Differentially Private Set-Valued Data Release against Incremental Updates

Xiaojian Zhang[1,2,⋆], Xiaofeng Meng[1], and Rui Chen[3]

[1] School of Information, Renmin University of China, Beijing, China
[2] School of Computer and Information Engineering, Henan University of Economics and Law
{xjzhang82,xfmeng}@ruc.edu.cn
[3] Department of Computer Science, University of British Columbia, Vancouver, Canada
rui_chen@cs.ubc.ca

Abstract. Publication of the private set-valued data will provide enormous opportunities for counting queries and various data mining tasks. Compared to those previous methods based on partition-based privacy models (e.g., *k*-anonymity), differential privacy provides strong privacy guarantees against adversaries with arbitrary background knowledge. However, the existing solutions based on differential privacy for data publication are currently limited to static datasets, and do not adequately address today's demand for up-to-date information. In this paper, we address the problem of differentially private set-valued data release on an incremental scenario in which the data need to be transformed are not static. Motivated by this, we propose an efficient algorithm, called *IncTDPart*, to incrementally generate a series of differentially private releases. The proposed algorithm is based on top-down partitioning model with the help of item-free taxonomy tree and update-bounded mechanism. Extensive experiments on real datasets confirm that our approach maintains high utility and scalability for counting query.

Keywords: Differential privacy, set-valued data, incremental updates.

1 Introduction

Set-valued data, in which a set of items are associated with an individual, is common in database ranging from web query logs, to credit card transactions, and to shopping transaction databases of customers' behavior. Publishing and sharing set-valued data is important, since they enable researchers to analyze and explore interesting patterns and knowledge. For example, revealing strong correlations and trends from collected patient health records can be a valuable knowledge base for society; for a retail company, analyzing common customer behavior from online shopping data can provide useful information for advertising. However, such data usually contains specific sensitive information (e.g., pregnancy test, health care), and directly releasing raw data could violate individual privacy and may result in unveiling the identity of the individual associated with a particular transaction. To prevent such information leakage, set-valued

⋆ This research was partially supported by the grants from the Natural Science Foundation of China (No. 61070055, 91024032, 91124001); the National 863 High-tech Program (No. 2012AA010701, 2013AA013204); the Fundamental Research Funds for the Central Universities, and the Research Funds of Renmin University(No. 11XNL010).

W. Meng et al. (Eds.): DASFAA 2013, Part I, LNCS 7825, pp. 392–406, 2013.

data must be sanitized before release. Previous works have been made in addressing the problem of set-valued data publication. Terrovitis et al. [1] propose a local and global recoding method k^m-anonymity, and He and Naughton [2] enhanced [1] by a top-down, partition-based approach to handling the same question. However, recent efforts have shown that the above partition-based privacy models are vulnerable to many types of privacy attacks, such as composition attack [3], and foreground knowledge attack [4]. Recently, differential privacy [5] has emerged as one of the most promising models for releasing different types of private data, because it can provide strong privacy guarantees against adversaries with arbitrary background knowledge (this claim may not be valid in some cases where there exist correlations among records [6], but in this paper we assume that records are independent of each other). The main idea of differential privacy is to inject noise into a dataset so that an adversary cannot decide whether a particular record is included in the dataset or not. The noise level is controlled by privacy budget ε. There have been a few set-valued data publishing algorithms proposed in the recent work, such as [7, 8], that efficiently publish such data under differential privacy. These approaches, however, only deal with static set-valued data releases. That is, all these approaches assume that they work in a one-time fashion: sanitize the entire database and obtain the statistic information. This assumption often heavily limits the applicability of these differentially private methods, as in many dynamic applications set-valued datasets are updated incrementally.

Example 1. Consider a supermarket's store T, shown in Table 1, which is required to share the transactional items purchased by its various customers with market researchers. In order to protect customers' privacy, the supermarket sanitize all the items prior to releasing. At first glance, the task seems reasonable straightforward, as existing techniques in [7, 8] can efficiently anonymize the items. The challenge is, however, that T is growing daily due to the appending of newly purchased items for existing customers and/or insertion of new shopping items for new customers, shown in Table 2, and it is critical for the market researchers to receive up-to-date items in timely manner.

Table 1. The original database T

TID	Items Purchased
T_1	$\{apple, banana, cherry, melon\}$
T_2	$\{apple, cherry\}$
T_3	$\{cherry, melon\}$
T_4	$\{apple, melon\}$

Table 2. The Incremental Update ΔT_1

TID	Items Purchased
T_5	$\{banana, cherry, melon\}$
T_6	$\{melon, cherry\}$
T_7	$\{apple, melon\}$

Due to the inherent dynamics and high-dimensionality of set-valued data in the context of incremental updates, it is challenging to apply differential privacy to incrementally publishing set-valued data. In the real world transactional data usually arrives in batches to update original database incrementally. Thus, a differentially private mechanism must periodically update the published statistics as new data items are inserted or removed. We assume the coming update is large enough that it can support differential privacy.

In order to employ the existing methods to publish the updates, two straightforward solutions may be developed.

Solution 1: Sanitizing Incremental Updates. Support a series of updates arriving. This solution is to sanitize and publish each update independently so that it satisfies differential privacy. For example, we employ DiffPart method [7] to separately release Table 1 and Table 2. Then researchers can merge multiple those released datasets together for comprehensive analysis. Although straightforward, the major drawback of this solution is when researchers merge multiple noisy statistics, the error is accumulated in the merged results. So the results become noisier over time, which may lead to lower data quality.

Solution 2: Publishing Sanitization of Current Versions. This solution is to sanitize and publish the entire dataset whenever the dataset is augmented with new updates (e.g., merging the update ΔT_1 to T). In this way, researchers are always provided with up-to-date statistics information. Although this can be easily accomplished by using the existing methods, there are two significant drawbacks. First, K such updates will raise the privacy budget to $K\varepsilon$. This means that the more updates we have, the higher the amount of noise we need to add to each release. Another, if the number of updates is infinite, then ε-differential privacy will be eroded.

