

# Frequent Itemset Mining Under Differential Privacy with Multiple Minimum Supports

1<sup>st</sup> Yum-Luh Wang

*Graduate Institute of Electronics Engineering  
National Taiwan University  
Taipei, Taiwan  
r07943161@ntu.edu.tw*

2<sup>nd</sup> Yao-Tung Tsou

*Department of Communications Engineering  
Feng Chia University  
Taichung, Taiwan  
yttso@fcu.edu.tw*

3<sup>rd</sup> Sy-Yen Kuo

*Graduate Institute of Electronics Engineering  
National Taiwan University  
Taipei, Taiwan  
sykuo@ntu.edu.tw*

**Abstract**—Frequent itemset mining is an extensively studied research domain of data mining. The aim is to find interesting correlations between items in a transaction database. However, malicious user might gain sensitive information in the mining process. Differential privacy is the de facto standard when it comes to protecting data. Combining differential privacy with frequent itemset mining can provide strong privacy guarantee while generating statistical information from sensitive data. In existing studies, most algorithms are focusing on finding frequent patterns using one predefined threshold with the protect of differential privacy. However, using single threshold to extract itemsets creates “rare item problem”, and setting respective support thresholds for each item is more adequate to reflect the nature of widely varied items. In this paper, we propose a novel FP-growth-like solution DPCFP++ to solve the rare item problem of frequent itemset mining while guarantee differential privacy at the same time. The experiments illustrate that our algorithm achieves high utility in privately solving rare item problem.

**Index Terms**—Differential Privacy, data mining, multiple minimum supports, frequent itemset mining

## I. INTRODUCTION

In modern era, the development of information technology grows rapidly, more and more services aggregate data from individuals. With the evolution of machine learning and statistics, organizations utilize these crowdsourced data to improve the accuracy of their model, understand the past and predict the future. Frequent itemset mining is a popular research topic in data mining, and it was first introduced in 1993 by Agrawal et al. [1] The original purpose of frequent itemset mining was to analyze customer habits in a supermarket. They tried to find associations between different items that customers frequently bought together. According to the mined results, the management team of the supermarket can make business decisions to maximize profit including product rearrangement, bundle selling, etc. After decades of research, frequent itemset mining can also be extended to many other area besides market basket analysis. For example, text mining, disease diagnosis, and so on.

Majority of existing literatures endeavors on single threshold mining, that is, treat all the items equally and apply a single threshold to decide whether an itemset is frequent or not. However, setting a single threshold for all items is also a drawback. For example, consider two items: soda and television in a wholesale store. Soda is frequently bought while television is an uncommon purchase. If the threshold is set too high, we might consider TV as a less important item, but TV contributes thousand times more to the store revenue than the common purchased soda. On the other hand, if the threshold is too low, the computational cost becomes expensive, the mining result will cause combinational explosion and we will receive many meaningless itemsets. The above dilemma is the famous “rare item problem”. In real life applications, some items tend to have more weights in natural than other items. Researchers [2] [3] [4] [5] have worked on this problem by allowing users to use “multiple minimum supports”, that is, users can apply different threshold on each item. Briefly speaking, rare itemset mining is a more advanced setting of frequent itemset mining.

The convenience which data mining brings comes with the price of privacy leakage. Participants often consider organizations as untrusted entities. The more precise a organization’s model is, the more personal behavioral information reveals. How to de-identification is a mainstream topic in related research. An intuitive approach is to remove participant identifying information such as ID numbers, last names. However, with little background knowledge, adversary can easily distinguish data contributors. Reference [13] uncovered sensitive information of users from Netflix Prize dataset by using another movie database, IMDB, as background knowledge.

Differential Privacy, first proposed by Dwork in [4], had come to rescue in 2006. Since then, many research based on differential privacy have been conducted. Tech giants such as FAANG(Facebook, Apple, Amazon, Netflix, Google) have added features to their products with different kinds of differential privacy model. Differential privacy hides individual information by injecting carefully calibrated noise into data.

It guarantees that the outputs of a mechanism will be approximately the same when two inputs are almost identical, so any adversary cannot explicitly tell from the output that an individual is part of the data or not, and thus protect privacy from exposure. However, too much noise will cause the output to lose utility. On the other hand, too little noise will lead to privacy leakage. Striking a balance between these two is a challenging problem. The additional execution time when combining differential privacy with frequent itemset mining is a big challenge as well.

In this paper, we propose a novel approach DPCFP++ that deals with the rare item problem of frequent itemset mining while satisfying differential privacy. The goal is to achieve both high utility and time efficiency. Our technique merges the advantage of [4] [6] [8]. DPCFP++ is a three-stage algorithm, and each stage consumes different degree of privacy budget. We use the length constraint strategy to limit the high sensitivity of databases. In addition, we build a noisy FP-tree-like data structure to accelerate the process of finding frequent itemsets. In general, our contributions can be summarized as follows:

- We propose DPCFP++, a differential private frequent itemset mining algorithm with multiple minimum supports. The algorithm keeps a good tradeoff between utility and privacy.
- We lower the sensitivity by limiting the length of transactions, and differential privately assign support thresholds to each item.
- We modified MIS-tree structure to meet the requirements of differential privacy, store database information and mine frequent itemsets,
- The overall process of our algorithm only requires as small as three scans of database. The experimental results on real-life and synthetic datasets show that our algorithm outperforms than state-of-art algorithm.

The rest of the paper is organized as follows. In Section II, we briefly introduce related works. Section III describes the background knowledge about differential privacy and frequent itemset mining and provides problem statement. Section IV introduces the proposed algorithm DPCFP++ in detail. Proofs and examples can also be found in this section. Section V evaluates our algorithm on benchmark datasets. Finally, we place conclusions and future works in Section VI.

