Frequent Itemset Mining Under Differential Privacy with Multiple Minimum Supports

1st Yum-Luh Wang
Graduate Institute of Electronics Engineering
National Taiwan University
Taipei, Taiwan
r07943161@ntu.edu.tw

2nd Yao-Tung Tsou
Department of Communications Engineering
Feng Chia University
Taichung, Taiwan
yttsou@fcu.edu.tw

3rd 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. Also, the execution time of prior approaches are lengthy. In this paper, we propose a novel FP-growth-like algorithm DPCFP-growth++ to solve the rare item problem of frequent itemset mining while guarantee differential privacy at the same time. We first limit the sensitivity by truncating the transactions, and assign each item with its own threshold. Finally, we construct a differentially private MIS-tree and derive frequent itemsets from the tree. We add Laplace noise in each step, and prove our algorithm satisfies differential privacy. The experiments on real-world and synthetic datasets illustrate that our algorithm acheieves both high utility and efficiency.**

***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 [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 explosure. 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-growth++ that deals with the rare item problem of frequent itemset mining while satisfying differential privacy. To the best of our knowledge, our work is the first algorithm that focus on this problem. The goal is to achieve both high utility and time efficiency. Our technique merges the advantage of [4] [6] [8]. DPCFP-growth++ 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-growth++, 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-growth++ 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

Liu, Hsu and Ma [2] first pointed out the drawbacks of single support threshold, and developed an apriori-based algorithm MSApriori to solve this problem. Hu and Chen [18] proposed a FP-growth-like algorithm CFP-growth to accelerate the mining process. Uday Kiran and Krishna Reddy [4] adapted it to CFP-growth++ and proposed four pruning techniques, further improved the mining efficiency. Reference [5] used a FP-ME-tree to vertically mine frequent itemsets with multiple minimum supports.

C. Differentially Private Frequent Itemset Mining

Reference [19] applied differential privacy on top-k most frequent itemset mining. Reference [7] proposed the PrivBasis algorithm which projects database onto lower dimensions, and compute maximal cliques in graph to find top-k most frequent itemsets. Zeng, Naughton and Cai [6] proposed SmartTruncating to lower the sensitivity by limiting the cardinality of transactions. Reference [20] argued that transaction should be splitted instead of being truncated. The NoisyCut algorithm in [8] is a new differential privacy mechanism, but was later proven violating differential privacy in [10] [11]. Despite this, [8] successfully adapted FP-tree in their literature. Wang et al. [21] invented a new differential privacy mechanism called sequence exponential mechanism(SEM). However, all the above algorithms do not consider multiple support thresholds, and either only find top-k frequent itemsets or face running time issue.

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

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

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

There are a few mechanisms that acheieves 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

$$\triangle_Q = \underset{D,D'}{max} \|Q(D) - Q(D')\|_1 \tag{2}$$

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 ε-differential privacy by adding noise to the origin output:

$$\mathcal{M}(D) = Q(D) + Lap(\frac{\triangle_Q}{\varepsilon}), \tag{3}$$

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

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 $\{\mathcal{M}_1, \mathcal{M}_2, \ldots, \mathcal{M}_n\}$. If each procedure $\mathcal{M}_i$ satisfies ε_i-differential privacy, then $\mathcal{M}$ satisfies $\sum_{i=1}^{n} \varepsilon_i$-differential privacy.

Lemma 2. *(Closure Under Post-Processing):*
For any algorithm $\mathcal{A}$ and a ε-differentially private mechanism $\mathcal{M}$, the computation $\mathcal{A} \circ \mathcal{M}(D)$ satisfies ε-differential privacy as long as A do not access database directly.

B. Frequent Itemset Mining

Given a threshold λ and a transaction database $D = \{t_1, t_2, t_3, \ldots, t_n\}$ consisting of n transactions where each transaction is composed of sets of distinct items in the item universe $I = \{i_1, i_2, i_3, \ldots, i_m\}$, 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.

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

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 interchangebly.

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 sup and λ, respectively. There are two ways to specify sup and λ. 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 λ.

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 sup(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 sup(a,g) = 1.

C. Rare Item Problem

By the nature of items, some items appear more frequently than others. However, this sometimes causes meaningless itemsets. We have to find frequent itemsets without generating too many meaningless itemsets. To deal with rare item problem, [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. Reference [2] 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

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

where $\beta \in [0, 1]$. β is a relevance parameter that controls how the MIS values for items should be related to their frequencies. Noted that when β=0, the MIS value equals to λ, 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 $X = \{i_1, i_2, i_3, \ldots, i_k\}$ *is the smallest MIS value of items in* X*, i.e.*

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

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 sup(b) = 13. However, itemset (b,f) is a frequent itemset since its support is 4 which is larger than MIS(b,f) = 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. 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 oberserve 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 direct 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 develope 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-growth++ consists of three stages, namely *Truncate Database, MIS-Assigning, MIS-tree.* The basic idea is to differential 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, ε_1, ε_2,and ε_3, respectively. ε_1, ε_2,and ε_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 a acceptable level as [6] proposed. The rationale behind truncating is that most transactions in a database is 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.

