Journal of Privacy and Confidentiality
Vol. 11 (3) 2021 Privacy Challenges

Submitted Jan. 11, 2021
Published Dec 2021

SYNTHETIC DATA GENERATION WITH DIFFERENTIAL PRIVACY VIA BAYESIAN NETWORKS

ERGUTE BAO, XIAOKUI XIAO, JUN ZHAO, DONGPING ZHANG, AND BOLIN DING

National University of Singapore
e-mail address: ergute@comp.nus.edu.sg

National University of Singapore
e-mail address: xkxiao@nus.edu.sg

Nanyang Technological University of Singapore
e-mail address: junzhao@ntu.edu.sg

National University of Singapore
e-mail address: d-zhang@comp.nus.edu.sg

Alibaba Group
e-mail address: bolin.ding@alibaba-inc.com

Abstract. This paper describes PrivBayes, a differentially private method for generating synthetic datasets that was used in the 2018 Differential Privacy Synthetic Data Challenge organized by NIST.

Introduction

This paper describes PrivBayes, a differentially private method for synethetic data generation that was used in our entry to the NIST Differential Privacy Synthetic Data Challenge 2018. PrivBayes is based on a method of the same name published in Zhang et al. (2017), but incorporates a number of modifications to improve its data utility and efficiency.

The high-level idea of PrivBayes is as follows. Given a sensitive input dataset, PrivBayes first constructs a Bayesian network, which (i) provides a private and succinct model of the correlation among the attributes in the input dataset and (ii) allows us to approximate the full distribution of the input dataset using a set of low dimension distributions. After that, PrivBayes adds noise into each low dimension distribution, resulting in the noisy approximations of the original distributions. Note that both the construction of the Bayesian network and the approximation of the low dimension distributions are conducted in a manner that satisfies differential privacy. Finally, PrivBayes generates synthetic tuples using the Bayesian network and noisy distributions, and releases them. The main advantage of

Key words and phrases: Differential privacy, synthetic data generation, Bayesian network.
Code accompanying this article can be found at Zhao and Zhang (2021).

Figure 1. Example Bayesian network $\mathcal{N}_1$.

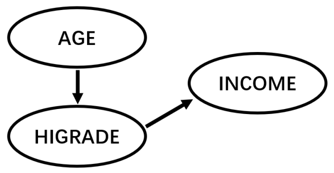

Table 1. Dependence of network $\mathcal{N}_1$.

Attribute	**Parents**
age	$\varnothing$
higrade	{age}
income	{higrade}

PrivBayes is that it circumvents the curse of dimensionality by injecting noise into the low dimension marginal distributions instead of the original high dimension distribution.

In the remainder of the paper, we will first review the problem definition and the basics of Bayesian networks and differential privacy; after that, we detail our implementation of PrivBayes.

1. Preliminaries

1.1. **Problem Definition.**

Let D be an input dataset consisting of n participants, each described by d attributes. We denote the set of attributes as $\mathcal{A}$, with $|\mathcal{A}| = d$. Then dataset D can also be regarded as a joint probability distribution over $\text{dom}(\mathcal{A})$. We denote this probability distribution specified by D as $P(\mathcal{A})$. The goal is to release a synthetic dataset D^*, which also specifies a distribution over $\text{dom}(\mathcal{A})$, denoted as $P^*(\mathcal{A})$, such that $P^*(\mathcal{A})$ resembles $P(\mathcal{A})$ while preserving the privacy of individual participants in the input dataset D.

1.2. **Bayesian Networks.**

A Bayesian network over a set of random variables is a way to compactly describe their joint distribution, by specifying conditional independence among certain random variables. In the rest of this paper, we abuse notation and use A_i to represent both a particular attribute A_i itself and also the random variable which takes on value from the domain of attribute A_i, when the context is clear. Now we can define a Bayesian network $\mathcal{N}$ (Koller and Friedman, 2009) over a set of attributes $\mathcal{A}$ as a fully connected set of attributes $\{A_1, ..., A_k\}$ and a set of *attribute-parent (AP) pairs*, $\{(A_{k+1}, \Pi_{k+1}), \ldots, (A_d, \Pi_d)\}$, for a certain $k \geqslant 1$. The fully connected set of attributes and the $(d-k)$ AP pairs in $\mathcal{N}$ (accompanied with the low dimension distributions) essentially define a way to approximate $P(\mathcal{A})$ using a joint distribution $P(A_1, \ldots, A_k)$ and $(d-k)$ conditional distributions $P(A_1 \mid \Pi_1), P(A_2 \mid \Pi_2), \ldots, P(A_k \mid \Pi_k)$. Intuitively, if the network accurately captures the conditional independence among the random variables in the distribution, then $P_{\mathcal{N}}(\mathcal{A})$ would be a good approximation of $P(\mathcal{A})$. We express $P_{\mathcal{N}}(\mathcal{A})$ as follows:

$$P_{\mathcal{N}}(\mathcal{A}) = P(A_1, ..., A_k) \cdot \prod_{i=k+1}^{d} P(A_i \mid \Pi_i).$$

For example, consider a dataset D_1 containing three attributes 'age', 'higrade', and 'income', where 'income' and 'age' are numerical attributes representing the income and age of a participant, respectively. For simplicity, we assume that each numeric attribute has a domain that is discretized into a number of ranges, e.g., attribute 'age' with range $[0, 99]$ may have a discretized domain consisting of 20 subranges $[0, 4], [5, 9], \ldots, [95, 99]$. The 'higrade' attribute is a categorical attribute that represents the highest grade of school attended by the participant. For example, code 04 to 11 represents the first to eighth grade in elementary school. Let the Bayesian network $\mathcal{N}_1$ represent the conditional independence of attributes in D_1, as shown in Figure 1. In this Bayesian network, the fully connected set consists of only one attribute–'age', and there are two AP pairs ('higrade',{'age'}) and ('income',{'higrade'}), as listed in Table 1. For network $\mathcal{N}_1$, we have:

$$P_{\mathcal{N}_1}(age, higrade, income) = P(age) \cdot P(higrade \mid age) \cdot P(income \mid higrade).$$

1.3. Differential Privacy.

We say that two datasets are *neighboring* if one dataset can be obtained by adding a record to the other dataset. Hence, the numbers of records in our notion of neighboring datasets differ by 1. Differential privacy requires that any release of information about an input dataset should be done via a randomized algorithm $\mathcal{M}$, such that the output of $\mathcal{M}$ does not reveal much information about any particular participant (tuple) in the input dataset. Roughly speaking, the output distributions of $\mathcal{M}$ should be similar on any two neighboring input datasets. The formal definition of differential privacy is as follows:

Definition 1.1 ((ε, δ)-Differential Privacy (Dwork et al., 2006))**.** A randomized algorithm $\mathcal{M}$ satisfies (ε, δ)-differential privacy, if for any two neighboring datasets D_1 and D_2, and for any subset $\mathcal{O}$ of the output domain of $\mathcal{M}$, we have

$$\Pr\left[\mathcal{M}(D_1) \in \mathcal{O}\right] \leqslant e^{\varepsilon} \cdot \Pr\left[\mathcal{M}(D_2) \in \mathcal{O}\right] + \delta. \tag{1.1}$$

In particular, $\mathcal{M}$ satisfies ε-differential privacy when $\delta = 0$.

To achieve (ε, δ)-differential privacy, we will use the analytic *Gaussian mechanism* (Balle and Wang, 2018).

Lemma 1.2 (Analytic Gaussian Mechanism (Balle and Wang, 2018))**.** *Let $F : \mathcal{D} \to \mathbb{R}^d$ be a function. The analytic Gaussian mechanism that injects Gaussian noise $\mathcal{N}\left(\mathbf{0}, \sigma^2 \cdot \mathbf{I}\right)$ into the output of F satisfies (ϵ, δ)-differential privacy, if and only if*

$$\frac{S(F)}{\sigma} \leqslant \sqrt{2}\left(\sqrt{\chi^2 + \epsilon} - \chi\right), \tag{1.2}$$

where $\mathbf{0}$ and $\mathbf{I}$ are a zero vector and a $d \times d$ identity matrix, respectively, and χ is the solution to

$$\operatorname{erfc}(\chi) - \exp(\epsilon) \cdot \operatorname{erfc}\left(\sqrt{\chi^2 + \epsilon}\right) = 2\delta, \tag{1.3}$$

and erfc() *denotes the complementary error function, i.e.,*

$$\mathrm{erfc}(x) \triangleq 1 - \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2}\,\mathrm{d}t.$$

Note that Lemma 1.2 is in a slightly different form from the original version (Balle and Wang, 2018). A proof sketch of Lemma 1.2 is provided in Appendix A.1