## II. RELATED WORK

### A. Frequent Itemset Mining

There are mainly two types of frequent itemset mining algorithms. One is Apriori [16], and the other is FP-growth [17]. Apriori finds frequent itemset in breadth-first search manner. It utilizes the Apriori property to reduce search space. More specifically, an itemset is a candidate if all of its subsets are frequent. Apriori scans database and identifies the frequent  $i$ -itemset, and use them to generate candidate  $(i+1)$ -itemsets. After that, it scans database again and verifies these candidates are frequent or not, and generates candidate

$(i+2)$ -itemsets. The process is iteratively executed until no candidate itemset can be generated. On the other hand, FP-growth only needs two database scan. In the first pass, it counts the support of each item and stores items in a header table in decreasing support order. Also, a prefix-tree structure, FP-tree, is constructed in this phase. Next, it injects transactions into FP-tree in second pass. The FP-growth algorithm recursively builds conditional pattern base and conditional FP-tree for each frequent item from the FP-tree and then uses them to discover all frequent itemsets. Due to the time-consuming exhaustive database scans of Apriori, FP-growth is much faster than Apriori,

### B. Frequent Itemset Mining with Multiple Minimum Supports

### C. Differentially Private Frequent Itemset Mining

The NoisyCut algorithm in [8] was proven violating differential privacy in

## III. PRELIMINARIES AND PROBLEM STATEMENT

In this section, we introduce and provide formal definitions of differential privacy and the rare item problem of frequent itemset mining. Finally, we give the problem statement in the last subsection.

### A. Differential Privacy

Differential privacy guarantees that the presence or absence of any individual will not significantly change the output of an algorithm. By this approach, one can only infer limited information about that particular individual from the output.

**Definition 1.** ( $\epsilon$ -Differential Privacy):

Let  $D$  and  $D'$  denote two neighboring databases, which means they differ by at most one record. A mechanism  $\mathcal{M}$  is  $\epsilon$ -differential private if and only if for any  $S \subseteq \text{Range}(\mathcal{M})$

$$\Pr[\mathcal{M}(D) \in S] \leq e^\epsilon \cdot \Pr[\mathcal{M}(D') \in S] \quad (1)$$

The parameter  $\epsilon$  is called privacy budget, it quantifies the degree of privacy. The larger the  $\epsilon$ , the less private the result is. If the  $\epsilon=0$ , the result is perfectly private.

There are a few mechanisms that achieves differential privacy. Laplace mechanism is one of the most popular method. It injects properly calibrated noise into the original output according to the sensitivity. We give the definition of sensitivity as follows.

**Definition 2.** (Sensitivity):

When answering numeric queries of neighboring databases, the sensitivity of a query  $Q$  is

$$\Delta_Q = \max_{D, D'} \|Q(D) - Q(D')\|_1 \quad (2)$$

Sensitivity is used to measure the maximum possible change in the outputs over any two neighboring databases.

**Definition 3.** (Laplace Mechanism):

For any function  $Q : D \rightarrow R$ , Laplace mechanism  $\mathcal{M}$  satisfies  $\epsilon$ -differential privacy by adding noise to the origin output:

$$\mathcal{M}(D) = Q(D) + \text{Lap}\left(\frac{\Delta_Q}{\epsilon}\right), \quad (3)$$

where  $Lap(\frac{\Delta Q}{\epsilon})$  is the noise drawn i.i.d from the Laplace distribution with scale  $\frac{\Delta Q}{\epsilon}$ .

There are some qualitative properties of differential privacy. We will use following properties in our solution.

**Lemma 1. (Sequential Composition):**

Given an algorithm  $\mathcal{M}$  consisting of a sequence of procedures  $\{\mathcal{M}_1, \mathcal{M}_2, \dots, \mathcal{M}_n\}$ . If each procedure  $\mathcal{M}_i$  satisfies  $\epsilon_i$ -differential privacy, then  $\mathcal{M}$  satisfies  $\sum_{i=1}^n \epsilon_i$ -differential privacy.

**Lemma 2. (Closure Under Post-Processing):**

For any algorithm  $\mathcal{A}$  and a  $\epsilon$ -differentially private mechanism  $\mathcal{M}$ , the computation  $\mathcal{A} \circ \mathcal{M}(D)$  satisfies  $\epsilon$ -differential privacy as long as  $\mathcal{A}$  do not access database directly.

**B. Frequent Itemset Mining**

Given a threshold  $\lambda$  and a transaction database  $D = \{t_1, t_2, t_3, \dots, t_n\}$  consisting of  $n$  transactions where each transaction is composed of sets of distinct items in the item universe  $I = \{i_1, i_2, i_3, \dots, i_m\}$ , i.e.  $t_j \subseteq I$ , Frequent itemset mining, also known as frequent pattern mining, refers to discovering all set of patterns while the support of which is greater than or equal to a user-specified threshold.

**Definition 4. (Itemset):**

Any subset of the item universe  $I$  is called an itemset.

An itemset which contains  $k$  distinct items is called  $k$ -itemset.  $k$  is also called the length of the itemset or the cardinality of the itemset. We will use them in this work interchangeably.

**Definition 5. (Support and Threshold):**

The number of transactions in the database containing an itemset is known as the support of that itemset.

We denote support and threshold as  $S$  and  $\lambda$ , respectively. There are two ways to specify  $\lambda$  and  $\lambda$ . One is the percentage way, and the other is by absolute number. It is simple to convert between them, just multiply or divide the number of total transactions  $n$ .

