Fast top-k frequent itemset mining under Local Differential Privacy*

*Note: Sub-titles are not captured in Xplore and should not be used

1st Wang JiaLi
dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address or ORCID

***Abstract*—This is the abstract.**

***Index Terms*—This is the keywords**

I. Introduction

Differential privacy (DP) [7] is the state-of-the-art approach that is used to protect individual privacy in the process of data collection, which has been named one of the world's top 10 breakthrough technologies in 2020 by the MIT technology review. It is a means in cryptography that aims to provide a way to maximize the accuracy of data queries when querying from statistical databases while minimizing the chances of identifying their records. Meanwhile, as a mathematical technique, it can add noise to the data while quantifying the extent of the increase in privacy, thus making the process of adding "noise" more rigorous.

Due to its unique advantages, DP has been widely studied by the academia and industry. For example, Google, Microsoft, apple and other companies use this technology to protect users' privacy, and at the same time, mobile phones aggregate data, so as to improve service quality. And the U.S. government is to complete a census of 330 million U.S. residents by 2020, keeping their identities secret, in what would be the largest application of DP ever.

There are two types of differential privacy - Centralized differential privacy (CDP) and Local differential privacy (LDP). Compared with CDP, the LDP does not require the assumptions of a trusted third party and provides stronger privacy guarantees. DP's research has involved many aspects, in recent years, the work in mining frequent itemsets has attracted the attention, which is one of the most important techniques because of its ability to locate the repeating relationships between different items in a data set and plays an essential role in mining association rules [9]. Formally, let $\mathcal{X} = \{x_1, x_2, ..., x_d\}$ be the global domain of items with size is d and $\mathcal{T} = \langle T_1, T_2, ..., T_n \rangle$ denote a transaction database for n users, where $T_i (i \in [1...n])$ denotes a transaction that is a subset of $\mathcal{X}$. For example, a sample of transational data is shown in Table I. The support of an itemset X, where $X \subseteq \mathcal{X}$ is a set of items, is the number of transactions containing X in $\mathcal{T}$. Then, given a minimum support threshold δ, the problem of finding the complete set of frequent itemsets that supports no less than δ is called the frequent itemset mining (FIM) problem.

Identify applicable funding agency here. If none, delete this.

TABLE I
Sample of transactional data.

TID	List of items
T01	a, f, c, g, p
T02	a, b, c, f, l, o
T03	b, f, h, o
T04	b, c, p
T05	f, a, c, l, p, n

A lot of work [3]–[6] has been done to solve DM problems in CDP. However, since the analyst holds the user's raw data in CDP setting, its main job is to add noise to the results to satisfy the DP definition.

In this paper, we consider the $top-k$ FIM problem in transaction databases under LDP. Although LDP is similar to CDP, the LDP has no reliance on third party. The data analyst wants to find k itemsets with highest support while while users are sensitive and unwilling to answer their real infomation. The main challenge is that the analyst does not hold the user's original sensitive information, which makes it quite difficult to mine useful information with sanitized data. Qin et al. [1] point out that if utlize directly existing FIM algorithm (e.g. Apriori [9], [10], FP-growth [8], Eclat [13]) would result in accumulation of dramatic noise because of multi-iteration between users and analyst.

Specifically for FIM in the local setting, Qin et al. [1] leave it as a future work but there is no clear solution. Wang et al. [2] solves the $top-k$ frequent itemset mining (FIM) task for the first time with **padding-and-sampling-based frequency oracle** (PSFO). In [2], the Set-Value Item Mining (SVIM) protocol is proposed to handles set values under LDP for the purpose of finding the k most frequent items and their frequencies. Meanwhile, to mine frequent itemsets, the core technique of Set-Value itemSet Mining (SVSM) is "Guessing Frequency (GF)" to construct the candidate set of itemsets.

Particularly, the analyst first calculates the guessing frequency of each candidate itemset. Then $2k$ itemsets with highest guessing frequencies are selected to construct candidate set IS, which is the domain of SVSM. Finally, it utilizes SVIM protocol again with the domain IS to identify $top-k$ itemsets. However, we observe that the number of possible itemsets to construct IS increases significantly with k. As a result, it is computationally expensive for guessing considerable frequencies when k is large (e.g., $k > 64$).

Inspiringly, to reduce the expensive computational cost caused by guessing frequencies of exponentially possible itemsets, we introduce the frequent-pattern tree (FP-tree) [8] structure to store compactly sensitive transactions of users, and then FP-growth [8] is awakened to mine frequent itemsets over the tree. Summarily, we propose Frequent-Pattern-based mine (fpmine) protocol to discover $top-k$ most frequent itemsets and their frequencies in the local setting. It has five steps. First, the SVIM protocol is used to estimate the k most frequent items and their frequencies. Second, users report the number of frequent items they have; the analyst estimates the distribution user reported and figure out the right M as the maximum iteration of the tree. Third, users interact with the analyst to build effectively the FP-tree [8]. Fourth, the analyst optimizes and mines the FP-tree. Fifth, the analyst publishes $top-k$ itemsets.

