Top-*k* Frequent Itemsets via Differentially Private FP-trees

Jaewoo Lee
Dept. of Computer Science / CERIAS
Purdue University
West Lafayette, IN USA
jaewoo@cs.purdue.edu

Chris Clifton
Dept. of Computer Science / CERIAS
Purdue University
West Lafayette, IN USA
clifton@cs.purdue.edu

ABSTRACT

Frequent itemset mining is a core data mining task and has been studied extensively. Although by their nature, frequent itemsets are aggregates over many individuals and would not seem to pose a privacy threat, an attacker with strong background information can learn private individual information from frequent itemsets. This has lead to differentially private frequent itemset mining, which protects privacy by giving inexact answers. We give an approach that first identifies top-k frequent itemsets, then uses them to construct a compact, differentially private FP-tree. Once the noisy FP-tree is built, the (privatized) support of all frequent itemsets can be derived from it without access to the original data. Experimental results show that the proposed algorithm gives substantially better results than prior approaches, especially for high levels of privacy.

Categories and Subject Descriptors

H.2.0 [**Information Systems**]: Database Management

General Terms

Algorithms, Security, Theory

Keywords

Frequent itemset, Differential privacy, FP-tree

1. INTRODUCTION

As volumes of personal data collected by many organizations increase, the problem of preserving privacy is increasingly important. In this paper, we investigate the problem of releasing top k frequent itemsets in a differentially private way. The problem of frequent itemset mining (FIM) has been extensively studied in the data mining community[2, 9] and it serves as an important building block for many exploratory data mining algorithms. Although frequent itemsets are inherently aggregated information common to many individuals, naïve release of frequent itemsets can reveal sensitive information about individuals in the dataset[3].

Differential privacy[7, 8] is a recent notion of privacy that guarantees the output of an algorithm is insensitive to the change of one individual's data. Differential privacy adds noise to the output (in our case, the frequent itemsets and their supports) such that the impact of any individual (transaction) on the outcome is small relative to the added noise - ensuring that even an adversary with extensive background knowledge cannot use the result to determine an individual's value. The main advantage of differential privacy is that it provides protection against a strong adversary without requiring a model of the adversary's knowledge, or defining what information is sensitive.

The main challenge in developing a differentially private FIM algorithm is that the sensitivity of an itemset query (the maximum impact a single individual can have on the set of frequent itemsets) is dependent on the dimensionality of itemsets in the dataset. Recently two works on differentially private FIM, PrivBasis[15] and SmartTruncation[19], were introduced. They reduce the dimensionality by projecting the data into a lower dimensional space and by truncating the transactions, respectively. While these methods guarantee differential privacy, they come at a substantial cost in the quality of the results.

In this paper, we propose a novel approach that integrates the low sensitivity of a threshold query set and the compactness of the FP-tree data structure[9]. The proposed algorithm uses a modified version of the general *sparse vector technique* introduced in [11, 10]. The sparse vector technique was originally used to release a small number of count queries whose count is greater than or equal to the given threshold. A variation of it appears in [17]. A big advantage of this technique is that information disclosure affecting differential privacy occurs only for count queries above the threshold; negative answers do not count against the "privacy budget". Intuitively, the itemsets whose supports are far above (below) the threshold remain frequent (infrequent) even after adding or removing one transaction. Only those itemsets whose supports are near the threshold can be switched from frequent to infrequent or vice versa by the change of single transaction. Based upon this observation, we modify the sparse vector technique to privately answer exponentially many threshold queries (with a constant number of non-null results) given a fixed threshold. The proposed algorithm is composed of two phases: (i) frequent itemset discovery and (ii) noisy support derivation. In the first phase, all frequent itemsets are identified. At

KDD'14, August 24–27, 2014, New York, NY, USA.
Copyright 2014 ACM 978-1-4503-2956-9/14/08 ...$15.00.
http://dx.doi.org/10.1145/2623330.2623723.

this stage, the algorithm does not know their supports, but only that their supports are above the threshold. Using this information, the second phase builds a differentially private FP-tree with privatized supports of the frequent itemsets. The proposed algorithm injects noise in the data structure at the intermediate (FP-tree) step. The final output (Frequent Itemsets, or any other output derivable from an FP-tree) can be further refined through an optional post-processing step.

The rest of this paper is organized as follows: In Section 2, related works are discussed. Section 3 defines the notations used throughout the paper and introduces the background knowledge about differential privacy. In Section 4, given the threshold, how the proposed algorithm identifies the set of frequent itemsets and derives their supports via differentially private FP-tree are described. In Section 5, the performance of the proposed algorithm is evaluated on a variety of datasets. Finally, Section 6 concludes our work.

2. RELATED WORKS

There has been concern for some time that data mining results could lead to privacy violations [13]. In [1], Aggarwal et al. claimed that in practice it is possible for an adversary to predict sensitive fields in records by using association rules learned from the dataset. Therefore, it is insufficient to simply hide sensitive values in the dataset. They introduced a technique for hiding minimal set of association rules to prevent disclosure of sensitive entries in the dataset. In order to hide sensitive association rules, the input dataset is anonymized so that sensitive rules cannot be learned.

Atzori et al. applied the concept of k-anonymity to the mining result instead of input dataset. To prevent inference on sensitive information, frequent patterns are distorted by changing their support values [3]. A collection of frequent patterns is called k anonymous if and only if the support of each pattern in the collection is greater than or equal to k.

The common problems from which both approaches described above suffer are difficulty on determining sensitive information and on modeling an adversary's background knowledge. Differential privacy, proposed in [8], has emerged as a promising solution for privacy protection and received considerable attention because of its privacy guarantee against an adversary with arbitrary background knowledge.

Bahaskar et al. studied the problem of differentially privately releasing top k frequent itemsets of length m [4]. Given the mining result of a non-private algorithm, it first selects top k itemsets to release using either the Laplace or exponential mechanism and then adds noise to each itemset's support value.

Li et al. proposed a method called *PrivBasis*[15], that finds top k frequent itemsets in a differentially private way. PrivBasis tries to reduce the search space by finding minimal sets of items in an initialization stage, that covers the top k frequent itemsets. Each such set is called a basis. The itemsets in transactional data are viewed as tabular data and projected onto each basis. The supports of all subsets of a basis are derived by using binary itemset support counting[5]. To find a basis set, PrivBasis employed the technique used in MaxClique[18]. However, basis finding based on frequent 2-itemsets may be very inaccurate and tends to generate many more bases than actually required.

Zeng et al. built a differentially private FIM algorithm[19] on apriori. Inspired by the observation that the sensitivity of a counting query set used for support verification is dependent on the maximal length of transactions in the dataset, they introduced the idea of truncating transactions to reduce the amount of noise to inject. However, truncating transactions in their algorithm is heuristic and its performance is quite sensitive to parameter settings.

Symbol	Description
$D = \{t_1, \cdots, t_n\}$	A database of transactions
$\mathcal{I} = \{I_1, \cdots, I_p\}$	Set of items
ℓ-itemset	An itemset of length ℓ
$\mathsf{Lap}(\lambda)$	Laplace distribution with mean 0 and scale factor λ
$\sigma(X)$	Support of X, $\lvert\{t \in D \mid X \subseteq t\}\rvert$
$\widehat{\sigma}(X)$	Noisy support of X, $\sigma(X) + \mathsf{Lap}(\cdot)$
τ	A user defined minimum support
$\mathcal{L}_i$	Set of large i-itemset
$\mathcal{C}_i$	Set of candidate i-itemset
$\mathcal{M}$	Set of maximal frequent itemsets
$v^{<t}$	Prefix of vector $\mathbf{v}$ of length $t-1$

Table 1: Notations

3. PRELIMINARIES

3.1 Notation

Let $\mathcal{I} = \{I_1, \cdots, I_p\}$ be a set of items. An ℓ-itemset $X = \{x_1, \cdots, x_\ell\}$ is a subset of $\mathcal{I}$ whose length is ℓ where $x_i \in \mathcal{I}(1 \leq i \leq \ell)$. The support of itemset X, denoted by $\sigma(X)$, is the number of transactions in D that includes X as a subset. An itemset X is frequent if and only if its support is greater than or equal to minimum support τ (i.e.,if $\sigma(X) \geq \tau$). We use $\widehat{Y}$ to denote the noisy version of Y, i.e., $\widehat{Y} = Y + \mathsf{Lap}(\lambda)$ where $\mathsf{Lap}(\lambda)$ is a random sample (i.i.d) drawn from a Laplace distribution whose scale factor is λ.