**Definition 6. (Frequent Itemset):**

An itemset  $X$  is frequent if and only if its support  $sup(X)$  is greater than or equal to support threshold  $\lambda$ .

This means that the presence of itemset  $X$  in the database is statistically significant.

**Example 1.** Consider the database shown in Table I. There are  $n = 20$  transactions in the database, and item universe  $I = \{a, b, c, d, e, f, g, h\}$ . The itemset (a,b) is a 2-itemset. It occurs in the database 6 times, so the  $S(a,b)$  is 6. Let's say the user-specified threshold is 2 and we can see (a,b) is a frequent itemset while (a,g) is not, for  $S(a,g) = 1$ .

**C. Rare Item Problem**

By the nature of items, some items appear more frequently than others. However, this sometimes causes meaningless

TABLE I  
A TRANSACTION DATABASE

TID	Items	TID	Items
1	a,b	11	a,d,e
2	a,b,e	12	b,e
3	a,b,f	13	a,b
4	b,e,f	14	a,b,f
5	b,c	15	b,f
6	a,b,e	16	b,c,d,e,f,h
7	b,c,g,h	17	a,e
8	e	18	b,c,d
9	c,d	19	a,e,g
10	c,d	20	c,d

itemsets. We have to find frequent itemsets without generating too many meaningless itemsets. To deal with rare item problem, Liu et al. [2] first tackled the problem with minimum item supports (MIS). MIS allows user to specify respective threshold for every item.

**Definition 7. (Minimum Item Support):**

Minimum Item Support refers to each item's respective user-defined threshold.

MIS can not only be set by user one by one, but also can be set adaptively. Liu et al. also proposed a equation for fast-setting MIS by items' supports. For an item  $i_j$  with support  $sup(j)$ , the Minimum Item Support(MIS) of  $j$  is

$$MIS(i_j) = \max\{\beta * sup(i_j), \lambda\}, \quad (4)$$

where  $\beta \in [0, 1]$ .  $\beta$  is a relevance parameter that controls how the MIS values for items should be related to their frequencies. Noted that when  $\beta=0$ , the MIS value equals to  $\lambda$ , which is the same as the traditional setting of frequent itemset mining.

**Definition 8. (MIS of an Itemset):**

The minimum item support of a  $k$ -itemset  $X = \{i_1, i_2, i_3, \dots, i_k\}$  is the smallest MIS value of items in  $X$ , i.e.

$$MIS(X) = \min\{MIS(i_j) | i_j \in X\}. \quad (5)$$

**Example 2.** Consider the database in Table I. If we set  $MIS(b)=15$  and  $MIS(f)=3$ , then itemset (b) is not frequent since  $S(b) = 13$ . However, itemset (bf) is a frequent itemset since its support is 3 which is equal to  $MIS(bf) = \min\{MIS(b), MIS(f)\} = 3$ .

Example 2 shows that frequent itemset mining using multiple minimum supports is different from single threshold. Under single threshold condition, if an itemset is not frequent, its superset is not frequent either. This principle is called Apriori property, or downward closure property. However, Apriori property cannot be directly applied to our problem.

**D. Problem Statement**

Based on above definitions, we give the formal statement of our problem. Our algorithm works in a centralized model. As illustrate in Figure. Given a transaction database, a threshold and a privacy budget, we want to find all frequent itemsets

whose support is no less than the threshold using multiple minimum supports in a differentially private way.

#### E. Main Challenges

We observe there are two major challenges in using multiple minimum supports to mine frequent itemsets under differential privacy.

##### 1) High Sensitivity

A straightforward approach is to count up each item's support, and directly apply Laplace noise to it. After that we can use traditional FIM algorithm to solve the problem. However, as we illustrate in the following example, this way is not practical.

**Example 3.** Consider a relatively tiny database which just contains 100 different items. If there is a transaction containing all 100 items, and its neighboring database doesn't, then when we start counting supports, all 100 items' support will change. Hence the sensitivity is 100 which is large, and we need to apply a tremendous amount of noise to achieve differential privacy. As a consequence, the enormous noise we add would make the output of traditional FIM algorithm meaningless.

##### 2) Running Time

Most FIM algorithms under differential privacy is time-consuming. We want to develop an algorithm that is agile and scans database as less as possible. An FP-growth-like algorithm doesn't generate candidate itemsets and it significantly scans database less than Apriori-like algorithm does. As a result, we aim to build an algorithm based on FP-growth.

### IV. PROPOSED METHOD

#### A. Overview

We now describe the framework of our proposed algorithm, which is called DPCFP-growth++, in the following paragraph. Motivated by the success in [4] [6] [8], we combine their advantage together in this paper. DPCFP consists of three stages, namely *TruncateDatabase*, *MISAssigning*, *CFP-Growth*. The basic idea is to differentially privately get each item's support and MIS. Next, we sort the items according to the descending order of MIS and put them into MIS header table. Afterwards, we add transactions into MIS-tree by the same order one after another. Finally, we use a FP-growth-like algorithm to find frequent itemsets. Each step consumes different degrees of privacy budget,  $\epsilon_1$ ,  $\epsilon_2$ , and  $\epsilon_3$ , respectively.  $\epsilon_1$ ,  $\epsilon_2$ , and  $\epsilon_3$  follows the sequential composition rule of differential privacy, and hence the sum of them is the amount of privacy budget that user allocated.

#### B. Truncate Database

In this part, we truncate database to limit sensitivity to an acceptable level as [6] proposed. The rationale behind truncating is that most transactions in a database are short in general case. If there exist a few long transactions, then sensitivity will be affected remarkably. These few long transactions which have major impact on sensitivity but less importance on frequent

itemsets is infuriating. To avoid this problem, we pose a limit to the length of transactions. Of course, get rid of some items in transactions will create degrees of information loss.