In the above solutions, the error increases with the number of updates, either because the noise accumulates (as in solution 1), or because the amount of noise relies on the number of updates (as in solution 2). This means that we cannot have an infinite number of updates. In other words, streaming approaches proposed in [10, 9] cannot be applied in our scenario because data stream has its own characteristics such as dynamics, continuity, and infinity. To tackle the above challenges, we investigate the problem of how to publish set-valued data against incremental updates while maintaining ε-differential privacy. First, we design a *update-bounded mechanism* to limit the number of updates which can help to reduce the error caused by the above two solutions. Second, based on update-bounded mechanism, we propose an efficient algorithm, called *IncTDPart* (Incrementally Top-Down Partitioning manner) to release set-valued data with the help of item-free taxonomy tree. In our algorithm, we incrementally maintain a tree structure, called *TBP-Tree* (Taxonomy-Based Partitioning Tree), in which leaf nodes store *p-sum* value (i.e., accumulated noisy counts) that can be used to construct a release after each update. Third, extensive experiments on several real datasets demonstrate that the proposed methods generate high utility for incrementally counting queries and scales to large datasets.

The remainder of this paper is organized as follows: Section 2 discusses related work. Section 3 briefly overviews ε-differential privacy and problem statement. Section 4 proposes a differentially private algorithm *IncTDPart* to support set-valued data releases against incremental updates. The experimental evaluation of our methods is presents in Section 5, and Section 6 concludes our work.

2 Related Work

The notion of differential privacy was presented by Dwork et al in [5]. The same authors also propose the addition of Laplace noise to guarantee differential privacy [11]. Work

on differential privacy has initially focused on answering statistical queries (e.g., count queries, range queries). However, a few recent works started addressing non-interactive data release that achieves differential privacy such as histograms publication [13, 12], and search logs releases [14]. McSherry et al [15]. present differentially private recommendation algorithms in Netflix prize competition. More recently, several works studied differentially private mechanism for releasing set-valued data [8, 7]. Chen et al. [7] presented the publishing of set-valued data while satisfying differential privacy, and in [8] they studied differentially private transit data publication. They present algorithms in [8, 7], which partition the set-valued data in top-down fashion guided by taxonomy tree, and release the noisy counts of the set-valued data at leaf nodes. Their methods generate synthetic set-valued data which can support counting queries. However, the existing publication approaches are currently limited to static datasets, do not adequately address the incremental updates. Among the existing approaches, the ones most related to ours are by Chan et al. [10] and Dwork, et al. [9], which continuously release statistics in the context of data streams. Due to the characteristics of data stream itself, their methods are not favorable to releasing set-valued datasets against updates.

In addition, there is a series of works [16, 18, 17] which are based on partition-based privacy models (e.g., k-anonymity) for incrementally releasing relational databases. The work [16] is among the first to identify possible attacks in the dynamic scenario. This work analyzes various inference channels that may exist in multiple anonymized datasets and discusses how to avoid such inferences. Xiao et al. [17] propose the novel m-invariance framework. This simple yet elegant method is the first work that successfully anonymizes a fully dynamic dataset. To enhance the methods in [17], He et al. [18] propose a graph-based anonymization algorithm to cope with equivalence attack. However, recent works have shown that these methods based on partition-based models are much weaker privacy notion than differential privacy.

3 Preliminaries

Let I=$\{I_1, I_2, ..., I_{|I|}\}$ be the universe of items, where $|I|$ is the size of the universe. The T=$\{t_1, t_2, ..., t_{|T|}\}$ denotes the initial set-valued table as it is created before updating, where each record $t_i \in T$ is a non-empty subset of I. Let ΔT_1, ΔT_2,... present the incremental updates (insertions only in this paper) to the table T. We assume that in the series of tables T, ΔT_1, ΔT_2,..., the item domain is fixed, that is, the universe of items is not changed. Suppose the item universe I=$\{I_1, I_2, I_3, I_4\}$, Table 3 presents an example of initial set-valued database T. Table 4 and Table 5 present the incremental updates ΔT_1, ΔT_2 to Table 3.

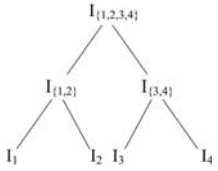


Fig. 1. IFT-Tree

Table 3. initial T

TID	Items
t_1	$\{I_1, I_2\}$
t_2	$\{I_2\}$
t_3	$\{I_1\}$

Table 4. update ΔT_1

TID	Items
t_4	$\{I_1, I_2, I_3, I_4\}$
t_5	$\{I_2, I_4\}$
t_6	$\{I_2\}$

Table 5. update ΔT_2

TID	Items
t_7	$\{I_1, I_2, I_3, I_4\}$
t_8	$\{I_2, I_3, I_4\}$
t_9	$\{I_1, I_2\}$

3.1 Item-Free Taxonomy Tree

In this paper, we assume that the series of tables T, ΔT_1, ΔT_2,..., are associated with a single taxonomy tree. In classical generalization mechanism, taxonomy tree relies on the original semantic of generalization to map several differential items to a single value in the destination nodes. In our method, we release only original items and counts, regardless of their semantics. Therefore, the taxonomy tree could be item free.

Definition 1. *(**Item-Free Taxonomy Tree**). An Item-Free Taxonomy Tree (IFT-Tree) is a taxonomy tree, whose internal nodes consist of their leaves, not necessarily to take into account the semantic generalization of the leaves.*

For example, Fig. 1 presents an item-free taxonomy tree for Table 3, Table 4, and Table 5, and one of its internal nodes $I_{\{3,4\}}$=$\{I_3, I_4\}$. An item can be mapped to an internal node if it is in the node's set. In this example, items $\{I_1, I_2\}$ can be generalized to $I_{\{1,2\}}$, items $\{I_3, I_4\}$ can be generalized to $I_{\{3,4\}}$, and the two sets $I_{\{1,2\}}$, $I_{\{3,4\}}$ can be further generalized to $I_{\{1,2,3,4\}}$.