3.2 Differential Privacy

Two databases D_1 and D_2 are referred to as neighboring if one can be obtained by adding or removing one tuple from the other, i.e., $|(D_1 - D_2) \cup (D_2 - D_1)| = 1$. Informally, differential privacy ensures that any changes in the probability of any outcome due to a single change in the input database is bounded by a constant ratio.

Definition 1 (ϵ-differential privacy). *A randomized mechanism $\mathcal{K}$ is ϵ-differentially private if for all neighboring databases D_1 and D_2, $\forall S \subseteq Range(\mathcal{K})$*

$$\mathrm{P}[\mathcal{K}(D_1) \in S] \leq \exp(\epsilon)\,\mathrm{P}[\mathcal{K}(D_2) \in S]$$

One way to achieve differential privacy is to perturb the output with random noise. The magnitude of noise should be large enough to hide the change that can be made by one individual in the universe. This is captured by the concept of sensitivity [8].

Definition 2 (Sensitivity). *The sensitivity of a query function q is defined as*

$$\Delta q = \max_{D_1, D_2} |q(D_1) - q(D_2)|$$

where D_1 and D_2 are neighboring databases.

It is known that Laplace mechanism achieves ϵ-differential privacy [7].

Definition 3 (Laplace mechanism). *Given a query function q, a mechanism $\mathcal{K}$ that adds a random noise drawn from $\mathsf{Lap}\left(\frac{\Delta q}{\epsilon}\right)$ to the output value of q is referred to as Laplace mechanism.*

One nice property of differential privacy is that it is composable. The privacy guarantee provided by differential privacy gracefully degrades under sequential composition: any sequential composition of differentially private sub-routines, each of which satisfies ϵ_i-differential privacy, satisfies $\sum_i \epsilon_i$-differential privacy. This is useful when designing a differentially private algorithm since the entire algorithm will be ϵ-differentially private as long as the privacy budget allocated to each sub-routine sums to ϵ.

3.3 Sparse Vector Algorithm

The sparse vector algorithm was developed to release κ count queries that are above the given threshold τ. The algorithm starts by calculating the noisy threshold $\hat{\tau}$ using part of the privacy budget. Given a count query q, it calculates the noisy count and compares it with $\hat{\tau}$. Let $\mathcal{K}$ denote the algorithm and q be a count query.

$$\mathcal{K}(D) = \begin{cases} q(D) + \mathsf{Lap}\left(\frac{2\kappa}{\epsilon}\right) & \text{if } q(D) + \mathsf{Lap}\left(\frac{2\kappa}{\epsilon}\right) \geq \hat{\tau} \\ \perp & \text{otherwise} \end{cases}$$

The algorithm uses another half of the privacy budget to answer above threshold queries. Notice that the remaining privacy budget $\frac{\epsilon}{2}$ is divided into κ count queries. After answering κ count queries it halts. The nice property of this algorithm is that it only pays privacy budget for above threshold queries and all below threshold queries are answered without wasting the privacy budget. Hence, any number of below threshold queries can be answered without compromising privacy. In essence, although the noisy count is calculated for each query and it is compared to the noisy threshold $\hat{\tau}$, the algorithm does not release it but $\perp$ for all below threshold queries hence not paying the privacy budget. The privacy guarantee of above threshold queries is immediate since each query pays the budget of $\frac{\epsilon}{2\kappa}$. The privacy of below threshold queries comes from the fact that it is compared to the noisy threshold. The formal privacy proof appears in [10].

4. THE NOISYCUT ALGORITHM

We now describe each step of the proposed algorithm, called NoisyCut. The algorithm takes an integer k and output the top k most frequent itemsets. The algorithm first learns the support of the k^{th} most frequent itemset σ_k and uses it as a threshold. This can be done by running any non-private FIM algorithm. Since the only output of this stage is the σ_k, using only the noisy threshold $\hat{\tau} = \sigma_k + \mathsf{Lap}(\cdot)$ ensures differential privacy.

For each itemset in the itemset lattice, the algorithm tests if its noisy support is above the noisy threshold. If it is, the algorithm regards it as frequent. This can be viewed as cutting the itemset lattice into two groups, frequent $\mathcal{L}$ and infrequent $\mathcal{L}^{\mathsf{C}}$, and hence the name of algorithm. We will show that this can be done with high accuracy and with small privacy budget while guaranteeing differential privacy. After identifying all frequent itemsets, to derive the support of each frequent itemset, the algorithm builds a noisy FP-tree. The supports of itemsets can be derived from the FP-tree

Algorithm 1 FindFreqItemsets

```
Input: Transactional database $D$, set of items $\mathcal{I}$, Top $k$,
    privacy budget $\epsilon_1$
Output: a set of frequent itemsets $\mathcal{L}$
 1: function FindFreqItemsets($D, \mathcal{I}, k, \epsilon_1$)
 2:     $\hat{\tau} \leftarrow \sigma_k + \mathsf{Lap}\left(\frac{4}{\epsilon_1}\right)$
 3:     $\mathcal{L}_1 \leftarrow$ GetFrequent($\mathcal{I}, \hat{\tau}, \frac{3\epsilon_1}{4}, \emptyset$)
 4:     $\ell \leftarrow 2$, $\mathcal{L} \leftarrow \mathcal{L}_1$
 5:     while $\mathcal{L}_{\ell-1} \neq \emptyset$ and $|\mathcal{L}| < k$ do
 6:         $\mathcal{C}_\ell \leftarrow \{a \cup b \mid a, b \in \mathcal{L}_{i-1} \wedge a < b\}$
 7:         $\mathcal{L}_\ell \leftarrow$ GetFrequent($\mathcal{C}_\ell, \hat{\tau}, \frac{3\epsilon_1}{4}, \mathcal{L}$)
 8:         $\mathcal{L} \leftarrow \mathcal{L} \cup \mathcal{L}_\ell$, $\ell \leftarrow \ell + 1$
 9:     return $\mathcal{L}$

10: function GetFrequent($\mathcal{C}, \hat{\tau}, \epsilon, L$)
11:     $S \leftarrow \emptyset$
12:     for each itemset $X \in \mathcal{C}$ do
13:         $\widehat{\sigma}(X) \leftarrow \sigma(X) + \mathsf{Lap}\left(\frac{1}{\epsilon}\right)$
14:         if $\widehat{\sigma}(X) \geq \hat{\tau}$ and $|L \cup S| < k$ then
15:             $S \leftarrow S \cup \{X\}$
16:     return $S$
```

in a recursive manner. Generating noisy counts for frequent itemsets can also be done using the binary itemset support counting method proposed in [5].

The privacy budget ϵ is allocated between two phases. Let $\epsilon_1 = \xi\epsilon$ and $\epsilon_2 = (1 - \xi)\epsilon$ be the privacy budget allocated to the first and second phase, respectively (i.e., $\epsilon = \epsilon_1 + \epsilon_2$), where $\xi \in (0, 1)$ is the parameter that controls privacy budget allocation between two phases. The larger ξ is, the more accurately the proposed algorithm can find frequent itemsets in the first phase while it would increase the amount of noise added to the support of each itemset in the second phase. We empirically settled on $\xi = \frac{1}{3}$ and use this value in the experiments.

4.1 Discovering Frequent Itemsets

The algorithm for finding a set of frequent itemsets $\mathcal{L}$ is described in Algorithm 1. The algorithm is Apriori-based. The candidate $(\ell + 1)$-itemsets are generated by joining two previously found frequent ℓ-itemsets that share a frequent $(\ell - 1)$-itemset as a prefix. While the proposed algorithm follows the intuition of Apriori algorithm, the biggest difference is that Algorithm 1 finds frequent itemsets by asking a set of threshold queries, each of which asks if the noisy support of itemset in question is above the noisy threshold or not and receives a boolean answer.