---

#### Algorithm 1 TruncateDatabase

---

**Input:** database  $D$ ; privacy budget  $\epsilon_1$ ;  
**Output:** truncated database  $D'$

```

1:  $D' \leftarrow \emptyset$ 
2: Read  $D$  to get total items  $m$  and the total number of transactions  $n$ 
3:  $z = \text{EstimateDistribution}(D, \epsilon_1, n, m)$ 
4: Let  $l$  be the smallest integer such that  $\sum_{i=1}^l z_i \geq 0.95$ 
5: for each transaction  $t$  in  $D$  do
6:   add  $t' = \text{RandomTruncate}(t, l)$  to  $D'$ 
7: return  $D'$ 
8: function ESTIMATEDISTRIBUTION( $D, \epsilon, n, m$ )
9:   Let  $z = [z_1, z_2, z_3, \dots, z_m]$ , where  $z_i$  is the number of transactions with cardinality  $i$  in  $D$ 
10:   $z' = z + [\text{Lap}_1, \text{Lap}_2, \text{Lap}_3, \dots, \text{Lap}_m]$ , where  $\text{Lap}_i$  is drawn i.i.d. from Laplace noise  $(\frac{1}{\epsilon_1})$ 
11:  return  $\frac{z'}{n}$ 
12: function RANDOMTRUNCATE( $t, l$ )
13:   $t' = \text{Random Sample min}(|t|, l)$  item from  $t$ 
14:  return  $t'$ 

```

---

In Algorithm 1, function *EstimateDistribution* estimates the noisy distribution of the database. The elements of vector  $z$  are the number of 1-itemsets, 2-itemsets, ..., respectively. Calculating the distribution of the database has privacy leakage concern, so we use Laplace mechanism to perturb the result after  $z$  is calculated. In line 4,  $l$  is the maximal length parameter that controls the tradeoff between information loss and sensitivity reduction. 0.95 is just an experimental value that produces best results for our testing dataset, as the way which [6] proposed. Afterwards, function *RandomTruncate* randomly selects  $l$  items from the transactions that are longer than  $l$  and keeps the rest transactions the same. Finally, we put these new transactions into a new database  $D'$ .

**Theorem 1.** *Algorithm 1 satisfies  $\epsilon_1$ -differential privacy.*

*Proof:* It is already proved in [6] that any  $\epsilon$ -differential private algorithm on a local transformation of databases guarantees differential privacy. Since adding or removing one transaction can only affect  $z$  in one element by one, the sensitivity is 1. Deploying Laplace noise  $(\frac{1}{\epsilon_1})$  to each element of  $z$  satisfies  $\epsilon_1$ -differential privacy. ■

**Example 4.** Let's continue the example in Table I. Say we set the maximal length constraint  $l = 3$ , so TID 7 and 16 will be truncated. We show the truncated database in Table II.

#### C. MIS-Assigning

With the transformed database, we can now count the noisy support and use it to fast assign MIS value to each item by (4).

TABLE II  
TRUNCATED DATABASE

TID	Items	TID	Items
1	a,b	11	a,d,e
2	a,b,e	12	b,e
3	a,b,f	13	a,b
4	b,e,f	14	a,b,f
5	b,c	15	b,f
6	a,b,e	16	b,e,f
7	b,c,h	17	a,e
8	e	18	b,c,d
9	c,d	19	a,e,g
10	c,d	20	c,d

TABLE III  
SUPPORT AND MISTABLE

item	support	MIS
a	9	5
b	13	7
c	6	3
d	5	3
e	9	5
f	5	3
g	1	2
h	1	2

---

**Algorithm 2** NoisySupportandMISTable

---

**Input:** database  $D'$ ; privacy budget  $\varepsilon_2$ ; truncated length  $l$ ; relevance parameter  $\beta$ ; threshold  $\lambda$

**Output:** noisy supports  $S$ ; MISTable  $M$

```

1:  $M \leftarrow \emptyset; S \leftarrow \emptyset$ 
2: for each  $i$  in item universe  $I$  do
3:   Count  $i$ 's support  $i.\text{sup}$ 
4:   Add  $i.\text{sup} = i.\text{sup} + \text{Lap}(\frac{l}{\varepsilon_2})$  to  $S$ 
5:   Add  $i.\text{MIS} = \max\{\beta * i.\text{sup}, \lambda\}$  to  $M$ 
6: end for
7: return  $S$ ;  $M$ 
```

---

In Algorithm 2, we first find the exact accumulated support count. Line 4 and 5 compute and store the noisy supports and MIS table.

**Theorem 2.** Algorithm 2 satisfies  $\varepsilon_2$ -differential privacy.

*Proof:* It straightforward to see the only part that accesses the database in Algorithm 2 is in line 3. Since we already set transaction length constraint to  $l$ , any addition or deletion of a transaction can at most increase or decrease the items' support by  $l$ . In this way, applying Laplace noise ( $\frac{l}{\varepsilon_2}$ ) is enough to guarantee  $\varepsilon_2$ -differential privacy. The rest of algorithm only performs post-processing. ■

**Example 5.** Continue the previous example in Table II, we omit the Laplace noise part for give a better understanding of the algorithm. We use (4) to compute MIS by setting  $\beta = 0.5$ , threshold = 2 and round the results to integer. The supports and the approximate MIS are shown in Table III. We can find (4) adaptively set larger threshold for items that occur more frequently.