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 a experimental value that produces best results for our testing dataset. as the way which [6] proposed. Afterwards, function *RandomTruncate* randomly select ℓ items from the transactions that are longer than ℓ and keep the rest transactions the same. Finally, we put these new transactions into a new database D'.

Theorem 1. *Algorithm 1 satisfies* ε_1*-differential privacy.*

Proof: It is already proved in [6] that any ε-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 $(\frac{1}{\varepsilon_1})$ to each element of z satisfies satisfies ε_1-differential privacy. ■

Algorithm 1 TruncateDatabase

```
Input: database $D$; privacy budget $\varepsilon_1$;
Output: truncated database $D'$
$D' \leftarrow \emptyset$
Read $D$ to get total items $m$ and the total number of transactions $n$
$z$ = EstimateDistribution($D$, $\varepsilon_1$, $n$, $m$)
Let $\ell$ be the smallest integer such that $\sum_{i=1}^{l} z_i \geq 0.95$
for each transaction $t$ in $D$ do
    add $t'$ = RandomTruncate($t$, $\ell$) to $D'$
end for
return $D'$
function EstimateDistribution($D$,$\varepsilon$,$n$,$m$)
    Let $z = [z_1, z_2, z_3, \ldots, z_m]$, where $z_i$ is the number of transactions with cardinality $i$ in $D$
    $z' = z + [Lap_1, Lap_2, Lap_3, ..., Lap_m]$, where $Lap_i$ is drawn i.i.d. from Laplace noise ($\frac{1}{\varepsilon_1}$)
    return $\frac{z'}{n}$
function RandomTruncate($t$,$\ell$)
    $t'$ = Random Sample min($|t|$,$\ell$) item from t
    return $t'$
```

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

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

C. MIS-Assigning

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

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

Theorem 2. *Algorithm 2 satisfies ε_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, appling Laplace noise ($\frac{\ell}{\varepsilon_2}$)is enough to garantee ε_2-differential privacy. The rest of algorithm only performs post-processing. ■

Algorithm 2 NoisySupportandMISTable

```
Input: database $D'$; privacy budget $\varepsilon_2$; truncated length $\ell$; 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 $i.sup$
4:     Add $i.\hat{sup} = i.sup + Lap(\frac{\ell}{\varepsilon_2})$ to $S$
5:     Add $i.MIS$ = $max\{\beta * i.\hat{sup}, \lambda\}$ to $M$
6: end for
7: return $S$; $M$
```

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

Example 5. Continue the previous example in Table II, we ommit 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.name, N.count, where N.name indicates which item this node represents and N.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 contains 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, $sup(X^+) \leq sup(X)$. If X's support is lower than LMS ($sup(X)$ <LMS), the property $sup(X^+) \leq sup(X)$ <LMS always holds. So X^+ must be an infrequent itemset. ■

Algorithm 3 FindLMS

```
Input: noisy supports $S$; MISTable $M$
Output: SortedMISTable $M$
Sort $M$ by descending MIS value.
for from the last item $i_j$ in $M$ to the first do
    if $sup(i_j) < MIS(i_j)$ then
        Delete $i_j$ in $M$
    else
        LMS= $MIS(i_j)$
        Break
end for
for each item in $M$ do
    if $sup(i_j) <$ LMS then
        Delete $i_j$ in $M$
end for
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.

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. The original CFP-growth++ is shown in Algorithm 5 for reader's information.

Theorem 4. *Algorithm 4 satisfies ε_3-differential privacy*

Algorithm 4 DPCFP-growth++