This threshold query based approach has a significant advantage over the count query based one in terms of noise addition. Let's consider the case where count queries are used instead of threshold queries.

Definition 4 (ℓ-count query set). *Given a set of candidate ℓ-itemsets, $\mathcal{C}_\ell$, an ℓ-count query set CQ_ℓ is defined as*

$$\mathsf{CQ}_\ell = \langle q_1, \cdots, q_{|\mathcal{C}_\ell|} \rangle$$

where each query q_i asks for the count of i^{th} itemset in $\mathcal{C}_\ell$.

At each iteration, the algorithm throws CQ_ℓ to the privacy mechanism to verify the supports of candidate itemsets.

Theorem 1. *The sensitivity of an ℓ-count query set $\Delta\mathsf{CQ}_\ell$ is the size of candidate set, $\mathcal{C}_\ell$.*

PROOF. Assume there is a transaction t that contributes to every $q_i (1 \leq i \leq n)$. Adding or removing t from D will change the answer of each q_i by one. Therefore, the global sensitivity of CQ_ℓ is the size of candidate set (i.e., $\Delta q = |\mathcal{C}|$). □

Since the sensitivity of a count query set is very high, a naïve application of the Laplace mechanism to the Apriori method will add unacceptable noise to each itemset.

DEFINITION 5 (ℓ-THRESHOLD QUERY SET). *An ℓ-threshold query set is a set of threshold queries for candidate ℓ-itemsets.*

$$\mathsf{TQ}_\ell = \langle q_1, \cdots, q_{|\mathcal{C}_\ell|} \rangle$$

where each q_i returns 1 if $\sigma(X) \geq \tau$, and 0 otherwise.

Privacy Analysis..

In a differentially private algorithm, all information disclosures must be performed in a differentially private way. Based on the sparse vector method, the noise in the result comes from the noise in the threshold, resulting in either near-frequent itemsets being treated as frequent, or vice-versa. This ensures that the effect of a single transaction on an itemset appearing in the result is low relative to the noise causing an itemset to appear in the result.

There would appear to be one issue: we don't ask if *all* itemsets are frequent, only candidates. However, the first round considers all possible items, and subsequent round *candidates* are generated from the (differentially private) output of the previous round without access to the database, satisfying differential privacy.

Given an integer $k > 0$, the algorithm starts from calculating the noisy threshold by adding noise to the support of the k^{th}most frequent itemset, $\sigma(k)$ (line 2). Observe that adding or removing a transaction can only change the support of the k^{th}most frequent itemset, $\sigma(k)$, by at most one. Hence, adding a random noise drawn from $\mathsf{Lap}\left(\frac{4}{\epsilon_1}\right)$ to $\sigma(k)$ satisfies $\frac{\epsilon_1}{4}$-differential privacy. Let $\hat{\tau_1}$ and $\hat{\tau_2}$ be the noisy thresholds for two neighboring database D_1 and D_2, respectively. For $\forall x \in \mathbb{R}$,

$$\mathrm{P}[\hat{\tau_1} = x] \leq \exp\left(\frac{\epsilon_1}{4}\right) \mathrm{P}[\hat{\tau_2} = x] \tag{1}$$

After computing $\hat{\tau}$, for each itemset $X \in \mathcal{C}$, the algorithm internally computes the noisy support $\widehat{\sigma}(X)$ by adding Laplacian random noise to its true support $\sigma(X)$. If the calculated noisy support is greater than or equal to the noisy threshold, the itemset is regarded frequent. Otherwise, it is regarded infrequent. Notice that in either case what is released by this phase of the algorithm is not the noisy support but *whether the itemset is frequent or not.* Let $\mathcal{C}$ be the set of all candidate itemsets and X_i be the i^{th}element in $\mathcal{C}$. We can model the set of answers the algorithm receives from the privacy mechanism as a vector $\mathbf{v} = \langle v_1, \cdots, v_t \rangle$ where $v_i = 1$ if $\widehat{\sigma}(X_i) \geq \hat{\tau}$, otherwise $v_i = 0$.

$$v_i = \begin{cases} 1 & \text{if } \widehat{\sigma}(X_i) \geq \hat{\tau} \\ 0 & \text{otherwise} \end{cases}$$

Therefore, to prove the privacy of Algorithm 1, it is sufficient to show that the privacy loss for all possible $\mathbf{v} \in \mathbb{Z}$ is bounded by $\exp(\epsilon)$ where $\mathbb{Z}$ is the set of all possible binary vectors. In Theorem 2, we prove that vector $\mathbf{v}$ released by the Algorithm 1 satisfies ϵ_1-differential privacy regardless of its length.

THEOREM 2. *Algorithm 1 is ϵ_1-differentially private.*

PROOF. Given any two neighboring databases D_1 and D_2, let V_1 and V_2 denote the output distribution on $\mathbf{v}$ when D_1 and D_2 are the input databases, respectively. We use $v^{<t}$ to denote $(t-1)$ previous answers from the mechanism (i.e., $v^{<t} = \langle v_1, \cdots, v_{t-1} \rangle$). Using the law of conditional probability, the privacy loss due to $\mathbf{v} = \langle v_1, \cdots, v_t \rangle$ for every $a_i \in \{0, 1\}$ is

$$\begin{aligned} \frac{V_1(\mathbf{v})}{V_2(\mathbf{v})} &= \frac{\prod_{i=1}^t V_1\left(v_i = a_i \mid v^{<i}\right)}{\prod_{i=1}^t V_2\left(v_i = a_i \mid v^{<i}\right)} \\ &= \prod_{i:a_i=1} \frac{V_1\left(v_i{=}1 \mid v^{<i}\right)}{V_2\left(v_i{=}1 \mid v^{<i}\right)} \cdot \prod_{j:a_j=0} \frac{V_1\left(v_j{=}0 \mid v^{<j}\right)}{V_2\left(v_j{=}0 \mid v^{<j}\right)} \end{aligned}$$

Once $v^{<i}$ is fixed, the conditional distributions of $\hat{\tau}$ and $\widehat{\sigma}(X)$ are just a Laplace distribution. Let $H_i^1(x)$ be the probability that the itemset X_i is frequent (*i.e.*, $v_i = 1$) in D_1 when the threshold is x.

$$\begin{aligned} H_i^1(x) &= \mathrm{P}\left[v_i = 1 \mid v^{<i}\right] = \mathrm{P}\left[\widehat{\sigma}_1(X_i) \geq x \mid v^{<i}\right] \\ &= \mathrm{P}\left[\mathsf{Lap}\left(\frac{4}{3\epsilon_1}\right) {\geq} x {-} \sigma_1(X_i) \,\middle|\, v^{<i}\right] \end{aligned}$$

Let $\lambda = \frac{4}{3\epsilon_1}$ and $f(y; \mu, \lambda) = \frac{1}{2\lambda} \exp\left(-\frac{|y-\mu|}{\lambda}\right)$.

$$H_i^1(x) = \int_x^\infty f\left(y; \sigma_1(X_i), \frac{4}{3\epsilon_1}\right) \mathrm{d}y$$

Recall that $\Delta\sigma = 1$. If we make a substitution $u = y + 1$, from $\mathrm{d}u = \mathrm{d}y$, the above yields

$$H_i^1(x) = \int_{x+1}^\infty f\left(u; \sigma_1(X_i) {+} \Delta\sigma, \frac{4}{3\epsilon_1}\right) \mathrm{d}u \tag{2}$$

There are two possible cases:

- If $\sigma_2(X_i) = \sigma_1(X_i) + 1$, then equation (2) yields
$$H_i^1(x) = \int_{x+1}^\infty f\left(u; \sigma_2(X_i), \frac{4}{3\epsilon_1}\right) \mathrm{d}u = H_i^2(x+1)$$
- If $\sigma_2(X_i) = \sigma_1(X_i)$, then $H_i^1(x) = H_i^2(x)$.