#### D. MIS-tree

Next, we introduce the concept of MIS-tree and least minimum support(LMS). MIS-tree is a kind of prefix tree structure that stores database information while LMS is a technique that can reduce the search space when we mining frequent patterns from MIS-tree.

**Definition 9.** (MIS-tree):

A MIS-tree structure consists of two parts. The tree itself and a header table.

- 1) The tree consists of one root labeled as “Null”, and a set of item prefix subtrees as its children. The header table contains all items in descending order of MIS.
- 2) Each node  $N$  in the tree consists of two fields:  $N.\text{name}$ ,  $N.\text{count}$ , where  $N.\text{name}$  indicates which item this node represents and  $N.\text{count}$  indicates the number of transactions reaching this node.
- 3) Each entry in the header table consists of four fields: item's name, item's support, item's MIS, and item's node list which points to each node with the same item name as the entry in the prefix tree.

**Definition 10.** (Least minimum support,LMS):

LMS refers to the lowest MIS value of all frequent itemsets.

**Theorem 3.** For any 1-itemsets  $X$ 's support is lower than LMS, it is not a frequent itemset, as well as its supersets.

*Proof:* Let  $X$  be a 1-itemset, and  $X^+$  be its superset. According to Apriori property,  $\text{sup}(X^+) \leq \text{sup}(X)$ . If  $X$ 's support is lower than LMS ( $\text{sup}(X) < \text{LMS}$ ), the property  $\text{sup}(X^+) \leq \text{sup}(X) < \text{LMS}$  always holds. So  $X^+$  must be an infrequent itemset. ■

---

**Algorithm 3** FindLMS

---

**Input:** noisy supports  $S$ ; MISTable  $M$

**Output:** SortedMISTable  $M$

```

1: Sort  $M$  by descending MIS value.
2: for from the last item  $i_j$  in  $M$  to the first do
3:   if  $\text{sup}(i_j) < \text{MIS}(i_j)$  then
4:     Delete  $i_j$  in  $M$ 
5:   else
6:      $\text{LMS} = \text{MIS}(i_j)$ 
7:     Break
8: end for
9: for each item in  $M$  do
10:  if  $\text{sup}(i_j) < \text{LMS}$  then
11:    Delete  $i_j$  in  $M$ 
12: end for
13: return  $M$ 
```

---

Algorithm 3 shows the algorithm that finds LMS. Since items with support less than LMS cannot be frequent as well as their supersets, we can delete them from the header table. LMS is of crucial significance in our algorithm as this constraint efficiently saves both time and memory when constructing

MIS-tree. Algorithm 3 does not access database and only performs post-processing. Therefore, algorithm 3 does not affect privacy.

**Example 6.** Let us continue with the example in Table III. Items are stored in descending MIS manner in the header table as entries. Because item g and h are not frequent, we can discard these two items in the header table, as shown in Fig. 1(a). The smallest MIS in frequent 1-itemsets is 3, so  $LMS = 3$  instead of original threshold 2.

---

**Algorithm 4** MIS-Tree

---

**Input:** database  $D'$ ; noisy supports  $S$ ; MIS-Table  $M$ ; privacy budget  $\epsilon_3$

**Output:** MIS-tree  $T$

```

1:  $\hat{M} = \text{FindLMS}(S, M)$ 
2: create  $root \leftarrow \emptyset$  for MIS-tree  $T$ 
3: for each transactions  $t'$  in  $D'$  do
4:   Sort  $t'$  in  $\hat{M}$  order, neglect items that are not in  $\hat{M}$ 
5:   AddTransaction( $root, t', \epsilon_3$ )
6: end for
7: UpdateTree( $root$ )
8: CFPGrowth( $T$ )
9: return MIS-tree  $T$ 
10: function ADDTRANSACTION( $root, t', \epsilon_3$ )
11:    $Node \leftarrow root$ 
12:   for each items  $i_j$  in  $t'$  do
13:     Check if  $i_j$  is  $Node$ 's child
14:     if false then
15:       Create NewNode  $i_j$  under  $Node$ ,
16:       initialize  $i_j$  with Laplace noise  $Lap(\frac{1}{\epsilon_3})$ 
17:        $Node \leftarrow NewNode$ 
18:       Add a pointer which points to  $Node$  under
19:       the item entry in header table
20:     else
21:        $Node \leftarrow i_j$ 
22:    $Node.sup ++$ 
23: function UPDATETREE( $node$ )
24:   for each child in  $node$ 's children do
25:     UpdateTree(child)
26:    $node.count += child.count$ 
```

---

As shown in Algorithm 4, We first find LMS, and discard the items that have support less than LMS in line 1. Next, we scan the database and sort each transaction in the descending MIS order. After that, we insert the sorted transactions into the MIS-tree in line 5. Motivated by the success of [8], we exploit similar approach as they did to obfuscate the support count of each node. In this step, only the count of the last node in the path is increased by one instead of every node in line 19. To get accurate support count, a depth-first search function is applied to propagate count from leaf nodes to root node in line 7. Finally, the MIS-tree is constructed and we use CFP-growth algorithm to find frequent itemsets.