3.2 Differential Privacy

Differential privacy, in general, guarantees that changing or removing any record from a database has negligible impact on the output of any analysis based on the databases. Therefore, an adversary will learn nothing about an individual, regardless of whether her record is present or absent in the database. Formally, in the context of incremental updates, differential privacy [5] is defined below.

Definition 2. *(ε-**Differential Privacy**). A randomized mechanism Ag for supporting incremental updates satisfies ε- differential privacy, iff for any output O of Ag and for any two neighbor databases T_1 and T_2, we have*

$$Pr[Ag(T_1) = O] \leq exp(\varepsilon) \cdot Pr[Ag(T_2) = O] \tag{1}$$

where ε is the privacy budget, and the probability is taken over the randomness of Ag. The neighbor database is obtained by removing on arbitrary record from either the original set-valued data, or any of the updates.

A principal technique for achieving differential privacy is *Laplace mechanism* [11]. A fundamental concept of this technique is the *global sensitivity* of a function f that maps underling databases to vectors of reals.

Definition 3. *(**Global Sensitivity**). For any function $f : T \rightarrow \mathbb{R}^d$, the sensitivity of f is defined as follow.*

$$\Delta f = \max_{T_1, T_2} \|f(T_1) - f(T_2)\|_1 \tag{2}$$

for all T_1, T_2 differing in at most one record.

The global sensitivity is also called L_1-sensitivity due to the L_1-norm used in its definition, which takes the maximum over all pairs of neighboring databases.

Laplace Mechanism. Dwork et al. [11] propose the Laplace mechanism which takes a database T, a function f, and the privacy budget ε. It first computes the true output $f(T)$, and then adds properly calibrated Laplace noise to the output. The noise is sampled from a Laplace distribution with the probability density function $Pr(x|b)=\frac{1}{2b}e^{-|x|/b}$, where b is dominated by both Δf and the allocated privacy budget ε.

Theorem 1. *For any function $f: T\rightarrow \mathbb{R}^d$, the private mechanism Ag*

$$Ag(T) = f(T) + \langle Y_1(\Delta f/\varepsilon), Y_2(\Delta f/\varepsilon), ..., Y_d(\Delta f/\varepsilon)\rangle \tag{3}$$

gives ε-differential privacy, where $Y_i(\Delta f/\varepsilon)(1 \leq i \leq d)$ are i.i.d Laplace variables with scale parameter $\Delta f/\varepsilon$.

Differential privacy has two important properties that are extensively used when differential privacy is employed to support combined computations. These two properties are known as *sequential* and *parallel* compositions [18].

3.3 Utility Metrics

In the incremental update case, sanitized set-valued data is mainly used to answer count queries that are crucial to counting queries task. We employ *relative error* [19] to measure the utility of the sanitized data. Given a series of databases $T, \Delta T_1, \Delta T_2, ...$, and let T_i be $T \cup_{j=1}^{i} \Delta T_i$ after appending the i-th incremental update ($i \geq 1$).

Definition 4. *(**Incremental Count Query**). Given an initial dataset T, after the i-th incremental update ($i \geq 1$), for a given set of items I' drawn from the universe I, an incremental count query Q over T_i is defined to be $Q(T_i)=|\{t \in T_i : I' \in t\}|$.*

relative error(*RE*): This measures the error to the actual answer on the actual database T_i, which is formalized as follows.

$$RE = \frac{|Q(\widetilde{T_i}) - Q(T_i)|}{max\{Q(T_i), b\}} \tag{4}$$

where $Q(\widetilde{T_i})$ denotes the answer on the sanitized database $\widetilde{T_i}$, $Q(T_i)$ denotes the true answer on the actual database T_i, and b denotes a sanity bound used to mitigate the influences of queries with extremely small selectivities.

Problem Statement. Given a private parameter ε, a transactional table T and a series of updates $\Delta T_1, \Delta T_2, ...$ to T, our objective is to generate a series of publications which satisfy differential privacy against incremental updates.

4 Update-Bounded Sanitization Algorithm

In our setting, due to the set-valued dataset incremental update, a private mechanism must update the published statistics as new updates arrive. Thus, traditional differentially private mechanisms either fail to apply directly to our setting, or result in an

unsatisfactory loss in terms of utility or privacy if applied naively (e.g., Solution 1 and Solution 2). To tackle the drawbacks caused by Solution 1 and Solution 2, we propose a novel constraint method, called *Update-Bounded Mechanism*, to limit the number of incremental updates.

Definition 5. *(**Update-Bounded Mechanism**). Given $U \in \mathbb{N}$, an incremental release mechanism Ag is U-bounded if it only accepts updates at most U. In other words, Ag needs to require a priori knowledge of an upper bound on the number of updates.*

Based on update-based mechanism, we present the *IncTDPart* algorithm that recursively partitions the series of set-valued datasets with the help of an IFT-Tree against incremental updates.

We first provide an overview of our *IncTDPart* algorithm in Algorithm 1. The algorithm first builds the IFT-tree H by iteratively grouping f nodes from one level to an upper level until a root is reached. If the size of the item universe is not divided by f, the remainder can be as a group. Given T_i=$\{T, \Delta T_1, \Delta T_2, ...\}$, a privacy budget ε, an upper bound of updates U and an IFT-Tree H, it returns a series of sanitized databases satisfying differential privacy. Based on the given IFT-Tree H, we employ DiffPart method to sanitize the initial dataset T, and release $\widetilde{T}$. *IncBuildTBP-Tree* incrementally maintains a noisy taxonomy-based partitioning tree by a top-down manner, and releases the series of sanitized datasets $\widetilde{T}_i$.

Algorithm 1. IncTDPart

Input: $T, \Delta T_1, \Delta T_2, ..., \Delta T_U$: The initial dataset T, and a series of updates; U: The upper bound on updates; ε: The privacy budget; f: The fan-out of IFT-Tree
Output: The sanitation $\widetilde{T}$, $\widetilde{T}_i$,...