Let the set of indices of answers for itemsets whose supports are the same in both D_1 and D_2 and for itemsets whose supports increase by one be $S = \{i | a_i{=}1 \wedge \sigma_1(X_i){=}\sigma_2(X_i)\}$ and $\bar{S} = \{i | a_i = 1 \wedge \sigma_2(X_i){=}\sigma_1(X_i){+}1\}$, respectively.

$$\begin{aligned} &\prod_{i:a_i=1} V_1\left(v_i{=}1 \mid v^{<i}\right) \\ &= \prod_{i \in S} V_1\left(v_i{=}1 \mid v^{<i}\right) \cdot \prod_{j \in \bar{S}} V_1\left(v_j{=}1 \mid v^{<j}\right) \\ &= A \cdot B \end{aligned} \tag{3}$$

where $A{=}\prod_{i \in S} V_1\left(v_i{=}1 | v^{<i}\right)$ and $B{=}\prod_{j \in \bar{S}} V_1\left(v_j{=}1 | v^{<j}\right)$.

$$\begin{aligned} A &= \int_{-\infty}^\infty \mathrm{P}[\hat{\tau_1}{=}x] \prod_{i \in S} H_i^1(x)\,\mathrm{d}x {=} \int_{-\infty}^\infty \mathrm{P}[\hat{\tau_1}{=}x] \prod_{i \in S} H_i^2(x)\,\mathrm{d}x \\ &\leq \exp\left(\frac{\epsilon_1}{4}\right) \int_{-\infty}^\infty \mathrm{P}[\hat{\tau_2}{=}x] \prod_{i \in S} H_i^2(x)\,\mathrm{d}x \\ &= \exp\left(\frac{\epsilon_1}{4}\right) \prod_{i \in S} V_2\left(v_i{=}1 \mid v^{<i}\right) \end{aligned} \tag{4}$$

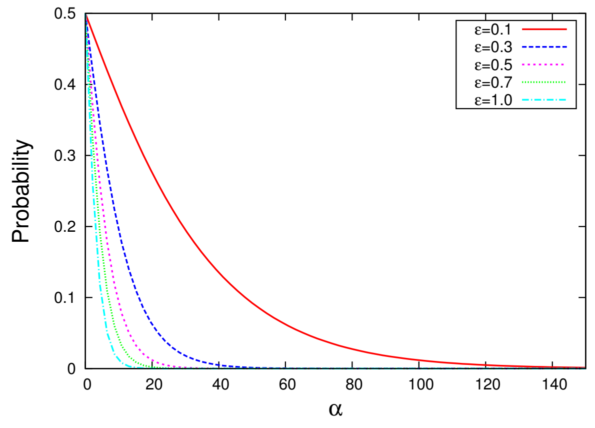


Figure 1: False negative probability

$$\begin{aligned}
B &= \int_{-\infty}^{\infty} \mathrm{P}[\hat{\tau}_1{=}x] \prod_{j\in\bar{S}} H_j^1(x)\,\mathrm{d}x \\
&= \int_{-\infty}^{\infty} \mathrm{P}[\hat{\tau}_1{=}x] \prod_{j\in\bar{S}} H_j^2(x+1)\,\mathrm{d}x \\
&\le \exp\left(\frac{\epsilon_1}{4}\right) \int_{-\infty}^{\infty} \mathrm{P}[\hat{\tau}_2{=}x{+}1] \prod_{j\in\bar{S}} H_j^2(x+1)\,\mathrm{d}x \\
&= \exp\left(\frac{\epsilon_1}{4}\right) \int_{-\infty}^{\infty} \mathrm{P}[\hat{\tau}_2{=}x] \prod_{j\in\bar{S}} H_j^2(x)\,\mathrm{d}x \\
&= \exp\left(\frac{\epsilon_1}{4}\right) \prod_{j\in\bar{S}} V_2\left(v_j{=}1 \mid v^{<j}\right) \qquad (5)
\end{aligned}$$

From (4) and (5), it is seen that

$$\prod_{i:a_i=1} V_1\left(v_i{=}1 \mid v^{<i}\right) \le \exp\left(\frac{\epsilon_1}{2}\right) \prod_{j:a_j=1} V_2\left(v_j{=}1 \mid v^{<j}\right)$$

By the same argument as in the preceding, we can prove

$$\prod_{i:a_i=0} V_1\left(v_i{=}0 \mid v^{<i}\right) \le \exp\left(\frac{\epsilon_1}{2}\right) \prod_{j:a_j=0} V_2\left(v_j{=}0 \mid v^{<j}\right)$$

Therefore, for all $\mathbf{v}$ of any length t, $\frac{V_1(\mathbf{v})}{V_2(\mathbf{v})} \le \exp(\epsilon_1)$. $\square$

Utility Analysis..

As shown in Figure 1, the farther an itemset is from the threshold, the lower chance of being a false negative it has, which means that, except for itemsets that are close to the "borderline", the algorithm is highly likely to find all frequent itemsets.

THEOREM 3. *Given an itemset X whose support is $\tau+\alpha$, the probability that this itemset will become a false negative is $\frac{1}{4}\exp\left(-\frac{\alpha\epsilon}{2}\right)\left(\frac{\alpha\epsilon}{2}+2\right)$.*

PROOF. The mechanism will answer correctly if the following condition is satisfied:

$$\hat{\tau} - \tau - (\widehat{\sigma}(X) - \sigma(X)) < \alpha \qquad (6)$$

Observe that $\hat{\tau} - \tau$ and $\widehat{\sigma}(X) - \sigma(X)$ are two independent random variables whose distribution is $\mathsf{Lap}(\lambda = \frac{2}{\epsilon})$. Let V and W be random variables corresponding to $\hat{\tau} - \tau$ and $\widehat{\sigma}(X) - \sigma(X)$, respectively. Let $Z = V - W$ denote the difference between two random variables. Since Laplace distribution is symmetric, $Z{=}V{-}W{=}V{+}W$. To denote the probability density function of Z, we use the notation $f_Z(z)$.

$$\begin{aligned}
f_Z(z) &= \int_{-\infty}^{\infty} f_V(x) f_W(x-z)\,\mathrm{d}x \\
&= \int_{-\infty}^{\infty} \frac{1}{4\lambda^2} \exp\left(-\frac{|x|}{\lambda}\right) \exp\left(-\frac{|x-z|}{\lambda}\right) \mathrm{d}x \\
&= \int_{-\infty}^{0} \frac{1}{4\lambda^2} \exp\left(\frac{x}{\lambda}\right) \exp\left(-\frac{|x-z|}{\lambda}\right) \mathrm{d}x \\
&\quad + \int_{0}^{\infty} \frac{1}{4\lambda^2} \exp\left(-\frac{x}{\lambda}\right) \exp\left(-\frac{|x-z|}{\lambda}\right) \mathrm{d}x \\
&= \frac{1}{4\lambda} \exp\left(-\frac{|z|}{\lambda}\right) \left(1 - \frac{|z|}{\lambda}\right)
\end{aligned}$$

The probability of not satisfying (6) is
$1 - \int_{-\infty}^{\alpha} \frac{1}{4\lambda} \exp\left(-\frac{|z|}{\lambda}\right)\left(1 + \frac{|z|}{\lambda}\right) \mathrm{d}z = \frac{1}{4}\exp\left(-\frac{\alpha}{\lambda}\right)\left(\frac{\alpha}{\lambda}+2\right)$ $\square$

False positives may also occur (again, most likely if close to the top k), but are less critical as they may later be filtered by having a low (noisy) support.

4.2 Deriving Noisy Support

Given a set of frequent itemsets $\mathcal{L}$, it is easy to find the set of maximal frequent itemsets $\mathcal{M}$. This can be done on the fly by checking if the newly added frequent itemset is subsumed by any existing itemsets.

Algorithm 3 derives noisy supports of all itemsets in $\mathcal{L}$. The algorithm employs an FP-tree data structure and the FPGrowth algorithm [9]. As the algorithm follows the usual procedure of the FPGrowth algorithm, we only remark on the differences.

