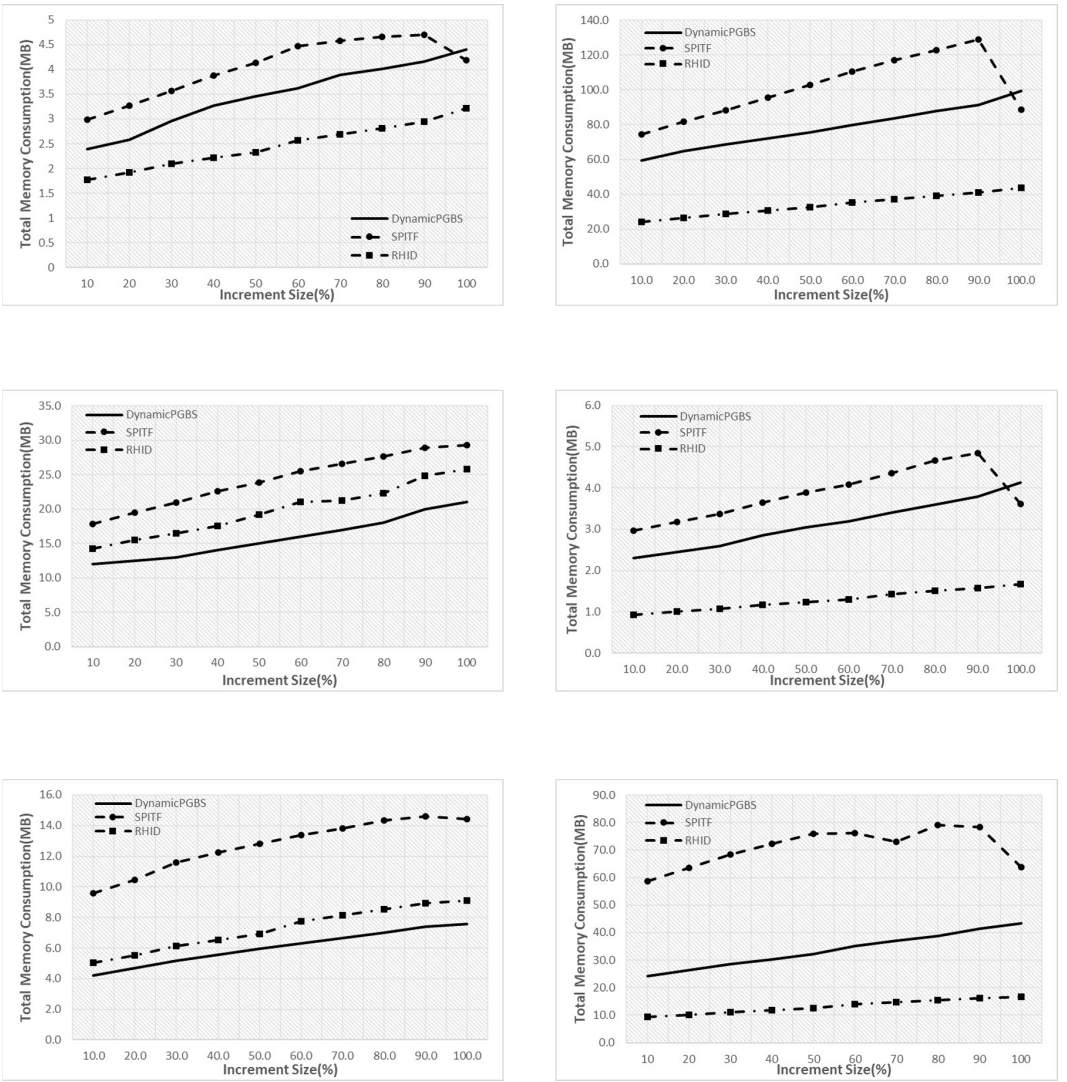
Figure 8. Information Loss during sanitization process; (a) Mushroom Database; (b) Connect Database; (c) Retail Database; (d) Chess Database; (e) SyntheticSparse Database; (f) SyntheticDense Database



The experimental results show that *DynamicPGBS* can achieve reasonable results when compared to other dynamic algorithms in the literature. Especially on sparse databases the *DynamicPGBS* achieves the best performance in terms of execution time, distance, information loss and total memory allocation. On the other hand, on dense databases the *DynamicPGBS* achieves the best performance in terms of execution time. Although the *DynamicPGBS* has the second-best results in term of distance, information loss and memory allocation on dense databases, it guarantees zero hiding failure.