1: SemiEmSet $\leftarrow \emptyset$; NoEmSet $\leftarrow \emptyset$;
2: Construct an IFT-Tree H with fan-out f;
3: $\varepsilon' \leftarrow \frac{\varepsilon}{U+1}$;
Sanitizing the initial dataset T
4: $TBP\text{-}Tree_{(0)} \leftarrow DiffPart(T, H, \varepsilon')$;
5: SemiEmSet$\leftarrow$semi-non empty nodes of $TBP\text{-}Tree_{(0)}$;
6: NoEmSet$\leftarrow$non empty nodes of $TBP\text{-}Tree_{(0)}$;
7: Release $\widetilde{T}$;
Sanitizing the incremental updates
7: **for** each $\Delta T_i (1 \leq i \leq U)$ **do**
8: Root of $TBP\text{-}Tree_{(i-1)} \leftarrow$ all records in ΔT_i;
9: $TBP\text{-}Tree_{(i)} \leftarrow$ **IncBuildTBP-Tree**(ΔT_i, ε', H);
10: Update SemiEmSet, NoEmSet;
11: Release leaf nodes' information of $TBP\text{-}Tree_{(i)}$;
12: Return $\widetilde{T}$, $\widetilde{T}_i$,...;

4.1 The Initial Dataset Sanitization

In principle, we can use any set-valued data sanitization algorithm to sanitize the initial dataset, as long as the sanitized results are differentially private. For example, given the initial dataset T, ε', and H, we use DiffPart method to release $\widetilde{T}$. The method recursively distributes the records in T into disjoint sub-datasets with more specific representations

in a top-down manner. In this method, $\frac{\varepsilon'}{2}$ budget is used to guide the partitioning process of sub-datasets, and the rest $\frac{\varepsilon'}{2}$ plus the budget left from the partitioning process to construct the release in the leaf nodes. In the top-down partitioning manner, all the records in T can be generalized a common generalization, called *hierarchy cut* which consists of a set of nodes in H. A record can be generalized to a hierarchy cut if every item in the record can be generalized to a node in the cut and every node in the cut generalizes some items in the record. For example, the record t_1=$\{I_1, I_2\}$ in Table 1 can be generalized to the cuts $\{I_{\{1,2\}}\}$ and $\{I_{\{1,2,3,4\}}\}$, but not $\{I_{\{1,2\}}, I_{\{3,4\}}\}$, while t_4=$\{I_1, I_3\}$ in Table 2 can be generalized to the cut $\{I_{\{1,2\}}, I_{\{3,4\}}\}$. Fig.2 demonstrates partitioning the three records T=$\{t_1, t_2, t_3\}$ into three leaf nodes of the TBP-tree such that each node contains the noisy count (e.g., 1 is the noisy count of the leftmost leaf node). Every node in the TBP-Tree consists of three fields: *hierarchy cut*, *records*, and *nc*, where *hierarchy cut* denotes the generalization of children of the node, *records* registers which records contain the hierarchy cut or its subsets, and *nc* refers to the accumulated noisy counts from the initial dataset T to the current update ΔT_i.

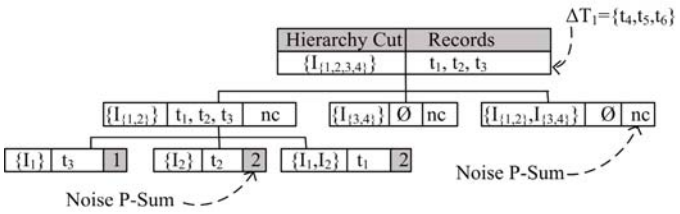


Fig. 2. The TBP-Tree on the initial dataset T

To conveniently partition the nodes in TBP-Tree, we propose an *noisy p-sum* mechanism to record the accumulated noisy counts. That is, each node v (or a *hierarchy cut*) in the TBP-Tree is associated with a *p-sum*(v). After a new batch of incremental updates, the *IncTDPart* method will release noisy versions of these *p-sums* in leaf nodes in TBP-Tree.

Definition 6. ***(noisy p-sum)****. The noisy p-sum of a hierarchy cut in each node of TBP-Tree is the number of records in consecutive updates which contain the hierarchy cut or its subsets. Let* $1 \leq m \leq n$*. We use the notation* $\sum[m,n] = \sum_{j=m}^{n} nc_j(\{I_i\})$ *to denote the noisy p-sum involving the hierarchy cut* $\{I_i\}$ *m trough n, where m and n denote the update timestamps in the sequence of updates.*

For example, in Fig.2, the *p-sum* of the hierarchy cut $\{I_2\}$ is $\sum[0,0]$=$nc_0(\{I_i\})$=2. According to the definition, the $\sum[m,n]$ of some hierarchy cut can be computed by $\sum[1,n]$-$\sum[1,m]$. This is quite consentient for researchers to obtain the noisy counts at every update step or any range of updates.

4.2 TBP-Tree Incremental Construction

Our strategy for *IncBuildTBP-Tree* is to recursively group records in ΔT_i into disjoint subsets based on H, either allocate records into relative nodes of the previous $TBP\text{-}Tree_{(i-1)}$, or create some new nodes that do not exist in the previous tree. Procedure 1

presents the details of *IncBuildTBP-Tree*. To incrementally build *TBP-Trees* based on the coming updates, we employ a uniform privacy budget allocation scheme, that is, divide the total privacy budget ε into equal portions $\varepsilon'=\frac{\varepsilon}{U+1}$, each is used for incrementally maintaining a *TBP-Tree*. For constructing the nodes of the *TBP-Tree* in each update, we use the same budget allocation scheme as DiffPart method that reserves $\frac{\varepsilon'}{2}$ to generate the noisy sizes of leaf nodes, and the rest $\frac{\varepsilon'}{2}$ to guide the top-down partitioning process.