Experimental results show that minefp outperforms SVSM in that it identifies quickly frequent itemsets as well as estimates the frequencies more accurately. Notably, when k is large, it provides significantly lower computation overhead as well as similar accuracy than existing SVSM protocol .

To summarize, the main contributions of this paper are:

- We study the application of FP-growth algorithm and design the FP-tree-based-mine (minefp) protocol to find frequent itemsets as well as their frequencies in the LDP setting. Experimental results on real-world datasets show the significant improvement over previous techniques.
- We investigate GF to construct candidate set and point out that it is beneficial to build hierarchically FP-tree.

Roadmap.

II. Preliminaries

A. Local Differential Privacy (LDP)

In the local setting, there is no trusted third party and an aggregator wants to gather information from users, where each user possesses an input t. The privacy of the data contributor is protected by perturbing her/his original data at the data contributor's side; thus, the agregator cannot access the original data, but is still able to obtain population statistics.

Formally, let $\mathcal{T}$ denote the whole transaction databases. ϵ-local differential privacy (or ϵ-LDP) is defined on an algorithm $\mathcal{A}$ and a privacy budget $\epsilon \geq 0$ as follows.

Definition 1 *($\epsilon - LDP$). A randomized algorithm $\mathcal{A}$ satisfies ϵ-local differential privacy (ϵ-LDP), if and only if for (1) all pairs of input $t_i, t_j \in \mathcal{T}$, and (2) any possible output $\mathcal{O}$ of $\mathcal{A}$, we have:*

$$\frac{\mathbf{Pr}[\mathcal{A}(t_i) = \mathcal{O}]}{\mathbf{Pr}[\mathcal{A}(t_j) = \mathcal{O}]} \leq e^{\epsilon}$$

$Sequential\ composability$ [12] and $post-processing$ [14] are vitally important properties of differential privacy. The former allows each user to divide privacy budget into multiple portions and use each portion to execute independent LDP protocols on the same input while the sequential executions provide $\sum \epsilon_i$-LDP; the latter guarantees that any processing of the noisy data do not disclose the privacy.

Theorem II.1 ***(sequential composability).*** *Given m randomized algorithms $\mathcal{A}_i (1 \leq i \leq m)$, each of which satisfies ϵ_i-local differential privacy. Then the sequence of $\mathcal{A}_i$ collectively provides $(\sum_{i=1}^{m} \epsilon_i)$-local differential privacy.*

Theorem II.2 ***(post-processing).*** *For any method ϕ which works on the output of an algorithm $\mathcal{A}$ that satisfies ϵ-LDP without accessing the raw data, the procedure $\phi(\mathcal{A}(\cdot))$ remains ϵ-LDP.*

B. Frequency Oracle in the LDP setting

A frequency oracle (FO) protocol under LDP enables aggregator to estimate the frequency of any given value $t \in \mathcal{X}$ from all sanitized data recived from the users. A fundamental FO protocol is Randomized Response (RR) [15], which is a traditional technique for estimating unbiasedly a population proprotion. It is the building block of many sophisticated LDP protocols, such as RAPPOR [16], GRR and OLH [11]. Suppose the respondents were asked to answer a sensitive Boolean question (e.g. have you ever cheated on your partner?) in a survey, then RR makes provisions for each person to be interviewed. That is, each respondent gives the raw answer with probability p and gives the opposite answer with probabilit $q = 1 - p$. Specially, to satisfy ϵ-LDP, the probability p is set to $\frac{e^{\epsilon}}{1+e^{\epsilon}}$.

Then, the agregator calculates the estimated percentage of "Yes" (denote as $\tilde{\theta}$) as example from all sanitized answers, which is unbiased, as follows:

$$\tilde{\theta}_{RR}(Yes) = \frac{\mathcal{C}(answer = Yes) - nq}{p - q}$$

where n is the total number of respondents and $\mathcal{C}(answer = Yes)$ denotes the number of occurrences respondents answered "Yes". Accordingly, the variance of it is

$$Var[\tilde{\theta}_{RR}(Yes)] =$$

However, the RR protocol only applies to binary Boolean problems, which greatlt limits its application. Therefore, in [11], two effective protocols, Generalized Random Response (GRR) and Optimized Local Hash (OLH), are proposed for the purpose of solving problems with large domain size $d = |\mathcal{X}|$.

$Generalized\ Random\ Response\ (GRR)$ [11]: GRR extends the RR protocol in case of $d \neq 2$ by setting probability $p = \frac{e^{\epsilon}}{e^{\epsilon}+d-1}$ to give the raw answer $y = t$ and probability

$q = \frac{1-p}{d-1}$ (i.e. $q = \frac{1}{e^{\epsilon}+d-1}$) to give the perturbed answer $y \neq t$. Specially, it has been shown that RR is the special case while $d = 2$. The shortage of GRR is that its estimated variance is linear with d, which makes poor perform when d is large:

$$Var\left[\tilde{\theta}_{GRR}(t)\right] = n \cdot \frac{d - 2 + e^{\epsilon}}{(e^{\epsilon} - 1)^2} \tag{1}$$

Optimized Local Hashing (OLH) [11]: In order to deal with a large domain size d as well as reduce the communication cost, OLH protocol applies a hash function to map each input value into a value in $[g]$, where $g \geq 2$ and $g \ll d$. Then randomized response is used to the hashed value in the smaller domain. In [11], the optimal choice of the parameter g is $\lceil e^{\epsilon} + 1 \rceil$ which meets the minimal variance.