As future work it is planned to; i) extend the algorithm to handle the deletions in the databases as well, ii) propose a new version of hiding process where degree of victim transactions will also be considered.

Figure 9. Total memory consumption during sanitization process; (a) Mushroom Database; (b) Connect Database; (c) Retail Database; (d) Chess Database; (e) SyntheticSparse Database; (f) SyntheticDense Database



ACKNOWLEDGMENT

This work is partially supported by The Scientific and Technological Research Council of Turkey (TUBITAK), under ARDEB 3501 Project No: 114E779.

REFERENCES

Agrawal, R., Agarwal, C. C., & Prasad, V. V. V. Depth-first generation of long patterns. In *ACM KDD Conference*, (pp. 108-118).

Agrawal, R., Imilinski, T., & Swami, A. (1996). Mining association rules between sets of items in large databases. In *Proceedings of the International Conference on Management of Data* (pp. 207-216). Washington DC.

Agrawal, R., & Srikant, R. (1994). Fast algorithms for mining association rules. In *Proceedings of the 20th International Conference on Very Large Databases*, San Francisco, CA (pp.487-499).

Amiri, A. (2007). Dare to share: Protecting sensitive knowledge with data sanitization. *Decision Support Systems*, *43*(1), 181–191. doi:10.1016/j.dss.2006.08.007

Ashrafi, M. Z., Taniar, D., & Smith, K. A. (2007). Redundant association rules reduction techniques. *International Journal of Business Intelligence and Data Mining*, *2*(1), 29–63. doi:10.1504/IJBIDM.2007.012945

Atallah, M., Bertino, E., Elmagarmid, A., Ibrahim, M., & Verykios, V. S. (1999). Disclosure limitation of sensitive rules. In *Workshop on Knowledge and Data Engineering Exchange* (pp. 45-52).

Ayav, T., & Ergenç, B. (2015). Full Exact approach for itemset hiding. *International Journal of Data Warehousing and Mining*, *11*(4), 49–63. doi:10.4018/ijdwm.2015100103

Bastide, Y., Pasquier, N., Taouil, R., Stumme, G., & Lakhal, L. (2000). Mining Minimal Non-redundant Association Rules Using Frequent Closed Itemsets. In Proceedings of the *1st International Conference of Computational Logic*. doi:10.1007/3-540-44957-4_65

Bayardo, R. J. Jr, Agrawal, R., & Gunopulos, D. (1999). Constraint based rule mining on large, dense data sets. *Data Mining and Knowledge Discovery*, *4*(2/3), 217–240. doi:10.1023/A:1009895914772

Bhalodiya, D.J. (2014). IBM Quest Market-Basket Synthetic Data Generator.

Blake, C. L., & Merz, C. J. (1998). *UCI Repository of Machine Learning Databases. University of California*. Irvine: Dept. of Information and Computer Sciences.

Bodon, F. (2003). A fast APRIORI implementation. In Workshop Frequent Itemset Mining Implementations (FIMI'03) (Vol. 90, pp. 56-65).

Boora, R. K., Shukla, R., & Misra, A. (2009). An improved approach to high level privacy preserving itemset mining. *International Journal of Computer Science and Information Security*, *6*(3), 216–223.

Brijs, T., Swinnen, G., Vanhoof, K., & Wets, G. (1999). The use of association rules for product assortment decisions: A case study. In *Proceedings of the 5th International Conference on Knowledge Discovery and Data Mining*, San Diego, CA (pp. 254–260).

Cheng, P., Roddick, J. F., Chu, S. C., & Lin, C.-W. (2016). Privacy preservation through a greedy, distortion-based rule hiding method. *Applied Intelligence*, *44*(2), 295–306. doi:10.1007/s10489-015-0671-0

Clifton, C., & Marks, D. (1996). Security and privacy implication of data mining. In *ACM SIGMOD Workshop on Data Mining and Knowledge Discovery* (pp. 15–19).

Dai, B. R., & Chiang, L. H. (2010). Hiding frequent patterns in the updated database. In *Proceedings of the International Conference on Information Science and Applications (ICISA)*. doi:10.1109/ICISA.2010.5480385

Daly, O., & Taniar, D. (2004) Exception Rules Mining Based on Negative Association Rules. In *Proceedings of the International Conference on Computational Science and Its Applications* (pp. 543-552). doi:10.1007/978-3-540-24768-5_58

Darrab, S., & Ergenc, B. (2017). Vertical Pattern Mining Algorithm for Multiple Support Thresholds. In *Proceedings of the International Conference on Knowledge Based and Intelligent Information and Engineering Systems*, Marseille, France, September 6-8 (pp. 417-426). doi:10.1016/j.procs.2017.08.051