Procedure 1. IncBuildTBP-Tree

Input: ΔT_i: The i-th update; ε': The allocated privacy budget for i-th update; H: The IFT-Tree; $TBP\text{-}Tree_{(i-1)}$: The (i-1)-th TBP-Tree after the (i-1)-th update;
Output: The i-th taxonomy-based partition tree $TBP\text{-}Tree_{(i)}$

1: Vector $V_1 \leftarrow$all sub-partitions of $T, \Delta T_1, \Delta T_2, ..., \Delta T_{i-1}$;
2: Partition $p \leftarrow$all records in ΔT_i;
3: Add p to the root of $TBP\text{-}Tree_{(i-1)}$;
4: $p.cut \leftarrow$ the root of H;
5: $p.\widetilde{\varepsilon'}=\frac{\varepsilon'}{2}$; $p.\alpha=\frac{p.\widetilde{\varepsilon'}}{|InternalNodes(p.cut)|}$;
6: Select a node v from $p.cut$ to partition;
7: Generate all non-empty sub-partition to P;
8: Allocate record in ΔT_i to P;
9: **for** each sub-partition $p_i \in P$ **do**
10: **if** $p_i \in V_1$ **then**
11: **if** $p_i \in$ NoEmSet **and** p_i is not a leaf node **then**
12: **FollowPreviousTraces**($p_i, H, p.\alpha$);
13: **if** $p_i \in$ SemiEmSet **and** $p_i.(\sum_{j=0}^{i-1} nc_j + nc_i) \geq \theta_1$ **then**
14: $p_i.\widetilde{\varepsilon'}=p.\widetilde{\varepsilon'}$-$p.\alpha$;
15: $p_i.\alpha=\frac{p_i.\widetilde{\varepsilon'}}{|InternalNodes(p_i.cut)|}$;
16: Add p_i to V_1;
17: **else**
18: **if** $p_i.nc_i \geq \theta_1$ **then**
19: Repeat Lines 14-16;
20: **for** $j=1$, $j \leq 2^l$-$|P|$ **do** // l is the number of v's children
21: **if** $p_j.nc \geq \theta_1$ **then**
22: Randomly generate an empty sub-partition p'_j;
23: Repeat Lines 14-16;
24: **if** $p_i.nc_i \geq \theta_2$ **and** p_i is a leaf node **then**
25: Add nc_i copies of $p_i.cut$ to $\widetilde{T}_i$;
26: **else**
27: Add p_i to V_1;
28: Return $TBP\text{-}Tree_{(i)}$;

To make the partition process easier in the updates, we define *Non-Empty node* and *Semi-Non Empty node* in each *TBP-Tree*.

Definition 7. *(**Non-Empty node and Semi-Non Empty node**). Let θ_1 be a pre-defined threshold. Give any non-leaf node v of the $TBP\text{-}Tree_{(i)}$, and let $p\text{-}sum(v)=\sum_{j=0}^{i-1} nc_j + nc_i$ be its current accumulated noisy counts. The two concepts are defined as follow.*

$$v = \begin{cases} \textit{Non-Empty node}, \textit{if } \textit{p-sum}(v) \geq \theta_1 \\ \\ \textit{Semi-Non Empty node}, \textit{if } 0 < \textit{p-sum}(v) < \theta_1 \end{cases} \tag{5}$$

NoEmSet and SemEmSet are two node sets that consist of Non-Empty nodes, and Semi-Non Empty nodes, respectively.

As the description of Procedure 1, on the i-th update ΔT_i arriving, we first insert the records in ΔT_i into the root of $TBP\text{-}Tree_{(i-1)}$, and recursively partitions them into disjoint subsets. If some records is added to those non-leaf nodes which are non-empty, we call *FollowPreviousTraces* to track the children traces of the non-leaf nodes in $TBP\text{-}Tree_{(i-1)}$, and allocate relative records (Lines 11-12). To some records are allocated to semi-non empty nodes which are in internal levels, we check whether the *p-sums* of the semi-non empty nodes exceed the threshold θ_1. If so, then we either further to partition the nodes or start to construct new releases in leaf level (Lines 14-16). If some records are added to new nodes that do not exist in $TBP\text{-}Tree_{(i-1)}$, Lines 18-19 are called to check whether to further partition or release these new nodes.

During partitioning ΔT_i, many empty nodes will be generated, which are associated with zero number of records. It is critical to prune out the empty nodes in that may lead to poor utility of the release. [20] has indicated that the number of empty nodes k follows the *binomial distribution* $B(m, p_{\theta_1})$, where m is the total number of empty nodes we have to check and $p_{\theta_1}=\frac{exp(-\alpha\theta_1)}{2}$. We can select k uniformly random empty nodes without replacement with noisy counts sampled from the cumulative distribution function $P(x)$=1-$exp(\alpha\theta_1 - \alpha x)$ $(\forall x \geq \theta_1)$. Lines 20-23 show the details of how to generate the empty nodes. Each non-leaf partition $p_i.cut$ in Procedure 1 tracks its unused privacy budget $\widetilde{\varepsilon}'$ and computes the portion of privacy budget α for the next partition operation.

To calculate the budget α, we have to obtain the maximum number of partition operations $InternalNodes(p_i.cut)$ from $p_i.cut$ to leaf nodes. Chen et al. [7] points out that $|InternalNodes(p_i.cut)|=\sum_{u_i \in cut} |InternalNodes(u_i,H)|$, where $|InternalNodes(u_i,H)|$ denotes the number of internal node of the subtree of H rooted at u_i. After the i-th update, in the leaf node level of the $TBP\text{-}Tree_{(i)}$, if a leaf node p_i satisfying $p_i.nc_i \geq \theta_2$ (Lines 24-25), we add nc_i copies of p_i to the $\widetilde{T}_i$. Fig.3 shows the sanitized release after the ΔT_1, ΔT_2 appended the initial dataset T.