Let H is randomly chosen from a family of hash functions that outputs a value in $[g]$ and $x = H(t)$. The perturbing protocol in OLH is $Perturb_{OLH}(\langle H, x \rangle) = \langle H, y \rangle$, where

$$\forall_{i \in [g]} \mathbf{Pr}[y = i] = \begin{cases} p = \frac{e^{\epsilon}}{e^{\epsilon}+g-1}, & \text{if } x = i \\ q = \frac{1}{e^{\epsilon}+g-1}, & \text{if } x \neq i \end{cases}$$

Accordingly, the aggregator first calculates the number of perturbed values that "supports" that the input is t (denote as $\mathcal{C}(t)$), then transforms $\mathcal{C}(t)$ to its unbiased estimation

$$\tilde{\theta}_{OLH}(t) := \frac{\mathcal{C}(t) - n/g}{p - 1/g} \tag{2}$$

The variance of this estimation is

$$Var\left[\tilde{\theta}_{OLH}(t)\right] = n \cdot \frac{4e^{\epsilon}}{(e^{\epsilon} - 1)^2} \tag{3}$$

Intuitively, OLH has a variance that does not depend on d. However, the bigger d is, the more hash collisions there are, resulting in large errors. In [11], it suggests that when domain size $d < 3e^{\epsilon} + 2$, GRR is the best among all approaches; but for large d, OLH meets high accuracy as well as low communication cost.

C. *FP-growth algorithm*

Frequent pattern growth (FP-growth) [8] is an algorithm that mines the complete set of frequent patterns without a costly candidate generation process, which based on the frequent pattern tree (FP-tree) structure that is an extended prefix-tree structure for storing compressed, crucial information about frequent patterns. The FP-Tree is further divided into a set of Conditional FP-Trees for each frequent item so that they can be mined separately. An example of the FP-Tree that represents the frequent items is shown in Fig. 1, where the minimum support threshold is set to 3.

The FP-growth algorithm solves the problem of identifying long frequent itemsets by searching through smaller conditional FP-tree repeatedly. The conditional pattern base is a "sub-database" which consists of every prefix path in the FP-Tree that co-occurs with every frequent length-1 itemset. It is used to construct the conditional FP-tree and generate all the frequent patterns related to the length-1 items. In this way, the cost of searching for the frequent patterns is substantially reduced.

TABLE II
NOTATIONS.

Symbol	Description
n	the number of users
X	the itemset
$\lvert X \rvert$	the cardinality of itemset X
$\mathbb{G}(X)$	the guessing frequency of X
$\tilde{\theta}(X)$	the estimated frequency of X
$\mathcal{T}$	the transaction database
T_i	the transaction of user i
$\mathcal{X}$	the set of items
d	the number of items, $d = \lvert \mathcal{X} \rvert$
$\mathcal{N}$	the noisy FP-tree

Definition 2 *(length $- \alpha$ itemset). is this necessary?*

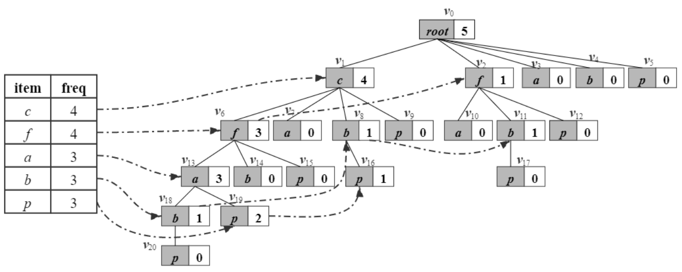

Fig. 1. Frequrnt pattern tree(FP-tree).

III. Problem Overview

A. *Problem Definition*

In this paper, we study the FIM problem while fully consider users' personal privacy. Or rather, we look at privacy preserving frequent itemset mining in the local setting. Formally, let $\mathcal{X} = \{x_1, x_2, ..., x_d\}$ be a set of items, the length-$|X|$ itemset X is a subset of $\mathcal{X}$, i.e.,$X \subseteq \mathcal{X}$, where $|X|$ denote its cardinality.

In general, there are n users, and each user has a sensitive transaction, which is a subset of $\mathcal{X}$. The transaction of i-th user is $T_i (i \in [1, n])$ and $\mathcal{T} = \langle T_1, T_2, ..., T_n \rangle$ denote the whole transaction database. An untrusted data analyst (or aggregator) is aimming to mine $top - k$ most frequent itemsets and their frquencies among all users under ϵ-LDP. More specifically, we focus on discovering k length-α itemsets with highest frequencies, where $\alpha \geq 2$, while discovering length-1 itemsets (or items) under LDP may take advantage of PSFO protocol. Table II lists the notations used in this paper

B. *Existing Approaches to FIM under LDP*

Since Qin et al. [1] first introduce set-valued data to differential privacy in the local setting for the purpose of discovering heavy hitters as well as their frequencies, it has been a challenge that mining frequent itemsets from all sanitized data. To the best of our konwledge, only SVSM [2] protocol that implements effectively FIM task in context of

LDP. Particularly, it focuses on mining $top-k$ most frequent length-α ($\alpha > 1$) itemsets while utilizing SVIM [2] protocol that obtains the frequent items (or length-1 itemsets) as initial condition to construct candidate set. Details as follows.