**Theorem 4.** Algorithm 4 satisfies  $\epsilon_3$ -differential privacy

*Proof:* In Algorithm 4, only line 3 to 6 accesses database. The process of inserting transactions into MIS-tree can be viewed as a query  $q_D = \{q_1, q_2, \dots, q_n\}$ , where each  $q_i$  ( $1 \leq i \leq n$ ) represents a count for a possible combination of items. So each node in the MIS-tree corresponds to a  $q_i$ . Each transaction is mapped to a single path in the MIS-tree and merely increases the count of the last node by one. In this way, adding or removing one transaction can only change the count of a node by at most one. Hence the sensitivity of releasing counts in the MIS-tree is also one. Initialize each node with Laplace noise ( $\frac{1}{\epsilon_3}$ ) satisfies  $\epsilon_3$ -differential privacy. The rest of the algorithm performs post-processing, and does not influence privacy. ■

**Example 7.** Here we show the construction process of MIS-tree. Consider the truncated transaction database in Table II and MIS table in Table III. Fig. 1 gives an example of building a noisy MIS-tree. Again, we omit the Laplace noise in the figure to give a better explanation of the mechanism. Fig. 1(b) shows the MIS-tree after adding the first transaction. The first transaction is  $\{a, b\}$ , we sort it by descending MIS order, and it thus becomes  $\{b, a\}$ . This creates two new nodes  $\langle b, 0 \rangle$  under root and  $\langle a, 0 \rangle$  under  $\langle b, 0 \rangle$ . We only increase the support count of the last node in the path by one to meet the Laplace mechanism requirements. So the  $\langle a, 0 \rangle$  becomes  $\langle a, 1 \rangle$ . Notice that the Laplace noise which we omitted should have been injected when initialing each node. The MIS-tree after scanning the second transaction is as Fig. 1(c) shows. After scanned all the transactions, we have the MIS-tree as Fig. 1(d). Finally, the support count of each node is added to its parent from leaf nodes toward the root node. The ultimate MIS-tree is shown in Fig. 1(e).

---

**Algorithm 5** CFP-Growth

---

**Input:** MIS-tree  $T$  **Output:** frequent itemsets  $F$

```

1:  $F \leftarrow \emptyset$ 
2: CFPGrowth()
3: return frequent itemsets  $F$ 
```

---

## V. EXPERIMENT

In this section, we evaluate our algorithm with [7] in terms of utility and execution time. We use PB to denote the method in [7]. The algorithms are implemented in python3 and we conduct all experiments on a 64 bit Windows 10 PC with 3.40GHZ Intel i7-6700 CPU and 16GB RAM. Due to randomness in differential privacy, we run each algorithm ten times and report the average results for each experiment. In our experiments, we use relative threshold. It is the percentage way to represent threshold as depicted in Definition 5.

### A. Datasets

We run our algorithm on the following six datasets. Five real-life datasets(BMS1, BMS2, BMS-POS, Kosarak, Retail)

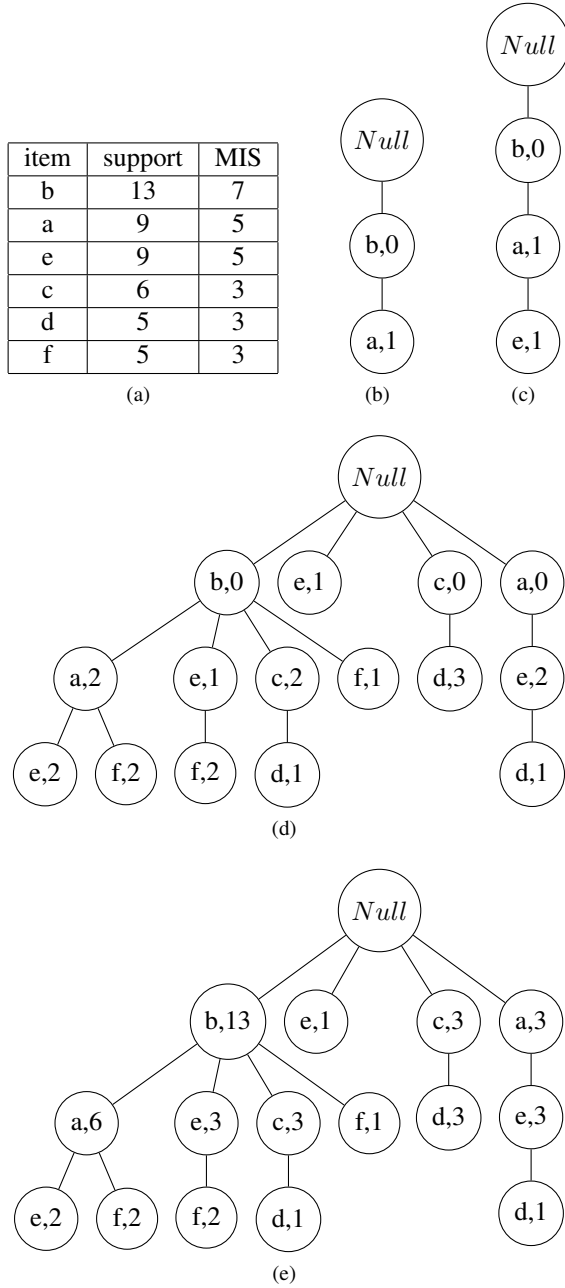


Fig. 1. Construction of a MIS-tree. (a) Header table (b) After scanning the first transaction (c) After scanning the second transaction (d) After scanning the whole database (e) After support count propagation

and one synthetic dataset(T10I4D100K) were used in the experiments. All of the datasets can be found on FIMI Repository [12]. Table IV provides key characteristics of these datasets.

- BMSWebview1(BMS1) and BMSWebview2(BMS2) contain several months of click stream data from two e-commerce website. Each transaction contains the web pages that a visitor viewed in one session.
- BMS-POS contains POS data of electronics retailer over several years. Each transaction contains all the product categories that a customer purchased.
- Kosarak contains the clickstream data of a Hungarian

news portal. Each transaction contains all web pages that a visitor clicked from the portal.

- Retail is a database that collected from an anonymous Belgian retail market over approximately five months. Each transaction is a set of products that a customer purchased.
- T10I4D100K is generated by IBM Almaden Quest market basket data generator.