```
Input: database $D'$; noisy supports $S$; MISTable $M$; privacy budget $\varepsilon_3$
Output: Frequent itemsets $F$
 1: $\hat{M}$ = 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($root, t', \varepsilon_3$)
 6: end for
 7: $T$ ←UpdateTree($root$)
 8: $F$ ←CFP-Growth($T$,[])
 9: return Frequent itemsets $F$
10: function AddTransaction($root, t', \varepsilon_3$)
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 $Lap(\frac{1}{\varepsilon_3})$
17:             $Node \leftarrow NewNode$
18:             Update the node list in header table
19:         else
20:             $Node \leftarrow i_j$
21:     end for
22:     $Node.sup$ ++
23: function UpdateTree($T, node$)
24:     for each child in $node$'s children do
25:         UpdateTree($T$,child)
26:         node.count += child.count
27:     end for
28:     return T
```

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 $q_D = \{q_1, q_2, \ldots, q_n\}$, 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 $(\frac{1}{\varepsilon_3})$ satisfies ε_3-differential privacy. The rest of the algorithm performs post-processing, and does not influence privacy. ■

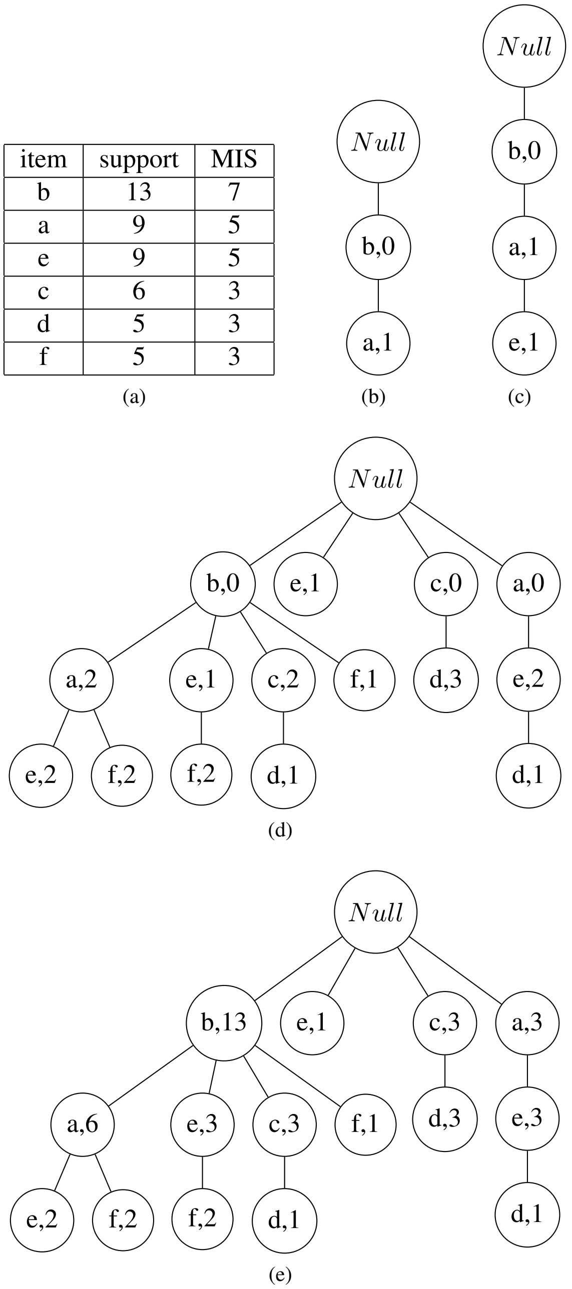
Example 7. Here we show the construction process of MIS-tree. Consider the truncated transaction database in Table II and MIS table in Table III. Fig. 1 gives an example of building a noisy MIS-tree. Again, we ommit the Laplace noise in the figure to give a better explaination of the mechanism. Fig. 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 ⟨b,0⟩ under root and ⟨a,0⟩ under ⟨b,0⟩. We only increase the support count of the last node in the path by one to meet the Laplace mechanism requirements. So the ⟨a,0⟩ becomes ⟨a,1⟩. Notice that the Laplace noise which we ommited 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 $T$; prefix $\alpha$
Output: frequent itemsets $F$
$F \leftarrow \emptyset$
for each item $i$ in header table do
    Generate pattern $\beta = i \cup \alpha$, support = i.support
    Construct $\beta$'s conditional pattern base and
    conditional MIS-tree $T_\beta$
    if $T_\beta \neq \emptyset$ then
        CFPGrowth($T_\beta, \beta, MIS(i)$)
end for
return frequent itemsets $F$
function CFPGROWTH($Tree, \alpha, MIS(i)$)
    for each i in the header table of $Tree$ do
        Generate pattern $\beta = i \cup \alpha$, support = i.support
        Construct $\beta$'s conditional pattern base and
        conditional MIS-tree $T_\beta$
        if $T_\beta \neq \emptyset$ then
            if $T_\beta$ contains a single path then
                for each combination $\gamma$ of nodes do
                    Generate pattern with $\gamma \cup \beta$,
                    support=min support of nodes in $\gamma$
                end for
            else
                CFPGrowth($T_\beta, \beta, MIS(i)$)
    end for
```