$Set-Value\ Item\ Mining\ (SVIM)$: SVIM is a PSFO protocol with the same problem setting as LDPMiner [1], which aimming at discovering the k frequent items with highest frequencies. It has four steps and divide proportionally all users into three groups G_1, G_2, G_3 so that each user is protected by ϵ-LDP.

1. Prune the Domain — G_1. The goal is to identify a candidate set for items so that it can reduce the domian size into $2k$, which is greatly less than the domain d. In this step, each user uses FO to perturb a randomly sampled item from his raw data. Then, the analyst estimates the frequency of each value in the domain and selects the $2k$ frequent items with highest frequencies. The analyst broadcasts the domain set S pruned to all users.

2. Size Estimation — G_2. To further estimate the frequency of the set S, it is necessary to approximate the distribution of the number of frequent items that users hold. Thus, the perturbing data of each user is the size of the raw data intersected with OLH protocol. After the perturbing of users finishes, the alalyst estimates the length distribution and calculates the $\gamma = 90\%$ length L by

$$\frac{\sum_{i=1}^{L} \tilde{\theta}(i)}{\sum_{i=1}^{2k} \tilde{\theta}(i)} > \gamma \tag{4}$$

3. Frequencies Estimation — G_3. Once the analyst gets S and L, it will use PSFO protocol to precisely estimate the frequencies of the items in small domian S. That is, each user in this step first pads his pruned data, which is the raw data intersected with S, to length L, then utilizes FO protocol for perturbing the item that randomly choose to report.

Finally, the analyst estimates the frequency of each item in S from all sanitized data. To ensure the estimation is unbiased, the estimated frequency needs to be multiplied by the length L.

4. Estimation Update. Due to the length is the 90th percentile length, which may lead to an underestimate. In order to improve the accuracy of the estimations, an update factor $u(L)$ is defined for correcting this under estimation. Practically, in this step, every frequency estimated is multiplied with the update factor,

$$u(L) := \frac{\sum_{i=1}^{2k} \tilde{\theta}(i)}{\sum_{i=1}^{2k} \tilde{\theta}(i) - \sum_{i=L+1}^{2k} \tilde{\theta}(i)(i-L)}$$

where $\tilde{\theta}(i)$ is the length distribution of length i estimated from step two.

Additionally, it is important to note that this step is the post-processing (Theorem II.2) in differential privacy and does not consume the privacy budget because informations are obtained from previous steps and there is no user involved.

When all the steps are done, the k items with highest frequencies are selected with high confidence.

$Set-Value\ itemSet\ Mining\ (SVSM)$: As mentioned above, SVSM protocol needs to know the set of $top-k$ frequent items, denoted by S'. Then, to construct candidate set of itemsets, the guessing frequency of each exponentially possible itemset that made up of items in S' is calculated by (5), and the domain set IS is constructed by selecting $top-2k$ itemsets with highest guessing frequencies.

Definition 3 *(GF). Let itemset X is a subset of a set of items $\{x_1, x_2, ..., x_m\}$, and the support of the item x_i is denoted by $\theta(x_i)$. The guessing frequency (GF) of X is $\mathbb{G}(X)$, defined as follows,*

$$\mathbb{G}(X) = \prod_{x \in X} \frac{\lambda \times \theta(x)}{\theta_{max}} \tag{5}$$

where θ_{max} is the maximum support of all items, $0 \leq \lambda \leq 1$ is a predifined parameter.

While the domain of candidate itemsets is pruned, it may use following steps of SVIM to discover frequent itemsets. In particular, every user's sensitive data in SVSM is the power set of his raw data intersected with the set S', which identified by SVIM.

introduce flaw of SVSM

Flaw. However, it is obvious that there are exponentially more possible itemsets during constructing candidate set. For example, such k items produce 2^k possible itemsets, leaving considerable computational cost in guessing frequencies. Although it is persuasive that the cardinality of frequent itemset would not be greater than $\log_2 k$, the computation space $O\left(\sum_{i=2}^{\lfloor \log_2 k \rfloor} \binom{k}{i}\right)$ is not acceptable when k is large.

IV. FPMINE: THE FP-TREE-BASED APPROACH

In this section, we propose a novel FP-tree-based mining approach called fpmine that discovers the $top-k$ most frequent itemsets under LDP with low computational cost as well as high accuracy. The main idea is to constrcut a noisy FP-tree, and then discover itemsets over the tree. In the following, Section IV-A overviews the framework of fpmine. Section IV-B describes the details of constructing and mining the nosiy FP-tree. Sections IV-C and IV-D further optimize the proposed approach, respectively.