Dehkordi, M. S., & Dehkordi, M. N. (2016). Introducing an algorithm for use to hide sensitive association rules through perturbation technique. *Journal of Artificial Intelligence and Data Mining*. doi:10.5829/-IDOSI.JAIDM.2016.04.02.10

Garg, V., Singh, A., & Singh, D. (2014). A hybrid algorithm for association rule hiding using representative rule. *International Journal of Computers and Applications*, *97*. doi:10.1007/978-3-319-07455-9_9

Gkoulalas-Divanis, A., & Verykios, V. S. (2006). An integer programming approach for frequent itemset hiding. In *Proceedings of the 15th ACM International Conference on Information and Knowledge Management*. doi:10.1145/1183614.1183721

Gkoulalas-Divanis, A., & Verykios, V. S. (2008). A parallelization framework for exact knowledge hiding in transactional databases. In IFIP International Federation for Information Processing (Vol. 278, pp. 349-363). doi:10.1007/978-0-387-09699-5_23

Gkoulalas-Divanis, A., & Verykios, V. S. (2009). Hiding sensitive knowledge without side effects. *Knowledge and Information Systems*, *20*(3), 263–299. doi:10.1007/s10115-008-0178-7

Guo, Y. (2007). Reconstruction based association rule hiding. In *SIGMOD Ph.D. Workshop on Innovative Database Research*. Retrieved from http://www.borgelt.net/apriori.html

Han, J., Pei, J., & Yin, Y. (2000). Mining frequent patterns without candidate generation. *In ACM SIGMOD International Conference on Management of Data*.

Hong, T.-P., Lin, C.-W., Yang, K.-T., & Wang, S.-L. (2013). Using tf-idf to hide sensitive itemsets. *Applied Intelligence*, *38*(4), 502–510. doi:10.1007/s10489-012-0377-5

Imilinski, T., & Swami, A. (1996). Mining association rules between sets of items in large databases. In *Proceedings of the International Conference on Management of Data*, Washington DC (pp. 207-216).

Jadav, K.B., Vania, J., Patel, D.R. (2014). Efficient hiding of sensitive association rules for incremental datasets. International Journal of Innovations & Advancement in Computer Science IJIACS, ISSN 2347 – 8616, 3(4).

Keer, S., & Singh, A. (2012). Hiding sensitive association rule using clusters of sensitive association rule. *International Journal of Computer Science and Network*, *1*(3).

Kiran, R., & Reddy, P. (2011). Novel techniques to reduce search space in multiple minimum supports based-frequent pattern mining algorithms. In *Proceedings of the International Conference on Extending Database Technology* (pp. 11-20). doi:10.1145/1951365.1951370

Kuo, Y., Lin, P. Y., & Dai, B. R. (2008). Hiding frequent patterns under multiple sensitive thresholds. In *Database and Expert Systems Applications* (pp. 5–18). DEXA; doi:10.1007/978-3-540-85654-2_2

Li, Y. C., Yeh, J. S., & Chang, C. C. (2007). MICF: An effective sanitization algorithm for hiding sensitive patterns on data mining. *Advanced Engineering Informatics*, *21*(3), 269–280. doi:10.1016/j.aei.2006.12.003

Lin, J., & Liu, J. Y. C. (2007). Privacy preserving itemset mining through fake transactions. In *Proceedings of the 22nd ACM Symposium on Applied Computing*, Seoul, Korea (pp. 375–379). doi:10.1145-/ doi:10.1145/1244002.1244092

Menon, S., Sarkar, S., & Mukherjee, S. (2005). Maximizing accuracy of shared databases when concealing sensitive patterns. *Information Systems Research*, *16*(3), 256–270. doi:10.1287/isre.1050.0056

Mohaisen, A., Jho, N., Hong, D., & Nyang, D. (2010). Privacy preserving association rule mining revisited: Privacy enhancement and resource efficiency. *IEICE Transactions on Information and Systems*, *93*(2), 315–325. doi:10.1587/transinf.E93.D.315

Moustakides, G. V., & Verykios, V. S. (2008). A maxmin approach for hiding frequent itemsets. *Data & Knowledge Engineering*, *65*(1), 75–89. doi:10.1016/j.datak.2007.06.012

Nourafkan, M., Rastegari, H., Dehkordi, M.N. (2015). An algorithm for hiding sensitive frequent itemsets. *International Journal of Advances in Soft Computing and its Applications*, *7*(1).