V. Experiment

In this section, we evaluate our algorithm in terms of utility and execution time. There is no differentially private frequent itemset mining algorithm dealing with minimum multiple supports available for comparison. As a consequence, We adapt PrivBasis [7] to find all frequent itemsets and compare our algorithm against it. We use PB to denote PrivBasis in the rest of the paper. 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 averge results for each experiment. In our experiments, we use relative threshold. It is the percentage way to represent threshold as depicted in Definition 5. We set the relative threshold $\lambda = 0.01$ and the relevance parameter $\beta = 0.25$, and use MSApriori [2] as ground truth. We allocate the total privacy budget as the following heuristic: $\varepsilon_1 = min(0.05, \frac{\varepsilon}{10})$, ε_2 and ε_1 takes 40% and 60% of the rest privacy budget ($\varepsilon - \varepsilon_1$),respectively.



item	support	MIS
b	13	7
a	9	5
e	9	5
c	6	3
d	5	3
f	5	3

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

A. Datasets

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

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

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: First, we compare the utility of the proposed DPCFP-growth++ method against PB. Our algorithm finds all frequent itemsets with respect to a threshold while PB finds top-k frequent itemsets. Since the aim of two algorithm is somewhat different, and using multiple support threshold decreases the number of mined frequent itemsets, we set the k of PB to be larger than the number of true frequent itemsets under single threshold condition for a given threshold. Table V shows the output of Apriori and MSApriori and the k we set when threshold = 0.01. Noted that this approach creates privacy concern since we need to run an unprotected FIM algorithm first and apply the output to PB.

Fig. 2, 3, 4 shows different privacy budgets on different datasets versus the F-score, precision and recall, respectively. The results of F-score shows that our algorithm outperforms PB most of the time. Only when privacy budget is small, our algorithm shows a slightly inferior result. The reason behind this is that the large noise we added leads to more frequent 1-itemsets so that the LMS cannot exert its influence by pruning infrequent 1-itemsets. Also, the large noise of each MIS-tree node severely affect the precision. A node has more descendants will have more noise after we update the MIS-tree. Hence, the mining result will contain more redundant itemsets. As the value of ε increases, the noise of each node decreases, and the precision becomes more accurate.

Another interesting phenomenon is that the recall decreases in some dataset. After further experiments, we find that it is caused by transaction truncation. In Table VI, we show the length constraint ℓ and the average of truncated items per truncated transaction. When privacy budget is small, there are tremendous redundant itemsets, and the recall is high due to fortuitous. As the privacy budget increases, the information loss of truncation gradually reveals. In some dataset, the truncated part is large when comparing to the length constraint. In Kosarak, the discarded items are even large enough to form a new transaction. By contrast, the Retail and T10I4D100K only discard a small protion of information. This explains why Kosarak dataset suffers the most serious decreasing in the recall.

TABLE V
RATIONALE OF K IN PB

Dataset	**frequent itemsets Apriori**	**frequent itemsets MSApriori**	**k of PB**
BMS1	77	77	100
BMS2	81	81	100
BMS-POS	1099	646	1200
Retail	159	147	180
Kosarak	383	299	400
T10I4D100K	385	384	400

TABLE VI
INFORMATION LOSS OF TRUNCATION

Dataset	ℓ	**# of truncated transactions**	**avg truncated items**
BMS1	7	2737	8.0
BMS2	15	3758	9.7
BMS-POS	20	22867	8.8
Retail	27	3920	7.8
Kosarak	28	48841	45.1
T10I4D100K	16	4820	2.3

2) Runtime: Fig. 5 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. In BMS-POS dataset, the execution time of PrivBasis is more than 4 hours while DPCFP-growth++ only needs about 35 seconds when $\varepsilon = 2.35$. This is because of PB does not scale when k is large.

We find an interesting result that in some dataset the runtime is a little longer when privacy budget is small. There are mainly two reasons for that. The first is LMS cannot effectively reduce the search space. The second is the tremendous redundant itemsets caused by noise after tree update. These are exact the same reason as the low precision mentioned in the previous paragraph. Despite this, DPCFP-growth++ still outperforms PB under almost every different ε condition.

VI. Conclusion

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

The algorithm is composed of three stages. The total privacy budget follows sequential composition theorem of differential privacy. In order to find frequent 1-itemsets and assign MIS, we first differential privately limit the length of transactions by random truncation to reduce the sensitivity. Secondly, we apply Laplace noise to the accumulated support counts of truncated transactions. After that, we store the information of the database into a differentially private MIS-tree. Finally, by using regular DPCFP-growth++ algorithm, we can uncover the frequent itemsets.