A. fpmine

As described in Section III-B, the existing solutions incur considerable computational cost duo to the exponential growth of candidate itemsets. We observe that the original FP-growth algorithm can mines frequent itemsets without a costly candidate generation process. However, in the local setting, the analyst's inability to access users' raw data makes it difficult to constrcut a noisy FP-tree. Note that if we can reduce the noise introduced, then we can efficiently reduce the computational cost as well as achieve better accuracy. Algorithm 1 presents the procedure of FPmine.

Specifically, FPmine first constructs a noisy FP-tree in a breadth-first manner for the purpose of storing informations without compromising privacy, and then mines frequent itemsets by pattern fragment growth. Note that the domain size d of items might be large in practice and only a combination of frequent items are possible to be a frequent itemset. Therefore, it is necessary and meaningful to identify the k frequent items, which may significantly cut down the size of itemsets ($2^k \ll 2^d$). In short, FPmine works as follows: First, a set of frequent items are identified. Second, approximate the maximum number of frequent items that user holds. Third, a noisy FP-tree is constructed in a breadth-first manner and then discover frequent itemsets.

Algorithm 1 fpmine($\mathcal{T}, \mathcal{X}, k, \epsilon$)

```
Input: transactional database $\mathcal{T}$, set of items $\mathcal{X}$, top $k$, privacy budget $\epsilon$
Output: a set of frequent itemsets $\tilde{\mathcal{P}}$
 1: Randomly divide $\mathcal{T}$ into three groups $G_1, G_2, G_3$;
 2: Collect the items set $S' \leftarrow SVIM(G_1, k, \epsilon)$;
 3: Each user in group $G_2$ perturbs the number of frequent items he holds to analyst with OLH($\epsilon$);
 4: $\mathcal{L} \leftarrow$;
 5: for length $l = 1$ to $k$ do
 6:     Compute the frequency $\tilde{\theta}(l)$;
 7:     if $\tilde{\theta}(l) < \mathcal{L}$ then
 8:         $\tilde{\theta}(l) \leftarrow 0$;
 9:     end if
10: end for
11: Compute $L_m$ by (4) where $\gamma = 80\%$;
12: $\mathcal{N} \leftarrow ConstructNoisyTree(G_3, S', L_m, \epsilon)$;
13: Mine the $k$ itemsets $\tilde{\mathcal{P}}$ with highest frequencies;
14: return $\tilde{\mathcal{P}}$
```

As shown in algorithm 1, given the privacy budget ϵ, similar to SVSM is that we divide all users into three groups G_1, G_2, G_3 to participate in each step (line 1), which provides ϵ-LDP for each user. That is, the first step is aiming to identify the set S' of k frequent items from users in G_1 with SVIM protocol (line 2). And then the second is to approximate the maximum number of frequent items that user holds (line 3-11). In this step, the 80th percentile length L_m is calculate by (4) while the differences are that we set $\gamma = 80\%$ (line 11) and initialize a threshold $\mathcal{L} =$ (line 4) to filter the estimates. Finally, on receiving S' and L_m, the noisy FP-tree $\mathcal{N}$ is constrcuted by ConstructNoisyTree (Algorithm 2 in Section IV-B), and then the analyst can discover accurate itemsets by pattern fragment growth (line 13), which follows the procedure of the FP-growth except accumulating frequencies of conditional patter bases as explained in Section IV-B.

B. Constructing and mining the noisy FP-tree

In the local setting, the main challenge of constructing a noisy FP-tree is that the analyst has no right to access users' raw data, and that leads to introduce considerable noise when the tree is constructed. In [8], the non-privacy FP-tree is constructed by scanning unconcealed transactions and updating nodes with frequent items that user holds in a depth-first manner. We observe that it enables to construct the FP-tree in a breadth-first manner if the final count of each node at same level is known in advance. Inspired by this, we propose a breadth-first approach called ConstructNoisyTree to costruct a noisy FP-tree layer by layer. Specifically, let L_m is the maximum length, and $x_1 \succ x_2 \succ \cdots \succ x_k$ is the ordered sequence of k frequent items, denoted by S', where $x_i \succ x_{i+1}$ means that x_i is more frequent than x_{i+1} and x_i comes before x_{i+1}. The ConstructNoisyTree has two phase:

TABLE III
PREPROCESSED TRANSACTIONAL DATA.

TID	List of items
T01	c, f, a, p
T02	c, f, a, b
T03	f, b
T04	c, b, p
T05	c, f, a, p

Phase I Preprocessing. All participating users first delete infrequent items in their transactions and then rearrange the rest in order of S'. For example, the frequent items in Table I is sorted to $c \succ f \succ a \succ b \succ p$ while the minimum support threshold is set to 3, and then the preprocessed transactional data is shown in Table III.

Additionally, after the preprocessing step finishes, there are many infrequent items discarded, which significantly cuts down the size of candidate sets and achieves good performance because of the Apriori property [10]:*only if the length* $-\alpha$ *itemset is frequent are its length* $-(\alpha+1)$ *supersets likely to be frequent.*

Phase II Constructing the FP-tree In contrast to FP-growth, we propose a breadth-first approach to construct the FP-tree layer by layer in the local setting. Details described in Algorithm 2. Note that we omit the mining procedure duo to the fact that it is similar to the original algorithm and remark on the differences.