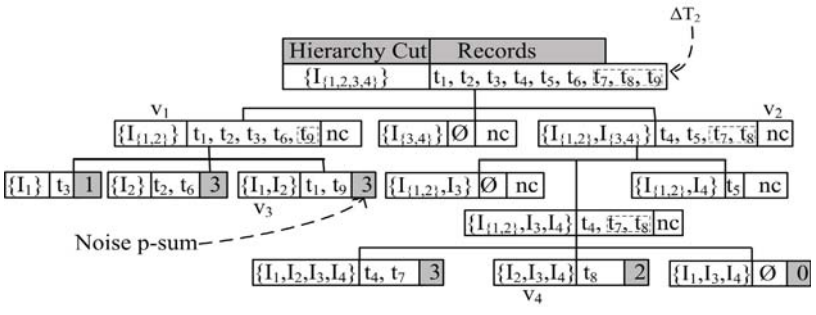


Fig. 3. The TBP-Tree after ΔT_1, ΔT_2 updates

Example 2. Given the update ΔT_2 in Table 5, the privacy budget ε', Procedure 1 works as follows (see Fig.3 for an illustration). It first distributes t_9 into the v_1 node, and t_7, t_8 into the v_2 node. Due to the same partitioning traces of the v_1 node in $TBP\text{-}Tree_{(2)}$ as $TBP\text{-}Tree_{(1)}$, we directly add t_9 to the v_3 leaf node. The p-sum(v_3) is 3 that equals $nc_0(v_3)+nc_2(v_3)$. The v_2 node in $TBP\text{-}Tree_{(1)}$ is semi-non empty node in terms of p-sum$(v_2) \leq \theta_1$. The inserted t_7, t_8 records change the node into a non empty node. So, we further split the v_2 node, and obtain three leaf nodes (e.g., the v_4 node).

4.3 Thresholds θ_1, θ_2 Computation

A node in the $TBP\text{-}Tree_{(i)}$ is further expanded if its *p-sum* value is not less than the threshold θ_1. In the static scenario of *TBP-Tree*, the threshold of non-empty and empty nodes $\theta_1=\frac{C1\sqrt{2}}{\alpha}$ (constant times of the standard deviation of noise), and the threshold of leaf nodes publication $\theta_2=\frac{C2\sqrt{2}}{\alpha}$, where α is the privacy budget assigned to the nodes. However, for the incremental updates, θ_1 and θ_2 may be changed at each time of update. A straightforward scheme would be using a fixed multiple value of the standard deviation of noise, that is, in different nodes of each *TBP-Tree* , one can use multiple value of standard deviation of noise to guide the partitions in terms of the budget α. Differing from the simple method, we propose an more significant scheme using the *mean* of multiple value of the standard deviation of noise. Let $\alpha_i,...,\alpha_j$ be the allocated privacy budgets in the series of datasets $T,\Delta T_1,...,\Delta T_U$ for a non-empty or empty node. The threshold θ_1 (i.e., *mean*) can be defined as follow. According to the same idea as θ_1, we can obtain θ_2. Here, $C1,C2$ are two constants defined by data publishers.

$$\theta_1 = \frac{\sum_{m=i}^{j} \frac{\sqrt{2}C1}{\alpha_m}}{j-i+1} \tag{6}$$

$$\theta_2 = \frac{\sum_{n=i}^{j} \frac{\sqrt{2}C2}{\alpha_n}}{j-i+1} \tag{7}$$

Example 3. In Fig.3, there are four records t_4,t_5,t_7,t_8 in node v_2, two of which came from ΔT_1, the other two from ΔT_2. In $TBP\text{-}Tree_{(1)}$, the privacy budget assigned to v_2 is $\alpha_1=\frac{\varepsilon'}{6}$, and $\alpha_2=\frac{\varepsilon'}{6}$ in $TBP\text{-}Tree_{(2)}$. According to the above equations, we get $\theta_1=\frac{3\sqrt{2}C1}{\varepsilon'}$. Due to p-sum$(v_2) \geq \theta_1$, we further partition the node v_2.

4.4 Analysis

Privacy Analysis. We give the differential privacy guarantee of our method below.

Theorem 2. *Each of the updates of IncTDPart algorithm is $\frac{\varepsilon}{U+1}$-differentially private, where U is the upper bound on the number of the updates.*

Proof. We prove the theorem by the definition of ε-differential privacy. Consider two neighboring datasets T_1 and T_2. We first consider Lines 7-11 of Algorithm 1, that is, the incremental construction of TBP-Tree. Let this part be denoted by Ag. Given any update

timestamp $i(1 \leq i \leq U)$, and ε' ($\varepsilon'=\frac{\varepsilon}{U+1}$), and let the i-th TBP-Tree be denoted by $Tree_i$. In essence, $Tree_i$ is constructed on the noisy answers to a set of incremental counting queries. Let each root-to-leaf path be indexed by k. We denote a node in level l and path k by v_{kl}, its privacy budget by ε'_{kl}, and its incremental count in T_1 and T_2 by $Q(T_1)_{kl}$ and $Q(T_2)_{kl}$, respectively. We first claim that a single record can only affect at most 2^c ($c \leq f$) root-to-leaf paths, where 2^c is the children number of the node v_{kl}. Then we have

$$\begin{aligned}
&\frac{Pr(Ag(T_1)=Tree_i)}{Pr(Ag(T_2)=Tree_i)} = \prod_{l=1}^{h}\prod_{k=1}^{2^c} \frac{exp(-\varepsilon'_{kl}\frac{|TC(v_{kl})-Q(T_1)_{kl}|}{2^c})}{exp(-\varepsilon'_{kl}\frac{|TC(v_{kl})-Q(T_2)_{kl}|}{2^c})} \\
&\leq exp\left(\frac{\sum_{l=1}^{h}\sum_{k=1}^{2^c}\varepsilon'_{kl}|Q(T_1)_{kl}-Q(T_2)_{kl}|}{2^c}\right) \\
&\leq exp\left(\frac{1}{2^c}\sum_{l=1}^{h}\sum_{k=1}^{2^c}\varepsilon'_{kl}\right)
\end{aligned}$$

where $TC(v_{kl})$ is the true count of the node v_{kl}, h is the height of the $Tree_i$. Since $\sum_l \varepsilon'_{kl}=\varepsilon'$, we have $\frac{Pr(Ag(T_1)=Tree_i)}{Pr(Ag(T_2)=Tree_i)} \leq exp(\varepsilon')$.