From the experiments on real-wold and synthetic datasets, we find our algorithm DPCFP-growth++ significantly outperforms current algorithm and acheieves both high utility and high efficiency.

As for future work, there are some directions that we can utilize. First of all, extend our work to high-utility itemset mining. High-utility mining is an advanced setting of frequent itemset mining. It takes the purchase quantities into account. User can set unit profit for each item, and the total utility can be computed by the sum of the products of the quantities and unit profits. Literatures such as [22] adapted FP-growth-like algorithms which is similar to our work.

Secondly, reduce the information loss. Transaction splitting method in [20] seems a better idea than simply truncate transactions. More specifically, it splits a long transaction into multiple pieces of small transcations and add them into the database while the random truncating method just neglects the items that are above length constraint. However, it needs more time to preprocess the dataset and may drastically slow down the total mining process.

References

[1] R. Agrawal, T. Imielinski, and A. Swami. “Mining association rules between sets of items in large databases,” in Proceedings of the 1993 ACM SIGMOD international conference on Management of data (SIGMOD ’93), pp. 207-216, 1993.

[2] B. Liu, W. Hsu, and Y. Ma. “Mining association rules with multiple minimum supports,” in Proceedings of the 5th ACM SIGKDD international conference on Knowledge discovery and data Mining(KDD ’99), 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, vol. 42, no. 1, pp. 1-24, 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 the 14th International Conference on Extending Database Technology (EDBT/ICDT ’11), pp. 11-20, 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, vol. 60, pp. 83-96, 2017.

[6] C. Zeng, J. F. Naughton, and J. -Y. Cai. “On differentially private frequent itemset mining,” in Proceedings of the VLDB Endowment, vol. 6, no. 1, pp. 25-36, 2012.

[7] N. Li, W. H. Qardaji, D. Su, and J. Cao. “PrivBasis: Frequent itemset mining with differential privacy,” in Proceedings of the VLDB Endowment, vol. 5, no. 11, pp. 1340-1351, 2012.

[8] J. Lee and C. W. Clifton. “Top-k frequent itemsets via differentially private fp-trees,” in Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data Mining(KDD ’14), 2014.

[9] C. Dwork. “Differential privacy,” in 33rd International Colloquium on Automata, Languages and Programming (ICALP), pp.1-12, 2006.

[10] Y. Chen and A. Machanavajjhala. “On the privacy properties of variants on the sparse vector technique,” in CoRR, 2015.

[11] J. Zhang, X. Xiao, and X. Xie. “PrivTree: A differentially private algorithm for hierarchical decompositions,” in Proceedings of the 2016 International Conference on Management of Data (SIGMOD ’16), pp. 155-170, 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 IEEE Symposium on Security and Privacy (S &P), pp. 111-125, 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,vol. 7, no. 4, 2017.

[15] T. Zhu, G. Li, W.Zhou, P. S. Yu. “Differentially private data publishing and analysis: a survey,” in IEEE Transactions on Knowledge and Data Engineering (TKDE), vol. 29, no. 8, pp. 1619-1638, 2017.

[16] R. Agrawal, R. Srikant. “Fast algorithms for mining association rules,”in Proceedings of the 20th International Conference on Very Large Data Bases (VLDB ’94), pp. 487-499, 1994.

[17] J. Han, J. Pei, and Y. Yin. “Mining frequent patterns without candidate generation,” in Proceedings of the 2000 ACM SIGMOD international conference on Management of data (SIGMOD ’00), pp. 1-12, 2000.