- A node v in the FP-tree has two fields: $v.item$ and $v.count$, where denote the indicated item of node v and the number of its prefix $pre(v)$ in transactions respectively.
- The informations of the S' and L_m are obtained in a way that satisfies the ϵ-LDP. The formar is used to arrange users' frequent items and the latter marks the end of constructing the FP-tree. This ensures that the noisy FP-tree does not depend on any specific transaction.
- We construct the FP-tree in a breadth-first manner, which is unlike to the original algorithm, that is, the information is collected for updating the tree nodes layer by layer. Notably, the input value of each user in l-th level is either an element of the candidate prefix set C_l or the dummy value $\dagger$. Therefore, we only use the existing FO protocol (e.g. OLH) with the finite domain $C_l \cup \dagger$ to collect information at each level.
- We randomly divide users into L_m equal-sized groups and users in each group use the full privacy budget ϵ. It

turns out that the overall has better accuracy and satisfies ϵ_2-LDP as well ([11], [17]).

- Let X denote a discovered itemset and its estimated frequency $\tilde{\theta}(X)$ is unbiased. Note that $\tilde{\theta}(X)$ is obtained from m conditional patter bases $B_1, B_2, ..., B_m$. Unlike the original algorithm to add directly, the final $\tilde{\theta}(X)$ is calculated in a way that minimize its variance as in Theorem IV.1.
  —–!!Notice whether it's different from a linear combination!!—–

Algorithm 2 ConstructNoisyTree(G_3, S', L_m, ϵ)

```
1: Randomly and evenly divide users into $L_m$ groups $g_1, g_2, ..., g_{L_m}$ with $n_g = \lfloor \frac{n}{L_m} \rfloor$ users each;
2: Initialize the tree $\mathcal{N}$ with a root node $v_r$;
3: Initialize the count of $v_r$ is $n_g$ and mark it as valid;
4: Preprocess transactions of all participating users;
5: for $l = 1$ to $L_m$ do
6:     while there exists a valid node $v$ and $v.count > 0$ do
7:         Initialize a candidate prefix set $C_l = \emptyset$;
8:         Mark $v$ as unvalid;
9:         for each item $x \in S'$ and $v.item \succ x.item$ do
10:            Add a child $v_c$ of $v$ with item is $x$ and count is 0;
11:            Mark $v_c$ as valid and compute its prefix $pre(v_c)$;
12:            $C_l \leftarrow C_l \cup pre(v_c)$;
13:        end for
14:    end while
15:    if $C_l$ is $\emptyset$ then
16:        Break;
17:    end if
18:    for each users in group $g_l$ do
19:        Perturbe his first $l$ items with OLH protocol for collecting and updating an estimation count $\tilde{\theta}(v)$ of each node $v$, whose prefix is in $C_l$;
20:    end for
21:    Update nodes at $l$-th level of $\mathcal{N}$;
22: end for
23: return The noisy tree $\mathcal{N}$.
```

For example, lets consider the processed transactions in Table III, where the frequnet items and their supports is denoted by $S' = [c:4, f:4, a:3, b:3, p:3]$ (for simplicity, we use the real support below). Then, the root node v_0 is initialized firstly as the 0-th level of the tree $\mathcal{N}$ (line 2-3) and the constructing process is as follows.

At the 1-st level, the children v_1, v_2, v_3, v_4, v_5 of v_0 are added to form the candidate node at the 1-st level of the tree, which initialized each count is zero (line 10). And then according to the candidate prefix set $C_1 = \{(c), (f), (a), (b), (p)\}$, their estimated informations are collected for the purpose of updating nodes' counts (line 18-21), that is, the counts of v_1, v_2 are updated to 4 and 1 while v_3, v_4, v_5 are pruned according to the collected information $\{(c,4), (f,1), (a,0), (b,0), (p,0)\}$. After the 1-st level are updated, the candidate set $C_2 = \{(c,f), (c,a), (c,b), (c,p), (f,a), (f,b), (f,p)\}$ at the 2-nd level is constructed. Then it can update the tree nodes in the same way while the transaction of each user is the first two items in his pre-processed data (e.g. the input value of first user is (c,f)). BSmine continues to the next iteration until the tree reaches its maximum level L_m or there are no node is valid. The final FP-tree is constructed as show in Fig. 1.