TABLE IV  
DATASET CHARACTERISTICS

Dataset	n	m	max t	avg t
BMS1	59602	497	267	2.5
BMS2	77512	3340	161	5.0
BMS-POS	515597	1657	164	6.5
Retail	88162	16470	76	10.3
Kosarak	990002	41270	2498	8.1
T10I4D100K	100000	870	29	10.1

### B. Metrics

We use F-score, which is a widely used F-measure to evaluate utility of algorithms. The larger the F-score, the higher the utility.

$$precision = \frac{|U_p \cap U_c|}{|U_p|}, \quad recall = \frac{|U_p \cap U_c|}{|U_c|},$$

where  $U_p$  is the mined frequent itemsets,  $U_c$  is the correct frequent itemsets. F-score is the harmonic mean of precision and recall. The equation below is the formal definition of F-score.

$$F - score = 2 * \frac{precision * recall}{precision + recall}$$

### C. Experimental Results

1) *Utility*: There is no differential private frequent itemset mining algorithm dealing with minimum multiple supports. Also, our algorithm finds all frequent itemsets with respect to a threshold while PB finds top-k frequent itemsets. As a consequence, We adapt PB to find all frequent itemsets. we set k to be larger the number of frequent itemsets for a given threshold. Noted that this approach might create privacy concern.

2) *Runtime*: In this section, we compare the runtime of the proposed algorithm and PB. We set the relevance parameter  $\beta=0.25$ .  $\epsilon_1, \epsilon_2, \epsilon_3 = 0.5$ , Fig. 3 shows the runtime of our algorithm and PB on six datasets with respect to different privacy budgets. We find that the runtime of proposed algorithm is significantly lower than PB. This is because of PB does not scale when k is large. In BMS-POS dataset, PrivBasis 4 hours while DPCFP++ only needs 35 seconds. We find an interesting result when privacy budget is small,

## VI. CONCLUSION

In this paper, we propose a novel differentially private algorithm for frequent itemset mining with multiple minimum supports.

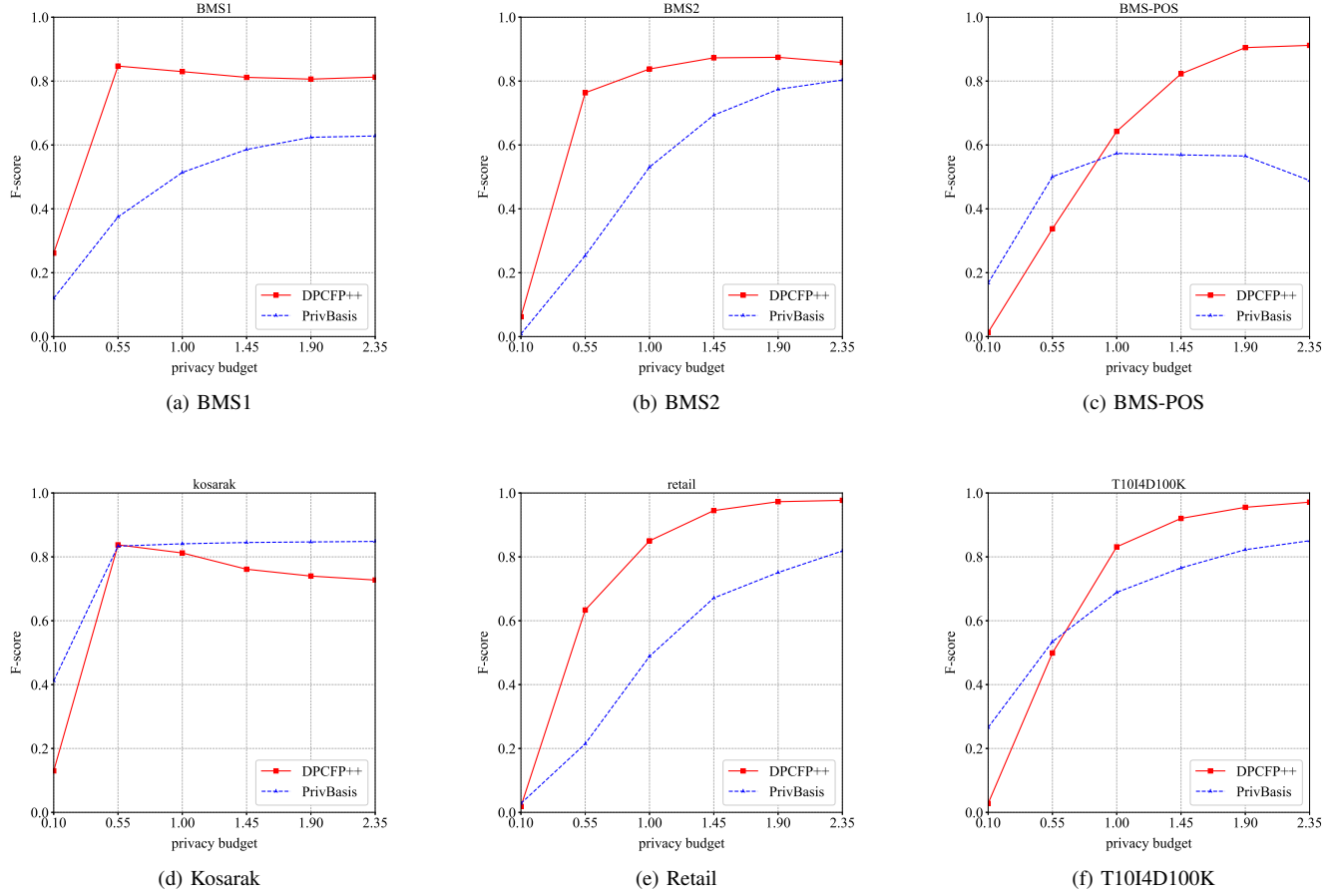


Fig. 2. F-score.