- A node v in the FP-tree has three fields: $v.item$, $v.count$ and $v.children$, where $item$ denotes which item v represents, $count$ is the number of transactions represented by the path from the root to v, and $children$ is an array of v's child nodes.
- The original algorithm arranges items in a transaction in a decreasing order of their supports. However, since that information is not available to our algorithm, we sort them lexicographically.
- Before counting the support of each itemset, a node is created for each itemset $l \in 2^M$ where 2^M denotes the power set of M (line 3 in Algorithm 2). This is to ensure that the structure of FP-tree is not dependent on any particular database instance; the tree structure is completely determined by the output of the previous phase, $\mathcal{M}$, learned in a differentially private way.
- The count attribute of a newly created node in the FP-tree is initialized to $\mathsf{Lap}\left(\frac{|\mathcal{M}|}{\epsilon_2}\right)$. Note that, although it is omitted in the algorithm, every newly created node should be linked to the nodes with the same item via the side-link structure (line 8 in Algorithm 2).
- A transaction is mapped to a single path in the FP-tree and only increases the count of the last node on the path (line 17) by one. This means that 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 FP-tree is 1.

Algorithm 2 BuildFPTree

```
Input: set of dimensions for projection $M$
Output: FP-tree $T$
 1: function BuildFPTree($M, \epsilon$)
 2:     $\gamma \leftarrow$ create a root node
 3:     for each itemset $l \in 2^M$ do
 4:         $v \leftarrow \gamma$
 5:         for each item $i \in l$ do
 6:             if $\nexists w$ s.t. $w \in v.children \wedge w.item = i$ then
 7:                 create a new node $w$ under $v$
 8:                 $w.item \leftarrow i$; $w.count \leftarrow \mathsf{Lap}\left(\frac{1}{\epsilon}\right)$
 9:             $v \leftarrow w$
10:     return $\gamma$

11: function UpdateCount($D, M, T$)
12:     for each transaction $t \in D$ do
13:         $t \leftarrow t \cap M$; $v \leftarrow T$
14:         for each item $i \in t$ do
15:             if $v$ has a child $w$ s.t. $w.item = i$ then
16:                 $v \leftarrow w$
17:         $v.count \leftarrow v.count + 1$
```

- To get the correct count after building the FP-tree, starting from the leaf nodes going up toward the root node, the count of each node is added to its parent (line 7 in Algorithm 3). Observe that both actual occurrence count and noise are propagated to the parent.

Given an itemset X, its noisy support $\widehat{\sigma}(X)$ is an unbiased estimator of the true support and we measure the error of noisy support of X as its variance, i.e., $\mathsf{Error}(X) = \mathbb{E}\big[(\widehat{\sigma}(X) - \sigma(X))^2\big]$. If an itemset X is a subset of multiple maximal frequent itemsets, say $M_1, M_2, \cdots, M_m$, its support is counted by m FP-trees and they need to be aggregated in a way that minimize $\mathsf{Error}(X)$. To aggregate multiple noisy supports, each noisy support is weighted proportional to the inverse of its variance as in Lemma 1.

Lemma 1. *Given m noisy supports, $\widehat{\sigma}_1(X), \cdots, \widehat{\sigma}_m(X)$, and their respective variances $v_1, \cdots, v_m$, the variance of weighted mean $\mathsf{Var}\left(\sum_{i=1}^{m} w_i \widehat{\sigma}_i(X)\right)$ is minimized by setting $w_i = \frac{1}{v_i} \Big/ \sum_{j=1}^{m} \frac{1}{v_j}$.*

While Algorithm 3 looks like a (non-noisy) query to the database, which would appear to violate differential privacy, the output is equivalent to constructing a tree from the (differentially private) set of maximal frequent itemsets, then filling in supports using a noisy count query for each node. The algorithm as written does this with a single pass through the database.

Theorem 4. *Algorithm 3 achieves ϵ_2 differential privacy.*

Proof. Given an input database D, the process of building the FP-tree can be viewed as a query function $g(D)$ which returns an FP-tree built upon D. An FP-tree is a set of nodes each of which contains the count of the corresponding itemset. Hence, using an arbitrary traversal order, it can be modeled as a vector $\mathbf{u_D} = (u_1, \cdots, u_i, \cdots, u_q)$ where $u_i (1 \le i \le q)$ represents a node count and q is the number of nodes in the tree. For the proof, it suffices to show that the function UpdateCount satisfies differential privacy since this is the only place in the algorithm that requires access

Algorithm 3 GetNoisySupport

```
Input: database $D$, set of maximal frequent itemsets $\mathcal{M}$,
       privacy budget $\epsilon_2$
Output: Top $k$ frequent itemsets in $\mathcal{L}$ and their support
 1: function GetNoisySupport($D, \mathcal{M}, \epsilon_2$)
 2:     $\mathcal{L} \leftarrow \phi$
 3:     for each maximal itemset $M \in \mathcal{M}$ do
 4:         $T \leftarrow$ BuildFPTree($M, \frac{\epsilon_2}{|\mathcal{M}|}$)
 5:         UpdateCount($D, M, T$)
 6:         for each node $v \in T$ do
 7:             $v.count = v.count + \sum_{w \in v.children} w.count$
 8:         FPGrowth($T, M, \phi, \mathcal{L}$)
 9:     return top $k$ itemsets and their supports in $\mathcal{L}$

10: function FPGrowth($T, M, \alpha, \mathcal{L}$)
11:     if $T$ has a single path $P$ then
12:         for each combination $\beta \in P$ do
13:             $\mathcal{L}(\beta \cup \alpha).count \leftarrow$ min. support of nodes in $\beta$
14:     else
15:         for each $b_i \in M$ do
16:             $\beta \leftarrow b_i \cup \alpha$
17:             $\mathcal{L}(\beta).count \leftarrow$ support of $b_i$
18:             construct $\beta$-conditional FP-tree $T_\beta$
19:             FPGrowth($T_\beta, M, \beta$)
```

to D. Other steps, such as noise propagation and support derivation, can be done purely based on the output of the function. Notice that changing a transaction in D can only change the value of one entry by at most 1. This is because the algorithm is designed to update the count of only one node per transaction. Suppose the itemset X corresponding to u_i is removed from D. It will only decrease u_i by exactly 1 and other entries will remain the same. The resulting vector will be $\mathbf{u_{D'}} = (u_1, \cdots, u_{i-1}, u_i - 1, u_{i+1}, \cdots, u_q)$. Therefore, for all D and its neighboring database D',

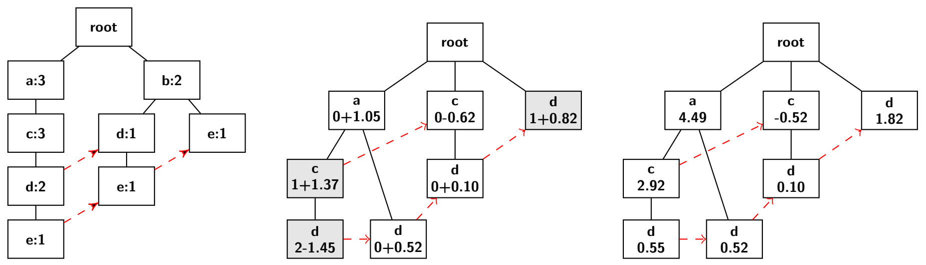
$$\Delta g = \max_{D,D'} \left| g(D) - g(D') \right| = 1$$

According to the Laplace mechanism adding independent Laplace noise drawn from $\mathsf{Lap}\left(\frac{|\mathcal{M}|\Delta g}{\epsilon_2}\right)$ to each entry satisfies $\frac{\epsilon_2}{|\mathcal{M}|}$-differential privacy, and by the composition theorem of differential privacy [16] constructing $|\mathcal{M}|$ noisy FP-trees satisfies ϵ_2-differential privacy. □

Figure 2 gives an example of building a noisy version of an FP-tree. Let the input database be Figure 2(a) and, for simplicity, assume each transaction is lexicographically ordered after removing infrequent items. Given the minimum support τ=2, there are two maximal frequent itemsets, $M_1 = \{a, c, d\}$ and $M_2 = \{b, e\}$. The proposed algorithm will construct two FP-trees, one for M_1 and the other for M_2. We now explain how the algorithm constructs the noisy FP-tree for $M_1 = \{a, c, d\}$. First, the algorithm starts by creating a root node and inserts all the subsets of M_1 to the tree. I.e., it creates nodes corresponding to $\{a\}$, $\{c\}$, $\{d\}$, $\{a, c\}$, $\{a, d\}$, $\{c, d\}$ and $\{a, c, d\}$. The count of each newly created node is initialized with Laplace noise (line 8 in Algorithm 2). After creating nodes, the algorithm takes each transaction and filters out any item that does not appear in M_1. The item e is removed from the first transaction since it is not a part of M_1. The count of the node corresponding to