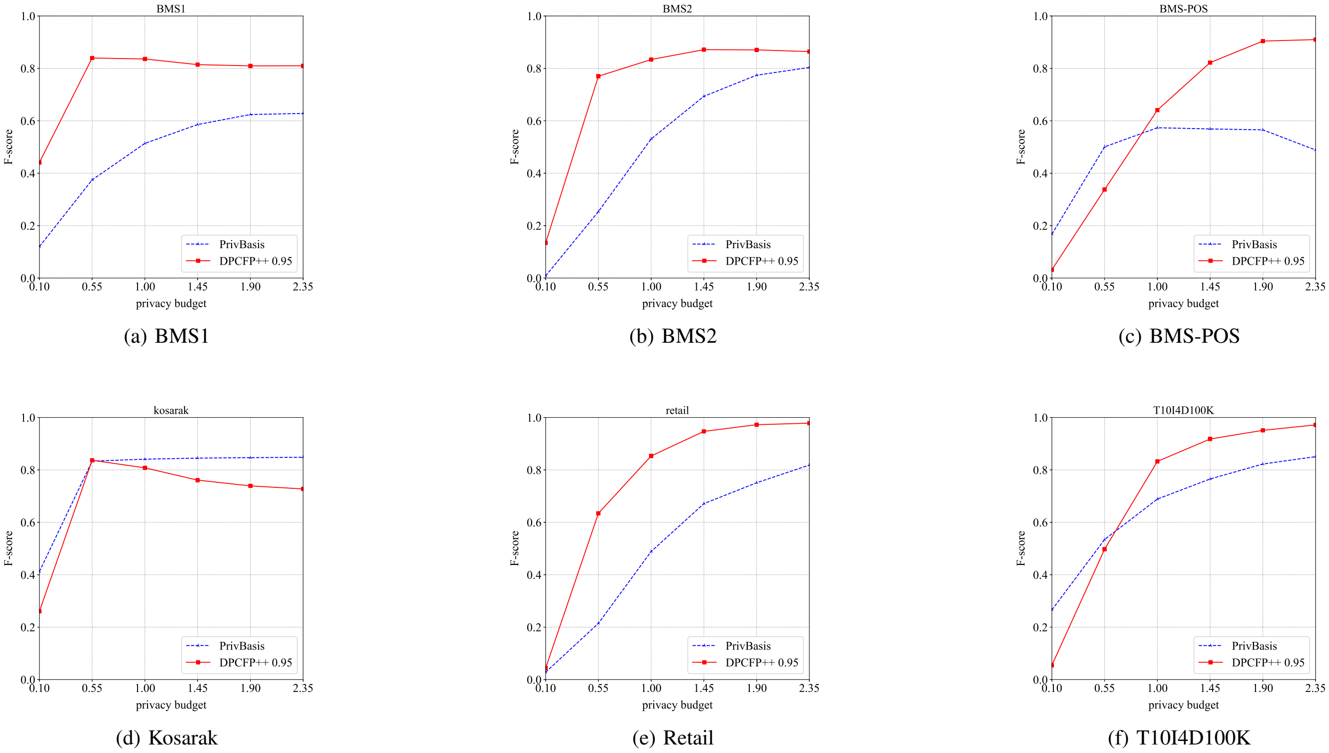
[18] Y.-H. Hu and Y.-L. Chen. “Mining association rules with multiple minimum supports: a new mining algorithm and a support tuning mechanism,” in Decision Support System, vol. 42, no. 1, pp. 1–24, 2006.

[19] R. Bhaskar, S. Laxman, A. Smith, and A. Thakurta. “Discovering frequent patterns in sensitive data,” in Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD ’10), pp. 503–512, . 2010.

[20] S. Su, S. Xu, X. Cheng, Z. Li and F. Yang. “Differentially private frequent itemset mining via transaction splitting,” in IEEE Transactions on Knowledge and Data Engineering (TKDE), vol. 27, no. 7, pp. 1875-1891, 2015.

[21] N. Wang, X. Xiao, Y. Yang, Z. Zhang, Y. Gu and G. Yu. “PrivSuper: a superset-first approach to frequent itemset mining under differential privacy,”in 2017 IEEE 33rd International Conference on Data Engineering (ICDE), pp. 809-820, 2017.

[22] V. S. Tseng, C. Wu, B. Shie, and P. S. Yu. “UP-Growth: An efficient algorithm for high utility itemset mining,” in Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data Mining(KDD ’10), pp. 253–262, 2010.