## REFERENCES

- [1] R. Agrawal, T. Imielinski, and A. Swami. "Mining association rules between sets of items in large databases," in Proceedings of ACM-SIGMOD, 1993.
- [2] B. Liu, W. Hsu, and Y. Ma. "Mining association rules with multiple minimum supports," in Proceedings of ACM-SIGKDD, 1999.
- [3] Y. Hu, and Y. Chen. "Mining association rules with multiple minimum supports: a new mining algorithm and a support tuning mechanism," in Decision Support System, 2006.
- [4] R. Uday Kiran, and P. Krishna Reddy. "Novel techniques to reduce search space in multiple minimum supports-based frequent pattern mining algorithms," in Proceedings of ACM-EDBT, 2011.
- [5] W. Gan, J. C.-W. Lin, P. Fournier-Viger, H. Chao, and J. Zhan. "Mining of frequent patterns with multiple minimum supports," in Engineering Applications of Artificial Intelligence, 2017.
- [6] C. Zeng, J. F. Naughton, and J. Cai. "On differentially private frequent itemset mining," in Proceedings of the VLDB, 2012.
- [7] N. Li, W. H. Qardaji, D. Su, and J. Cao. "Privbasis: frequent itemset mining with differential privacy," in Proceedings of the VLDB, 2012.
- [8] J. Lee and C. W. Clifton. "Top-k frequent itemsets via differentially private fp-trees," in KDD, 2014.
- [9] C. Dwork. Differential privacy. In ICALP, 2006.
- [10] Y. Chen and A. Machanavajhala. "On the privacy properties of variants on the sparse vector technique," CoRR, 2015.
- [11] J. Zhang, X. Xiao, and X. Xie. "Privtree: A differentially private algorithm for hierarchical decompositions," in ACM-SIGMOD, 2016.
- [12] Frequent Itemset Mining Implementations Repository. <http://fimi.uantwerpen.be/data/>, 2004.
- [13] A. Narayanan and V. Shmatikov. "Robust de-anonymization of large sparse datasets," in S&P, 2008.
- [14] P. Fournier-Viger, J.C.-W. Lin, B. Vo, T.T. Chi, J. Zhang, and H.B. Le. "A survey of itemset mining," in WIREs Data Mining and Knowledge Discovery, 2017.
- [15] T. Zhu, G. Li, W. Zhou, P. S. Yu. "Differentially private data publishing and analysis: a survey," in TKDE, 2017.
- [16] R. Agrawal, R. Srikant. "Fast algorithms for mining association rules," in Proceedings of the VLDB, 1994.
- [17] J. Han, J. Pei, and Y. Yin. "Mining frequent patterns without candidate generation," in ACM-SIGMOD, 2000.



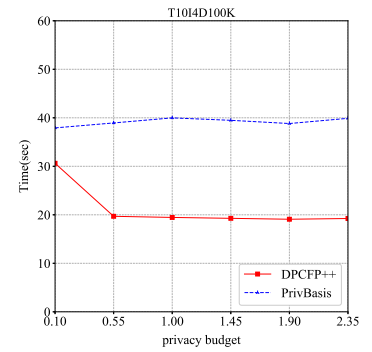
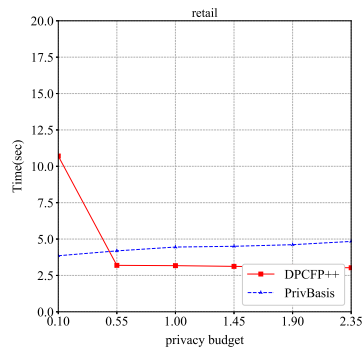
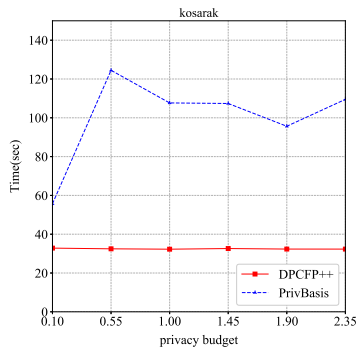
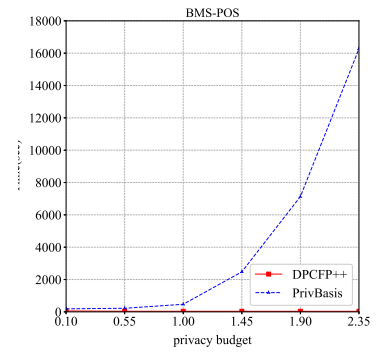
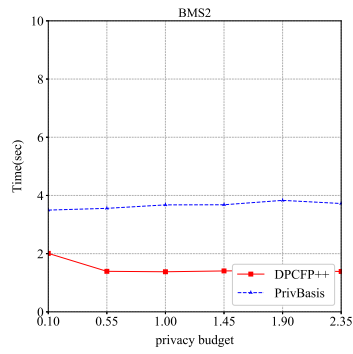
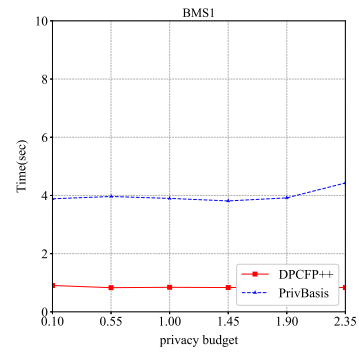


Fig. 3. Runtime.