The use of privacy budget on different updates follows *sequential composition* [18].

Theorem 3. ***(Sequential Composition)****. If a randomized algorithm Ag runs a sequence of $Ag_1(T)$, $Ag_2(T)$,..., $Ag_U(T)$ over the dataset T, where each Ag_i provides ε_i differential privacy, then Ag(T) is $\sum_{i=1}^{U}\varepsilon_i$-differentially private.*

We now show that Algorithm 1 satisfies ε-differential privacy.

Theorem 4. *IncTDPart algorithm is ε-differentially private.*

Proof. Given the upper bound U on updates. According to the Theorem 2, we obtain that each of the updates of IncTDPart algorithm satisfies $\frac{\varepsilon}{U+1}$-differential privacy. Besides the initial dataset, we run IncTDPart algorithm $U+1$ times. According to Theorem 3, we get $\sum_{i=1}^{U+1}\frac{\varepsilon}{U+1}=\varepsilon$. Therefore, IncTDPart algorithm is ε-differentially private.

5 Experiments

In this section, we evaluate the utility of our proposed algorithm in terms of utility for incremental counting queries, and examine the scalability of our method for processing large-scale datasets. Our implementation was done in C++, and all experiments were performed on an Intel Core 2 Duo 2.94GHz Pc with 4GB RAM. Extensive experiments were performed on two real datasets, *MSNBC* [21], and *kosarak* [22], which record the URL categories visited by users in time order, and clickstream data, respectively.

Table 6. Shows more details of the two datasets

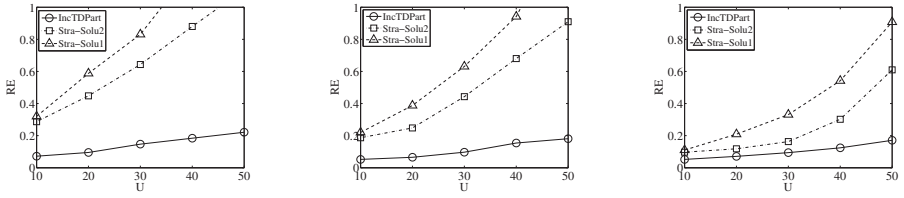
Dataset	***N***	$\|I\|$	$Avg\|t\|$
MSNBC	989,818	17	1.72
Kosarak	990,002	41,270	8.1

The characteristics of the datasets are summarized in Table 6, where N is the number of records in each datasets, $|I|$ the number of distinct items and $Avg|t|$ the average record length. We compare the utility and scalability of our method IncTDPart described in Algorithm 1 to the two straightforward solutions discussed in Section 1. We use Stra-Solu1 and Stra-Solu2 to denote the two basic methods, respectively.

5.1 Utility

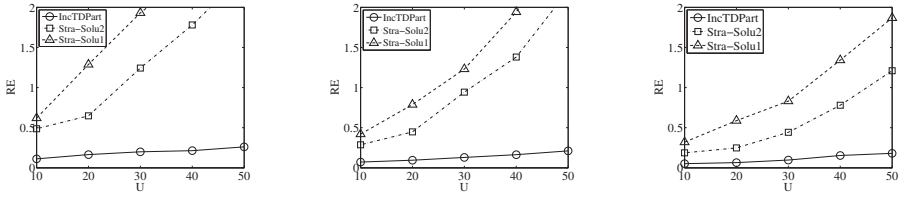
Based on the above two datasets, we perform our experiment to demonstrate the utility of our method under different number of updates. In the experiment, we randomly chose 400,000 records as the initial dataset T from *MSNBC* and *Kosarak*, respectively, and chose another 10,000 records without replacement as the incremental update ΔT_i.

Effect of U on Utility. In the first set of experiments, we examine the relative error (RE) of incrementally counting queries on the sanitized datasets under varying the upper bound U from 10 to 50, and fixing f=10. For each dataset, we randomly generate a counting query whose items is drawn from I. Fig. 4 and Fig.5 show the relative error of IncTDPart, Stra-Solu1, and Stra-Solu2 with respect to different privacy budget ε. The relative error decreases when ε varies from 0.5 to 1.0 because less Laplace noise is injected. From the two figures, we can see that the error of the two straightforward solutions are larger than that of IncTDPart algorithm in all cases. When the upper bound U increases, the performances of Stra-Solu1 and Stra-Solu2, especially Stra-Solu1, deteriorate sharply because they do not employ the update-bounded mechanism to incrementally release the sanitized datasets.



(a) ε=0.5 (b) ε=0.75 (c) ε=1.0

Fig. 4. Relative error under MSNBC dataset



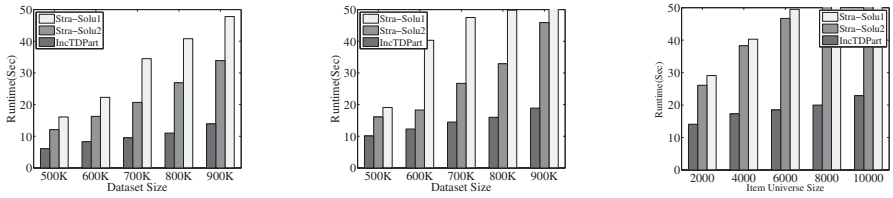
(a) ε=0.5 (b) ε=0.75 (c) ε=1.0

Fig. 5. Relative error under Kosarak dataset

5.2 Scalability

In the last set of experiments, we examine the scalability of our method against Stra-Solu1 and Stra-Solu2. The runtime is used as our performance metric, which is dominated by the datasets size and item universe size. We first evaluate the efficiency of the three algorithms by varying the dataset size from 500K to 900K and setting ε=1.0, U=50, and f=10. Fig.6(a) and Fig.6(b) show the runtime of the three methods under the

two datasets *MSNBC* and *Kosarak*. As dataset size grows, in particular, the size exceeds 600K, the runtime of Stra-Solu1 and Stra-Solu2 increase more dramatically. This is because when increasing dataset size, the two methods have to take more time to partition numerous candidate nodes under lacking the help of taxonomy-based partitioning tree. As expected, the runtime of our method is linear of the dataset size. Fig.6(c) presents how the runtime varies under different item universe size on the dataset *Kosarak*, where ε=1.0, U=50, and f=10. From the Fig.6(c), it can be seen that the runtime of our algorithm scales linearly with the increase of universe size, however, the runtime of Stra-Solu1 and Stra-Solu2 grows quickly.