(a) BMS1

(b) BMS2

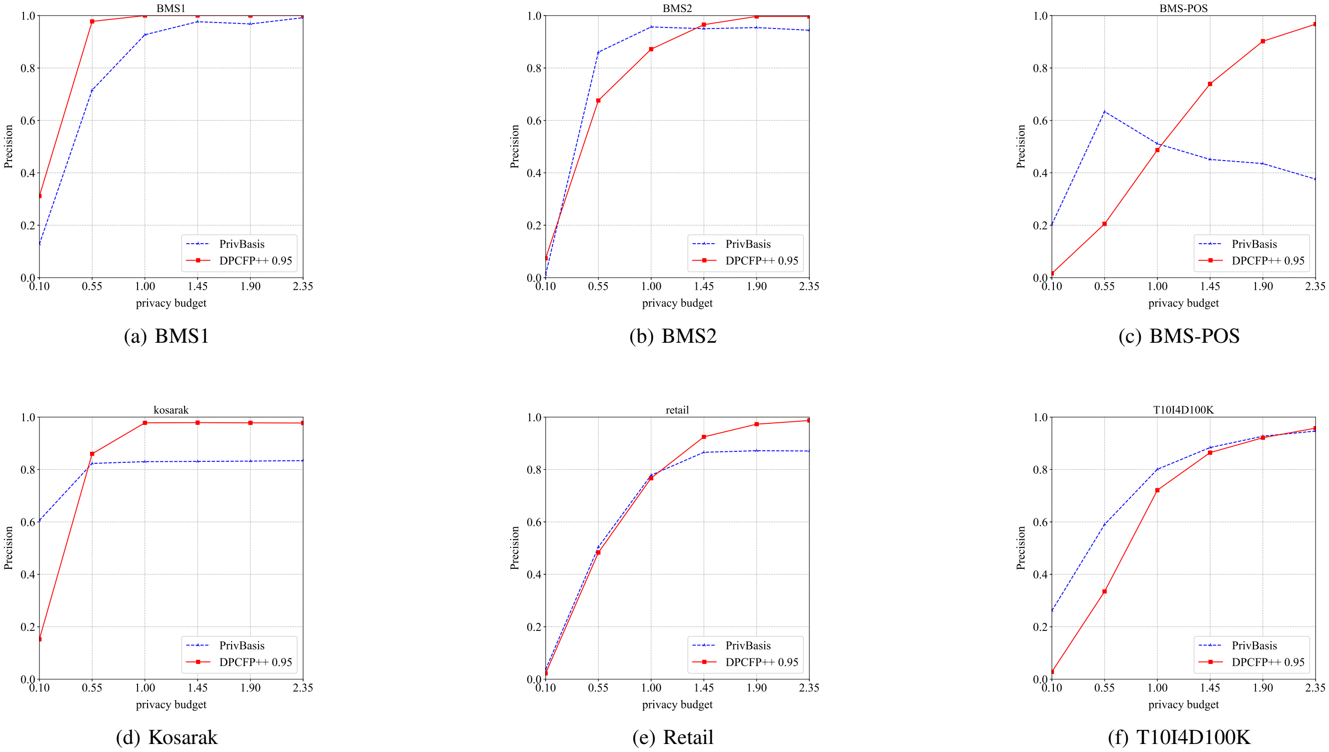
(c) BMS-POS

(d) Kosarak

(e) Retail

(f) T10I4D100K

Fig. 2. F-score



(a) BMS1

(b) BMS2

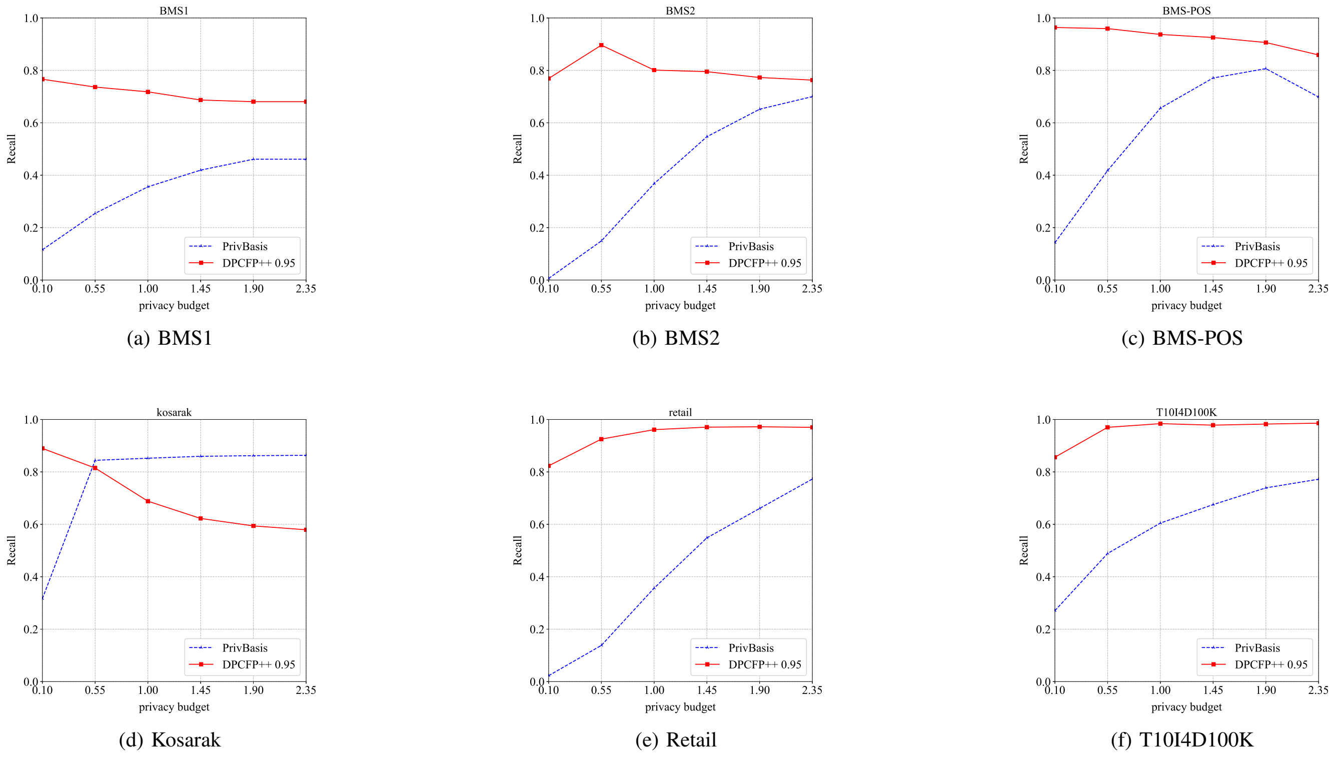
(c) BMS-POS

(d) Kosarak

(e) Retail

(f) T10I4D100K