TID	Items
1	$\{a, c, d, e\}$
2	$\{a, c, d\}$
3	$\{b, d, e\}$
4	$\{a, c\}$
5	$\{b, e\}$

(a) Input database



(b) original FP-tree (c) the tree built (d) after count propagation

Figure 2: Construction of noisy FP-tree

this filtered transaction is updated. Unlike the original FP-tree, only the count of the last node in the matching path is increased by 1 to reflect the occurrence of the transaction. In Figure 2(c), those nodes whose counts are updated to reflect the occurrence of transactions are marked as shaded. The first transaction causes to increase the count of node d, on the path $\langle(a : 1.05), (c : 1.37), (d : -1.45)\rangle$, by 1. The second transaction also increases the count of the same node since they become the same after the filtering step. The third transaction updates the count of node d on the first level. The fourth transaction results in $\langle(a : 1.05), (c : 2.37)\rangle$. The fifth transaction is ignored since it has no common item with M_1. All the items are removed from the transaction and the resulting transaction becomes an empty set. After building the tree, from the leaf toward the root, the count of each node is added to its parent. The FP-tree after the count propagation is shown in Figure 2(d).

4.3 Imposing Consistency

Given a node v in a non-perturbed FP-tree, according to the construction of the tree, the count of v cannot be smaller than the sum of counts of its children. Formally,

$$v.count \geq \sum_{w \in v.children} w.count \tag{7}$$

However, the constraint (7) may be violated in the noisy tree created by Algorithm 3 due to the noise injected to each node. The accuracy of randomized output can be improved though post-processing to impose consistency on the tree-like data structure [12, 6]. This is done without reference to the original data, only the already differentially-private structure, so the result is still differentially private.

In this section, we formulate the problem of imposing consistency on the noisy FP-tree as a *constrained least squares* problem and provide the formulation to find the solution. Let $\hat{x} = (\hat{x}_1, \cdots, \hat{x}_n)$ be an n-vector of noisy counts of nodes in an FP-tree, i.e., the i^{th}element of $\hat{x}_i$ corresponds to the noisy count of the i^{th}node v_i in the tree. Note that the ordering of nodes can be arbitrarily chosen. We impose consistency constraints on the noisy FP-tree by constructing a consistent vector $\bar{x}$ such that the distance between $\bar{x}$ and $\hat{x}$ is minimized. In other words, given a noisy FP-tree $\widehat{T}$, our algorithm tries to find another tree that is closest to $\widehat{T}$ while also satisfying the constraint.

Constraint (7) can be represented by a system of linear inequalities $C\bar{x} \leq \bar{x}$ where

$$C = (c_{ij}) = \begin{cases} 1 & \text{if } v_j \in v_i.children \\ 0 & \text{otherwise} \end{cases}$$

Let c_i^T denote the i^{th}row of C and W_i be the set of indices of v_i's children, i.e., $w_i = \{j \mid v_j \in v_i.children\}$. The $i^{\text{th}} c_i^T$ represents the constraint $\bar{x}_i \geq \sum_{j \in W_i} \bar{x}_j$. Hence, the problem can be formulated as follows:

$$\underset{\bar{x}}{\text{minimize}} \quad \|\bar{x} - \hat{x}\|_2 \qquad \text{subject to} \quad C\bar{x} \leq \bar{x} \tag{8}$$

We can reformulate the problem as a *least distance programming* (LDP) by substituting $\bar{x} - \hat{x}$ with $\bar{x}'$.

$$\underset{\bar{x}'}{\text{minimize}} \quad \|\bar{x}'\|_2 \qquad \text{subject to} \quad G\bar{x}' \geq g \tag{9}$$

where $G = I - C$ and $g = -G\hat{x}$.

Theorem 5. *Let u be the solution to the* nonnegative least squares *problem*

$$\underset{u}{\textit{minimize}} \quad \|Eu - f\|_2 \qquad \textit{subject to} \quad u \geq 0$$

and $r = (r_1, \cdots, r_{n+1}) = Eu - f$ *where*

$$E = \begin{bmatrix} G^T \\ g^T \end{bmatrix}, f = [\overbrace{0, \cdots, 0}^{n}, 1]^T$$

. *The unique solution to the problem* (8) *is*

$$\bar{x} = (\bar{x}_1, \cdots \bar{x}_n), \quad \bar{x}_j = -r_j/r_{n+1} + \hat{x}_j$$

Proof. See [14]. ☐

The impact of constraint enforcement on the accuracy is evaluated in Section 5.2.

5. EXPERIMENTAL RESULTS

In this section, we evaluate the performance of the proposed algorithm (NC) on a variety of datasets and compare it with two state-of-the-art algorithms: PrivBasis (PB) and SmartTruncation (ST). The datasets used in the experiments are described in Table 2. All experiments are conducted on an Intel Xeon 2.4GHz machine with 24GB of physical memory. To obtain an average performance estimate, each algorithm was run 10 times. PrivBasis finds top k frequent itemsets while SmartTruncation reports all frequent itemsets with respect to the given threshold. For comparison, the minimum support for SmartTruncation was set to the frequency of the k^{th}most frequent itemset. To compare the performance of algorithms, we employ F score and Relative error (RE) as measures of utility.

Definition 6 (F score). *Let $\mathcal{F}$ and $\hat{\mathcal{F}}$ be the set of correct and published frequent itemsets, respectively. The* F score *is defined as follows*

$$Fscore = 2 \times \frac{precision * recall}{precision + recall}$$

dataset	$|D|$	$|\mathcal{I}|$	$\max|t|$	avg $|t|$
mushroom (MUS)	8,124	119	23	23
pumsb star (PUMSB)	49,046	2,088	63	50.5
retail (RETL)	88,162	16,470	76	10.3
kosarak (KOS)	990,002	41,270	2,498	8.1
aol (AOL)	647,377	2,290,685	48,070	34.9
BMS-POS (POS)	515,597	1,657	164	6.5
BMS-WebView1 (WV1)	59,602	497	267	2.5
BMS-WebView2 (WV2)	77,512	3,340	161	5.0

Table 2: Dataset description

where $precision = \frac{|\{\mathcal{F} \bigcap \hat{\mathcal{F}}\}|}{|\hat{\mathcal{F}}|}$ *and* $recall = \frac{|\{\mathcal{F} \bigcap \hat{\mathcal{F}}\}|}{|\mathcal{F}|}$

Notice that for top k FIM algorithms $|\mathcal{F}| = \left|\hat{\mathcal{F}}\right| = k$, hence $precision = recall = Fscore = 1 - \text{FNR}$ where FNR, used in [15] for utility measure, is the fraction of false negatives in the released frequent itemsets.

Definition 7 (Relative Error). *The relative error of published frequent itemset $\hat{\mathcal{F}}$ is defined as*

$$RE = \text{median}_{X \in \hat{\mathcal{F}}} \frac{|\hat{\sigma}(X) - \sigma(X)|}{\sigma(X)}$$

Before discussing the results, we note that the ST algorithm has maximal cardinality parameter ℓ that controls the trade-off between information loss and sensitivity reduction due to the transaction truncation. Its performance heavily relies on the proper setting of this parameter. However, there's no systematic way to set the parameter and the choice is merely heuristic. We fine-tuned this parameter until reasonable accuracy is obtained.

5.1 Comparison of Algorithms

Figure 3 shows the F scores of each algorithm by different values of ϵ. Observe that the proposed algorithm consistently outperforms both PB and ST algorithms and shows stable performance for different privacy budget and k. The F scores of PB and ST rapidly decrease at higher privacy levels (smaller ϵ). As the value of ϵ is decreased — higher privacy level is imposed—, the performance gap between the proposed algorithm and other two algorithms becomes more noticeable. Especially, in Figure 3(f) and 3(i) only the proposed algorithm shows a good performance when $\epsilon = 0.1$; both PB and ST have unacceptably low F scores.