Theorem IV.1 *Given m noisy counts $\tilde{\theta}(b_1), \tilde{\theta}(b_2), ..., \tilde{\theta}(b_m)$ with variances of $r_1, r_2, ..., r_m$. The variance of $X = \sum_{i=1}^{m} \omega_i \tilde{\theta}(b_i)$ with constrain $\sum_{i=1}^{m} \omega_i = m$ is minimized by setting $\omega_i = m \cdot \frac{r_i^{-1}}{\sum_{j=1}^{m} r_j^{-1}}$.*

Proof. *Obviously, with $\sum_{i=1}^{m} \omega_i = m$, finding the minimized variance of $X = \sum_{i=1}^{m} \omega_i \tilde{\theta}(b_i)$ is equivalent to solve*

$$\min \sum_{i=1}^{m} \omega_i r_i, s.t. \sum_{i=1}^{m} \omega_i = m \quad (6)$$

Then, we have the Lagrangian function

$$\mathcal{L} = \sum_{i=1}^{m} \omega_i r_i + B(m - \sum_{i=1}^{m} \omega_i)$$

Let $\frac{\partial \mathcal{L}}{\partial \omega_i} = 2r_i\omega_i - B$ be zero, we have $\omega_i = \frac{B}{2r_i}$. And according to the condition $\sum_{i=1}^{m} \omega_i = m$, we get the constant $B = \frac{2m}{\sum_{i=1}^{m} r_i^{-1}}$.

Finally, we have the optimal solution is $\omega_i = m \cdot \frac{r_i^{-1}}{\sum_{j=1}^{m} r_j^{-1}}$.

Lemma IV.1 *Algorithm 2 satisfies ϵ-LDP.*

Proof. *proof Algorithm satisfies LDP.*

Lemma IV.2 *For any node v in the tree, let $\tilde{\theta}(v)$ denote its estiamted count obtained by the algorithm 2 and $\theta(v)$ denote the ground truth. With at least $1 - \beta$ probability, we have:*

$$|\tilde{\theta}(v) - \theta(v)| = O\left(\frac{L_m \cdots}{\epsilon}\right)$$

C. Narrowing the cadidate prefix set

We observer that the biggest limitation of ConstructNoisyTree is that the size of candidate prefix set C_l constructed during l-th iteration is indisciplinable and large. Deu to C_l is the prefixes domian of OLH to gather estimations, the large size affects not noly accuracy but also computational cost. Observe that the size of C_l is linearly related to nodes reserved in the upper $(l-1)$-th level. That is, the fewer nodes are reserved in $(l-1)$-th level, the fewer children are generated in l-th level, which makes up small C_l. However, it is inefficient since reserving fewer nodes may result in information loss and underestimation.

In view of this, we set a threshold $\mathcal{L}'$ to remove as many invalid nodes as possible while retain effective estimations. Formally, let v is a node of the tree while $v.count$ is initialized to zero before estimation. Once we obtain an estimated count $\tilde{\theta}(v)$ of its prefix $pre(v)$, $\mathcal{L}'$ is used to determine if it is valid. Specifically, v is valid when $\tilde{\theta}(v) > \mathcal{L}'$ and invaild otherwise. Finally, we update $v.count = \tilde{\theta}(v)$ when v is valid and $v.count = 0$ when it is invalid. It is noted that the larger

$\mathcal{L}'$, the fewer invalid estimations, but more valid values may be lost. In this paper, we set $\mathcal{L}' =$, which is the maximum error boundary of OLH.

—-introduce $\mathcal{L}'$—-

Sequentially, it is obvious that there are many redundant prefixes in C_l. For example, at the 3-rd level of the tree in Fig. 1, the initial set C_3 is made up of prefixes of $v_6, v_7, v_8, v_9, v_{10}, v_{11}, v_{12}$. However, as the information is collected, four invalid nodes v_7, v_9, v_{10}, v_{12} are removed. Recall from users' transactions that there are no prefixes for these nodes. Therefore, if we can remove beforehand as many of these as possible, then the size will be significantly narrowed.

Similar to SVSM [2] is to limit the candidate set to a fixed size. We propose NarrowCandidate (Algorithm 3) to further narrow the set C_l down to fixed size $\xi \cdot k \ll |C_l|$, where ξ is a predefined adiustable parameter. Specifically, when the size is greater than $\xi \cdot k$, we first guess each candidate prefix in C_l. And then select the $top - \xi k$ prefixes with highest guessing frequencies to construct a pony-size set C'_l. Note that each prefix in C_l has the same cardinality and we just need to get their relative order. Therefore, we use the simplified guessing frequency $\mathbb{G}'(P)$ of prefix $P \in C_l$ to construct C'_l,

$$\mathbb{G}'(P) = \prod_{x \in P} \tilde{\theta}(x) \tag{7}$$

where $\tilde{\theta}(x)$ is the frequency in S' of item x.

Algorithm 3 NarrowCandidate(S', C_l, ξ)

```
1: Initialize $d_c \leftarrow C_l$;
2: Initialize $C'_i = \emptyset$;
3: if $d_c > \xi k$ then
4:     for each candidate prefix $P \in C_l$ do
5:         Guess the frequency $\mathbb{G}(P) = \prod_{x \in P} \tilde{\theta}(x)$ of $P$;
6:     end for
7:     Construct $C'_l$ by selecting the $\xi k$ prefixes in $C_l$ with
       highest guessing frequencies;
8: else
9:     $C'_l \leftarrow C_l$
10: end if
11: return $C'_l$
```

D. Imposing consistency and weighted combination

Consistency. Without loss of generality, let $\tilde{\theta}(v)$ is the noisy count of a node v in the tree that constructed, and $\theta(v)$ is its ground-truth count. For each node v, we have the following constraints