Fig. 3. Precision



(a) BMS1

(b) BMS2

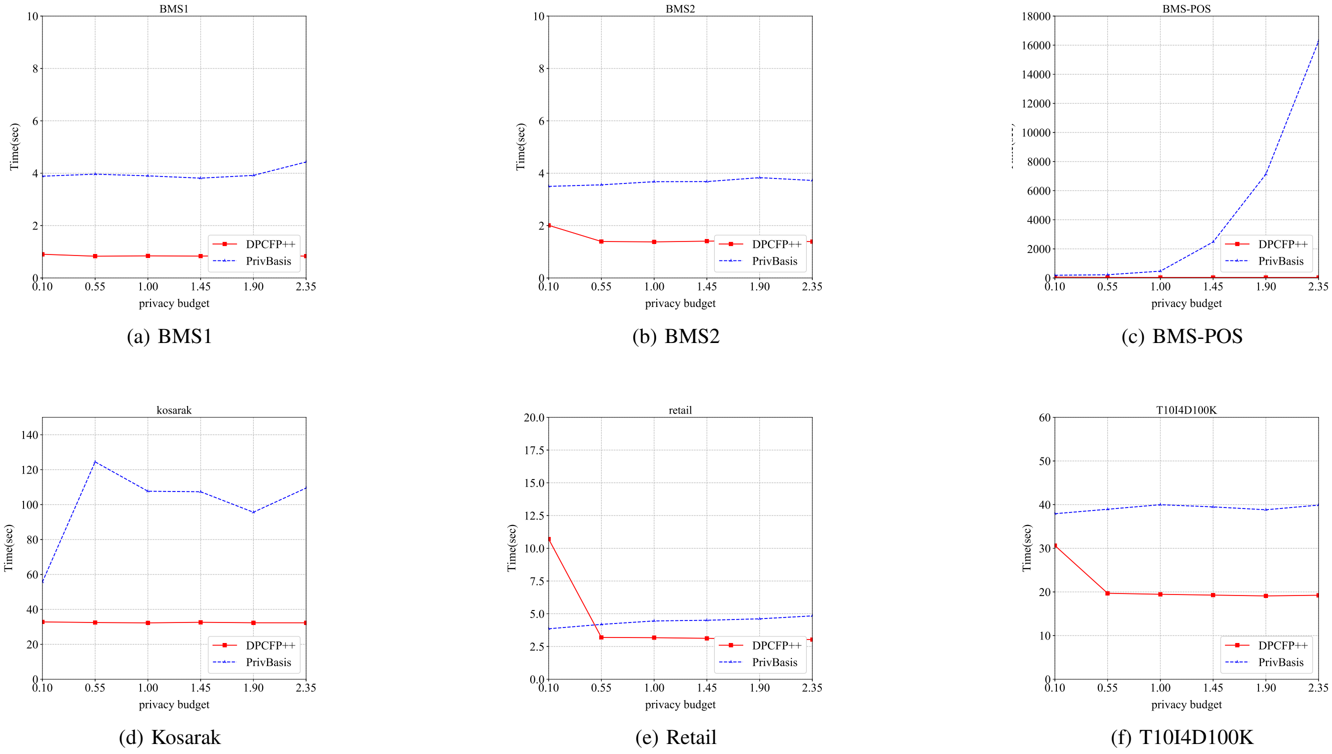
(c) BMS-POS

(d) Kosarak

(e) Retail

(f) T10I4D100K

Fig. 4. Recall



(a) BMS1

(b) BMS2

(c) BMS-POS

(d) Kosarak

(e) Retail

(f) T10I4D100K

Fig. 5. Runtime