Oliveira, S. R. M., & Zaiane, O. R. (2002). Privacy preserving frequent itemset mining. In *Proceedings of the International Conference on Data Mining (ICDM)*, Maebashi City, Japan (pp. 43-54).

Oliveira, S. R. M., & Zaiane, O. R. (2003). Algorithms for balancing privacy and knowledge discovery in association rule mining. In *Proceedings of the 7th International Database Engineering & Applications Symposium* (pp. 54–63). doi:10.1109/IDEAS.2003.1214911

Öztürk, A. C., & Ergenç-Bostanoğlu, B. (2017) Itemset hiding under multiple sensitive support thresholds. In *Proceedings of the 9th International Joint Conference Knowledge Engineering and Knowledge Management (KMIS)* (pp. 222-231), Madeira, Portugal.

Pei, J., Han, J., & Mao, R. (2000). CLOSET: An efficient algorithms for mining frequent closed itemsets. In *Proceedings of the DMKD Workshop*.

Pei, J., Han, J., & Mao, R. (2000). CLOSET: An efficient algorithm for mining frequent closed itemsets. In *Proceedings of the ACM-SIGMOD Int. Workshop Data Mining and Knowledge Discovery* (pp. 11–20).

Pontikakis, E. D., Tsitsonis, A. A., & Verykios, V. S. (2004). An experimental study of distortion-based techniques for association rule hiding. In *Proceedings of the 8th Annual Conference on Data and Applications Security*, Catalonia, Spain. doi:10.1007/1-4020-8128-6_22

Stavropoulos, E. C., Verykios, V. S., & Kagklis, V. (2016). A transversal hypergraph approach for the frequent itemset hiding problem. *Knowledge and Information Systems*, *47*(3), 625–645. doi:10.1007/s10115-015-0862-3

Sun, X., & Yu, P. S. (2005). A border-based approach for hiding sensitive frequent itemsets. In *Proceedings of the 5th IEEE International Conference on Data Mining (ICDM)* (pp. 426-433).

Sun, X., & Yu, P. S. (2007). Hiding sensitive frequent itemsets by a border-based approach. *Computing in Science & Engineering*, *1*(1), 74–94. doi:10.5626/JCSE.2007.1.1.074

Taniar, D., Rahayu, W. C. S., Lee, V., & Daly, O. (2008). Exception rules in association rule mining. *Applied Mathematics and Computation*, *205*(2), 735–750. doi:10.1016/j.amc.2008.05.020

Verykios, V. S., Emagarmid, A. K., Bertino, E., Saygin, Y., Dasseni, E. (2004). Association rule hiding. *IEEE Transactions on Knowledge and Data Engineering*, *16*(4), 434-447. doi:10.11-.09/TKDE.2004.1269668

Weng, C., Chen, S., & Lo, C., H. (2008). A novel algorithm for completely hiding sensitive association rules. In *Proceedings of the 8th International Conference on Intelligent Systems Design and Applications*. doi:10.1109/ISDA.2008.180

Wu, Y. H., Chiang, C. M., & Chen, A. (2007). Hiding sensitive association rules with limited side effects. *IEEE Transactions on Knowledge and Data Engineering*, *19*(1), 29–42. doi:10.1109/TKDE.2007.250583

Yildiz, B., & Ergenç, B. (2012). Integrated approach for privacy preserving itemset mining. *Lecture Notes in Electrical Engineering*, *110*, 247–260. doi:10.1007/978-1-4614-1695-1_19

Ahmet Cumhur Öztürk received the BSc degree in Computer Engineering from Atılım University, Turkey. From 2006 to 2009 he worked as a software engineer in IT industry. In 2010 he joined Adnan Menderes University as lecturer. He received the MS degree in Computer Engineering from Izmir Institute of Technology. He is currently a lecturer in Adnan Menderes University and PhD student in Computer Engineering department of İzmir Institute of Technology.

Belgin ERGENÇ received the Diploma Degree in Computer Science from Middle East Technical University, Ankara, Turkey in 1983. She worked with different titles and responsibilities in companies like Kordsa, Aksa, Dusa, and Tesco during 1983 – 2000. She received her Master's Degree in Computer Engineering from Izmir Institute of Technology, Turkey in 2002. She continued her education and research in joint PhD program between Paul Sabatier University of Toulouse, France and Ege University, İzmir, Turkey between the years of 2002-2008 and received her PhD degree from both universities. After 17 years of industrial experience she joined academia in 2000. She is an associate professor in the Department of Computer Engineering at Izmir Institute of Technology, Turkey. Her main research interests include query optimization in distributed databases, association rule mining and query processing for linked data. She is managing the research laboratory of Dworld.