- The noisy count $\tilde{\theta}(v)$ is unbiased, and its expectation satisfies $\mathbb{E}[\tilde{\theta}(v)] = \mathbb{E}[\theta(v)]$.
- If v has a set S_c of children, then

$$\mathbb{E}[\tilde{\theta}(v)] \geq \sum_{v_c \in S_c} \mathbb{E}[\tilde{\theta}(v_c)]$$

— introduce a solution or cite a paper —

Weighted combination. In order to obtain better accuracy, we propose the weighted combination, which satisfies post-processing property. Specifically, we weight the results that discovered through the tree with guessing frequencies, and then release the $top - k$ most frequent itemsets. Let $\tilde{P}$ denote the set of itemsets that discovered throught the tree. For each itemset X in $\tilde{P}$, we have its weighted frequency $\Omega(X)$.

$$\Omega(X) = \omega \cdot \tilde{\theta}(X) + (1 - \omega) \cdot \mathbb{G}(X) \tag{8}$$

Then, the k itemsets is selceted as

V. Ease of Use

A. Maintaining the Integrity of the Specifications

The IEEEtran class file is used to format your paper and style the text. All margins, column widths, line spaces, and text fonts are prescribed; please do not alter them. You may note peculiarities. For example, the head margin measures proportionately more than is customary. This measurement and others are deliberate, using specifications that anticipate your paper as one part of the entire proceedings, and not as an independent document. Please do not revise any of the current designations.

References

[1] Qin, Zhan, et al. "Heavy Hitter Estimation over Set-Valued Data with Local Differential Privacy." computer and communications security (2016): 192-203.

[2] Wang, Tianhao, Ninghui Li, and Somesh Jha. "Locally Differentially Private Frequent Itemset Mining." ieee symposium on security and privacy (2018): 127-143.

[3] Bhaskar, Raghav, et al. "Discovering frequent patterns in sensitive data." knowledge discovery and data mining (2010): 503-512.

[4] Li, Ninghui, et al. "PrivBasis: frequent itemset mining with differential privacy." very large data bases (2012): 1340-1351.

[5] Lee, Jaewoo, and Chris Clifton. "Top-k frequent itemsets via differentially private FP-trees." knowledge discovery and data mining (2014): 931-940.

[6] Zeng, Chen, Jeffrey F. Naughton, and Jinyi Cai. "On differentially private frequent itemset mining." very large data bases (2012): 25-36.

[7] C. Dwork. Differential privacy. In ICALP, pages 1–12, 2006.

[8] Han J, Pei J, Yin Y, et al. Mining frequent patterns without candidate generation[C]. international conference on management of data, 2000, 29(2): 1-12.

[9] Agrawal R, Imielinski T, Swami A N, et al. Mining association rules between sets of items in large databases[C]. international conference on management of data, 1993, 22(2): 207-216.

[10] Agrawal R, Srikant R. Fast Algorithms for Mining Association Rules in Large Databases[C]. very large data bases, 1994: 487-499.

[11] Wang T, Blocki J, Li N, et al. Locally Differentially Private Protocols for Frequency Estimation[C]. usenix security symposium, 2017: 729-745.

[12] F. D. McSherry. Privacy integrated queries: an extensible platform for privacy-preserving data analysis. In SIGMOD, pages 19–30, 2009.

[13] Zaki M J. Scalable algorithms for association mining[J]. IEEE Transactions on Knowledge and Data Engineering, 2000, 12(3): 372-390.

[14] C. Dwork, F. McSherry, K. Nissim, and A. Smith, "Calibrating noise to sensitivity in private data analysis," in Theory of Cryptography Conference. Springer, 2006, pp. 265–284.

[15] S. L. Warner. Randomized response: A survey technique for eliminating evasive answer bias. Journal of the American Statistical Association, 60(309):63–69, 1965.

[16] U. Erlingsson, V. Pihur, and A. Korolova. Rappor: Randomized aggregatable privacy-preserving ordinal response. In CCS, pages 1054–1067. ACM, 2014.

[17] Wang N, Xiao X, Yang Y, et al. PrivTrie: Effective Frequent Term Discovery under Local Differential Privacy[C]. international conference on data engineering, 2018: 821-832.

[18] Q. Ye, H. Hu, X. Meng and H. Zheng, ”PrivKV: Key-Value Data Collection with Local Differential Privacy,” 2019 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 2019, pp. 317-331, doi: 10.1109/SP.2019.00018.

[19] 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 ICDE. IEEE, 2017.

[20] J. Tang, A. Korolova, X. Bai, X. Wang, and X. Wang. Privacy loss in apple’s implementation of differential privacy on macos 10.12. arXiv preprint 1709.02753, 2017.

[21] G. Cormode, S. Jha, T. Kulkarni, N. Li, D. Srivastava, and T. Wang. Privacy at scale: Local differential privacy in practice. In SIGMOD, pages 1655–1658. ACM, 2018.

[22] B. Ding, J. Kulkarni, and S. Yekhanin. Collecting telemetry data privately. In NIPS, pages 3574–3583, 2017.