One interesting observation is that PB shows better performance on POS dataset than ST while on the WV1 and WV2 dataset ST outperforms PB. One reason for this is that the maximal length of FIs in WV1 and WV2 dataset is 2, hence transaction truncation effectively reduces noise with relatively small cost of information loss. In contrast, PB tends to generate long and many bases because of abundance of frequent 2-itemsets. In addition, PB shows low performance when k is large. The best F scores of ST we were able to get for the PUMSB, retail and aol datasets were under 0.6 even after substantial parameter tuning. As shown in Figure 3, PB and ST algorithms are only useful when the low privacy level is required (large ϵ). Figure 4 describes the change of RE on different values of ϵ. Since ST algorithm doesn't report the support values of frequent itemsets, the RE of proposed algorithm is only compared with that of PB. The proposed algorithm shows similar or better performance on all tested datasets. Observe that in Figure 4(b) and 4(c) relative errors of PB are extremely high, which make the released FIs almost useless in practice.

5.2 Impact of Imposing Consistency

Figure 5 illustrates how optional consistency enforcement described in Section 4.3 affects performance. This experiment used the PUMSB dataset. As shown in Figure 5(a), imposing consistency on the FP-tree data structure not only produces consistent results but also improves utility. Although more improvement on accuracy is obtained when $k = 200$, this post-processing step increases processing time. The bigger FP-tree is, the longer it takes to find the consistent result. This is because that the size of constraint matrix is the number of nodes in the FP-tree, and solving least distance for a huge matrix is computationally expensive. Therefore, this post-processing step can be optionally executed when the consistency is essential to the problem or when there is enough computational power since the performance of the proposed algorithm without the optimization step is in general better than those of others. Although it is unfair to compare the processing time of each algorithm since the implementation of each algorithm is not written in the same language, we present it to show that the proposed algorithm is comparable to others.

6. CONCLUSION

We have proposed an algorithm for finding top-k frequent itemsets in a differentially private way. Our algorithm first discovers frequent itemsets (without their support values) using a small portion of the privacy budget. This enables the algorithm to use the remaining privacy budget to efficiently and effectively build a differentially private FP-tree. Once the tree is built, the supports of all frequent itemsets can be derived from the tree without access to the database.

We believe that the sparse vector mechanism demonstrated in this paper may also be applicable to other types of differentially private data mining, as it allows a potentially large amount of access to the data with the noise dependent only on the data that affects the output, rather than all data accessed.

Acknowledgment

This material is based upon work supported by the National Science Foundation under Grant No. CNS-1012208.

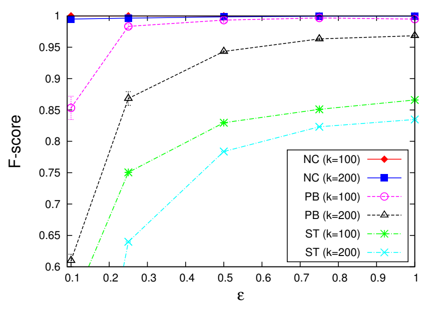
7. REFERENCES

[1] C. C. Aggarwal, J. Pei, and B. Zhang. On privacy preservation against adversarial data mining. In *KDD 2006*, pages 510–516, New York, NY, USA, 2006. ACM.

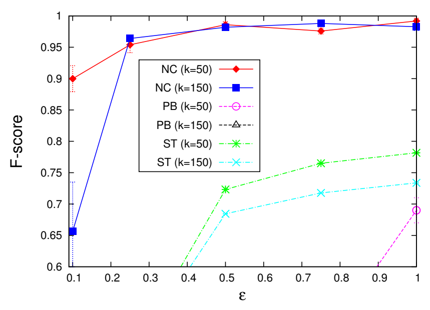
[2] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *VLDB '94*, pages 487–499, San Francisco, CA, USA, 1994. Morgan Kaufmann Publishers Inc.

[3] M. Atzori, F. Bonchi, F. Giannotti, and D. Pedreschi. Anonymity preserving pattern discovery. *The VLDB Journal*, 17(4):703–727, July 2008.

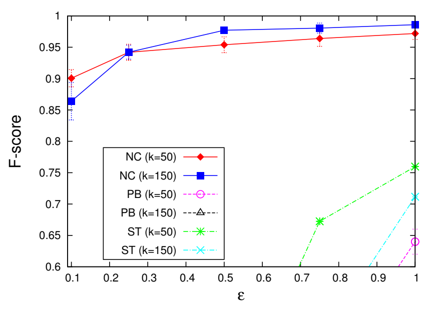
[4] R. Bhaskar, S. Laxman, A. Smith, and A. Thakurta. Discovering frequent patterns in sensitive data. In



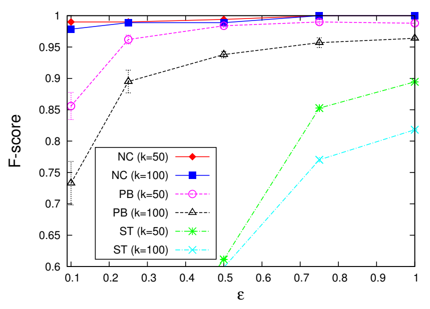
(a) BMS-POS



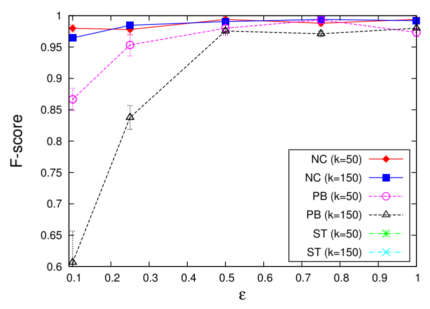
(b) BMS-WebView-1



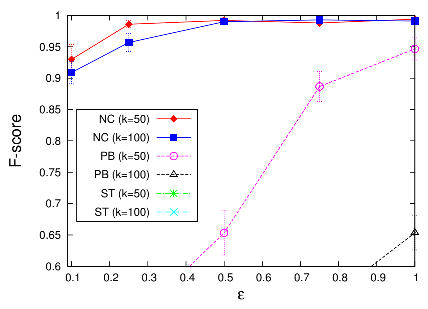
(c) BMS-WebView-2



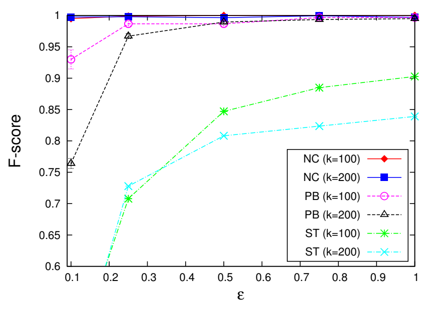
(d) mushroom



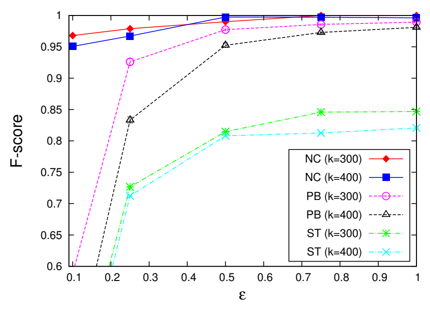
(e) pumsb star



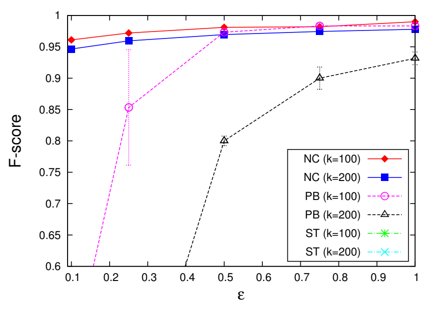
(f) retail



(g) kosarak



(h) kosarak



(i) aol

Figure 3: F score by varying ϵ

KDD 2010, pages 503–512, New York, NY, USA, 2010. ACM.