(a) Runtime on MSNBC (b) Runtime on Kosarak (c) Runtime on Kosarak

Fig. 6. Scalability under MSNBC and Kosarak datasets

6 Conclusions

In this paper, we have studied the problem of releasing set-valued data against incremental updates in the framework of differential privacy. Based on the update-bounded mechanism, we first proposed an efficient algorithm IncTDPart for answering the incrementally counting queries with the help of taxonomy-based partitioning tree. We dynamically maintained the tree to partition the incremental records. Then we proved our algorithm that satisfied ε-differential privacy. Experiments on real datasets show that our algorithm outperforms the two straightforward solutions. As the future work, we will investigate how to preserve ε-differential privacy against incremental updates for other application scenarios (e.g., releasing sequential data, and temporal data).

Acknowledgement. We sincerely thank Yin Yang (ADSC) for his comments.

References

1. Terrovitis, M., Mamoulis, N., Kalnis, P.: Privacy-preserving anonymization of set-valued data. In: Proceedings of the VLDB Endowment (PVLDB 2008), vol. 1(1), pp. 115–125 (2008)
2. He, Y., Naughton, J.F.: Anonymization of Set-Valued Data via Top-Down, Local Generalization. In: Proceedings of the VLDB Endowment (PVLDB 2009), vol. 2(1), pp. 934–945 (2009)
3. Ganta, S.R., Kasiviswanathan, S.P., Smith, A.: Composition attacks and auxiliary information in data privacy. In: Proc. of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2008), pp. 265–273 (2008)

4. Wong, R.C.W., Fu, A., Wang, K., Yu, P.S., Pei, J.: Can the utility of anonymized data be used for privacy breaches. ACM Transaction on Knowledge Discovery from Data (TKDD 2011) 5(3), 16 (2011)
5. Dwork, C.: Differential privacy. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) ICALP 2006, Part II. LNCS, vol. 4052, pp. 1–12. Springer, Heidelberg (2006)
6. Kifer, D., Machanavajjhala, A.: No free lunch in data privacy. In: Proc. of the 31th International Conference on Management of Data (SIGMOD 2011), pp. 193–204 (2011)
7. Chen, R., Mohammed, N., Fung, B.C.M., Desai, B.C., Xiong, L.: Publishing Set-Valued Data via Differential Privacy. In: PVLDB 2011, vol. 4(11), pp. 1087–1098 (2011)
8. Chen, R., Fung, B.C.M., Desai, B.C., Sossou, N.M.: Differentially private transit data publication: a case study on the montreal transportation system. In: Proc. of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2012), pp. 213–221 (2012)
9. Dwork, C., Naor, M., Pitassi, T., Rothblum, G.N.: Differential pirvacy under continual observation. In: Proc. of the 42nd ACM Symposium on Theory of Computing (STOC 2010), pp. 715–724 (2010)
10. Chan, T.-H.H., Shi, E., Song, D.: Private and continual release of statistics. ACM Transactions on Information and System Security (TISS 2011) 14(3), 26 (2011)
11. Dwork, C., McSherry, F., Nissim, K., Smith, A.: Calibrating noise to sensitivity in private data analysis. In: Halevi, S., Rabin, T. (eds.) TCC 2006. LNCS, vol. 3876, pp. 265–284. Springer, Heidelberg (2006)
12. Xu, J., Zhang, Z., Xiao, X., Yang, Y., Yu, G.: Differentially private histogram publication. In: Proc. of the 28th International Conference on Data Engineering (ICDE 2012), pp. 32–43 (2012)
13. Hay, M., Rastogi, V., Miklau, G., Suciu, D.: Boosting the accuracy of differentially private histogram through consistency. In: Proceedings of the VLDB Endowment (PVLDB 2010), vol. 3(1), pp. 1021–1032 (2010)
14. Götz, M., Machanavajjhala, A., Wang, G., Xiao, X., Gehrke, J.: Publishing search logs - a comparative study of privacy guarantees. IEEE Transaction Knowledge and Data Engineering (TKDE 2012) 24(3), 520–532 (2012)
15. McSherry, F., Mironov, I.: Differentially private recommender systems: building privacy into the netflix prize contenders. In: Proc. of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2009), pp. 627–636 (2009)
16. Byun, J.-W., Sohn, Y., Bertino, E., Li, N.: Secure anonymization for incremental datasets. In: Jonker, W., Petković, M. (eds.) SDM 2006. LNCS, vol. 4165, pp. 48–63. Springer, Heidelberg (2006)
17. Xiao, X., Tao, Y.: m-invariance: Towards privacy preserving republication of dynamic datasets. In: Proc. of the 27th International Conference on Management of Data (SIGMOD 2007), pp. 689–700 (2007)
18. McSherry, F.: Privacy integrated queries: an extensible platform for privacy-preserving data analysis. In: Proc. of the 29th International Conference on Management of Data (SIGMOD 2009), pp. 19–30 (2009)
19. Xiao, X., Bender, G., Hay, M., Gehrke, J.: iReduct: differential privacy with reduced relative errors. In: Proc. of the 31th International Conference on Management of Data (SIGMOD 2011), pp. 229–240 (2011)
20. Gormode, G., Procopiuc, M., Srivastava, D.: Differentially private summaries for sparse data. In: Proc. of the 15th International Conference on Database Theory (ICDT 2012), pp. 299–311 (2012)
21. UCI machine learning repository, `http://archive.ics.uci.edu/ml`
22. Frequent itemset mining dataset repository, `http://fimi.ua.ac.be/data/`