[5] J. Chen and K. Xiao. Bisc: A bitmap itemset support counting approach for efficient frequent itemset mining. *ACM Trans. Knowl. Discov. Data*, 4(3):12:1–12:37, Oct. 2010.

[6] B. Ding, M. Winslett, J. Han, and Z. Li. Differentially private data cubes: optimizing noise sources and consistency. In *SIGMOD 2011*, pages 217–228, New York, NY, USA, 2011. ACM.

[7] C. Dwork. Differential privacy. In *ICALP 2006*, pages 1–12, July 9-16 2006.

[8] C. Dwork, F. McSherry, K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. In *TCC 2006*, pages 265–284. Springer, 2006.

[9] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. *SIGMOD Rec.*, 29(2):1–12, May 2000.

[10] M. Hardt. *A Study of Privacy and Fairness in Sensitive Data Analysis*. PhD thesis, Princeton University, 2011.

[11] M. Hardt and G. N. Rothblum. A multiplicative weights mechanism for privacy-preserving data analysis. In *FOCS 2010*, pages 61–70, Washington, DC, USA, 2010. IEEE Computer Society.

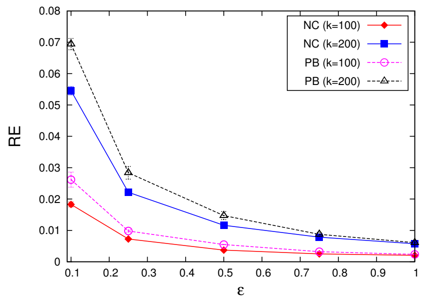
[12] M. Hay, V. Rastogi, G. Miklau, and D. Suciu. Boosting the accuracy of differentially private histograms through consistency. *Proc. VLDB Endow.*, 3(1-2):1021–1032, Sept. 2010.

[13] M. Kantarcioğlu, J. Jin, and C. Clifton. When do data mining results violate privacy? In *KDD 2004*, pages 599–604, New York, NY, USA, 2004. ACM.

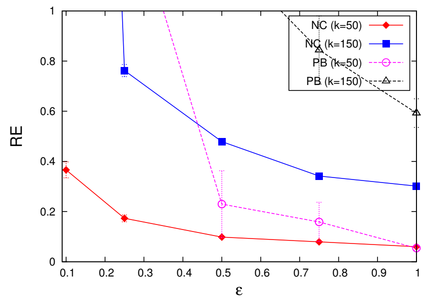
[14] C. L. Lawson and R. J. Hanson. *Solving least squares problems*. Classics in applied mathematics. SIAM, Philadelphia, PA, 3 edition, 1995.

[15] N. Li, W. Qardaji, D. Su, and J. Cao. Privbasis: frequent itemset mining with differential privacy. *Proc. VLDB Endow.*, 5(11):1340–1351, July 2012.

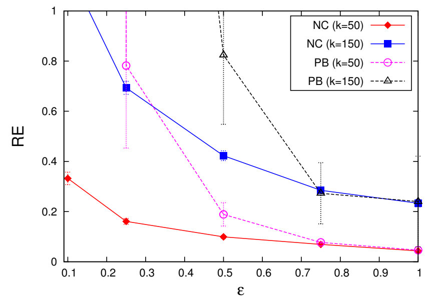
[16] F. D. McSherry. Privacy integrated queries: An extensible platform for privacy-preserving data analysis. In *SIGMOD 2009*, pages 19–30, New York, NY, USA, 2009. ACM.



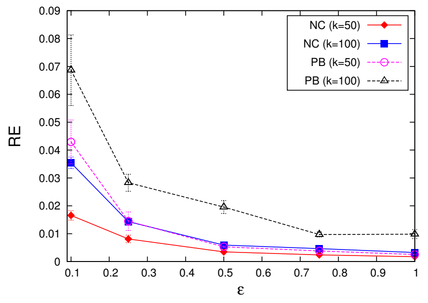
(a) BMS-POS



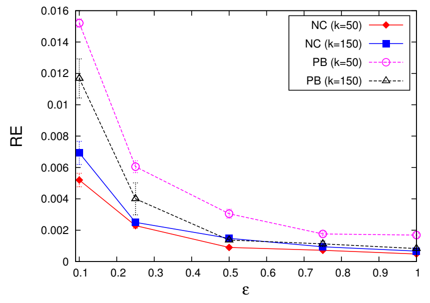
(b) BMS-WebView1



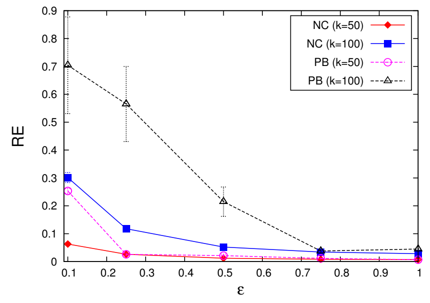
(c) BMS-WebView2



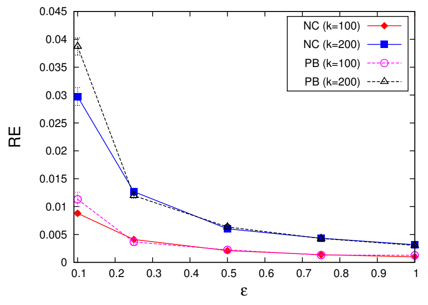
(d) mushroom



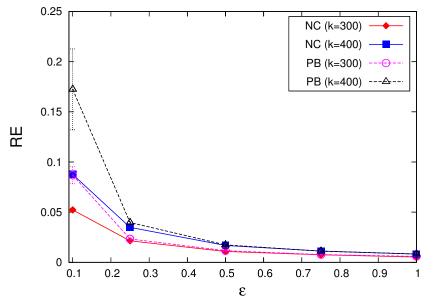
(e) pumsb star



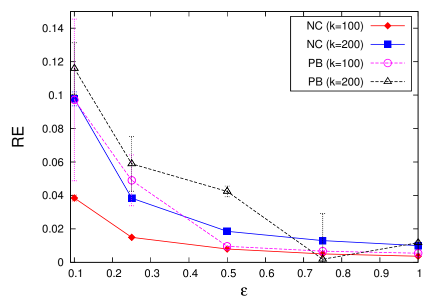
(f) retail



(g) kosarak

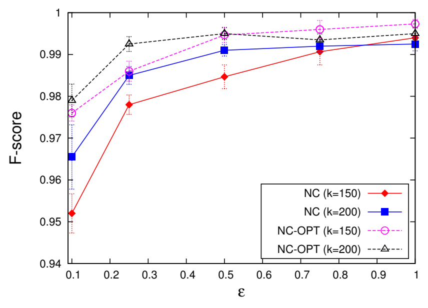


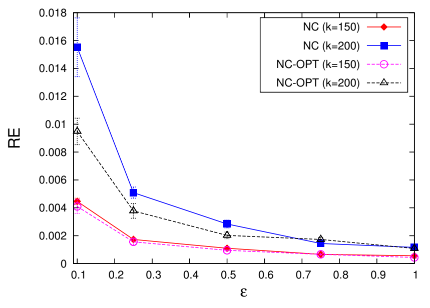
(h) kosarak



(i) aol

Figure 4: Relative error by varying ϵ





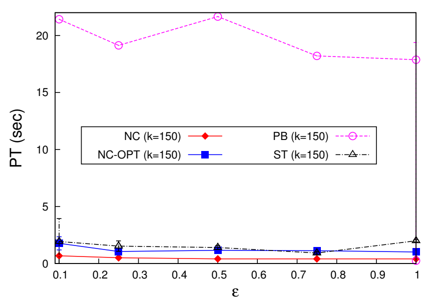


Figure 5: Effectiveness of consistency enforcement

[17] A. Roth and T. Roughgarden. Interactive privacy via the median mechanism. In *STOC 2010*, pages 765–774, New York, NY, USA, 2010. ACM.

[18] M. J. Zaki. Scalable algorithms for association mining. *IEEE Trans. on Knowl. and Data Eng.*, 12(3):372–390, May 2000.

[19] C. Zeng, J. F. Naughton, and J.-Y. Cai. On differentially private frequent itemset mining. *Proc. VLDB Endow.*, 6(1):25–36, Nov. 2012.