2. Solution Overview

In a nutshell, PrivBayes runs in three phases:

1. (*Network learning*) Construct a Bayesian network $\mathcal{N}$ over the attributes in D using the analytical Gaussian mechanism. Recall that the Bayesian network is represented as a fully connected set of attributes and a set of *attribute-parent (AP) pairs*.
2. (*Distribution learning*) Generate the corresponding joint and conditional distributions for the Bayesian network learned in the first phase using the analytical Gaussian mechanism.
3. (*Data synthesis*) From the results of network learning and distribution learning, we derive an approximate distribution for D, and then generate tuples from the approximate distribution to form a synthetic dataset D^*.

This paper is different from Zhang et al. (2017) in the following ways. First, we consider the easy-to-compute score function for attribute pairs (see Eq. (2.1) and (2.2)) suggested in Section 5.3 of Zhang et al. (2017). Next, we consider a more generalized version of the Bayesian network. In particular, the head of the network may consist of multiple attributes in this paper (see Section 2.1.2) while there is only a single head in the original solution. In addition, we present more detailed explanations on enforcing consistencies on the noisy low dimension distributions and tuple generation in Section 3. Finally, we adopt (ϵ, δ)-DP and rely heavily on the analytic Gaussian mechanism (Balle and Wang, 2018) in this paper while the original solution preserves ϵ-DP and is based on the Laplace mechanism (Dwork, 2006).

Next, we elaborate on the three phases of PrivBayes in detail.

2.1. Network Learning.

2.1. Network Learning. The network learning phase of PrivBayes consists of two steps. In the first step, we compute a differentially private *score* for each pair of attributes, such that a large score tends to indicate that the attribute pairs are highly correlated, and vice versa. Next, we derive a Bayesian network based on the differentially private scores, without using additional information from the input dataset. In other words, the second step does not consume any privacy budget. In Sections 2.1.1 and 2.1.2, we present the details of the first and second step, respectively.

2.1.1. *Computing the Score of each Attribute Pair.* Given two **attributes** A_i and A_j, a canonical approach to measure the correlation between A_i and A_j is to compute the mutual information between **random variables** A_i and A_j. Formally, let A_i and A_j be random variables defined over space $\mathcal{A}_i$ and $\mathcal{A}_j$, respectively; let $P(A_i, A_j)$ be the joint distribution of attributes A_i and A_j; and let $P(A_i) \otimes P(A_j)$ be the product distribution, both determined by the input dataset D. The mutual information (Cover and Thomas, 2001) between random variables A_i and A_j is defined as

$$I(A_i, A_j) = D_{KL}(P(A_i, A_j) \| P(A_i) \otimes P(A_j)),$$

Algorithm 1: Generating noisy scores for attribute pairs

```
Data: The original dataset $D$
Result: The differentially private score of each attribute pair
foreach attribute pair $(A_i, A_j)$ do
    compute $\widetilde{\mathcal{S}}(A_i, A_j) \leftarrow \mathcal{S}(A_i, A_j) + \mathcal{N}(0, {\sigma_{NL}}^2)$;
end
return $\widetilde{\mathcal{S}}(A_i, A_j)$ for all $i, j$;
```

where D_{KL} is the Kullback–Leibler divergence. As pointed out in Zhang et al. (2017), however, I has a relatively large sensitivity, which makes it difficult to be utilized effectively under differential privacy. We consider an alternative. We let n be the number of tuples in D, then the score of attribute pairs A_i and A_j is defined as n times the total variation distance between distributions $P(A_i, A_j)$ and $P(A_i) \otimes P(A_j)$. Formally,

$$\mathcal{S}_D(A_i, A_j) \triangleq n \cdot \delta \left(P(A_i, A_j), P(A_i) \otimes P(A_j)\right), \tag{2.1}$$

where $\delta_{P,Q}$ denotes the total variation distance (TVD) between distributions P and Q defined over the same sample space Ω, defined as follows:

$$\delta_{P,Q} = \sup_{X \subset \Omega} |P(X) - Q(X)|.$$

We omit the subscript D in $\mathcal{S}_D$ when the context is clear. To demonstrate the computation for the right hand side of Eq. (2.1), we let $C[X_i]$, $C[X_j]$, and $C[X_i, X_j]$ represent the number of participants whose value of attribute A_i, A_j, and attribute pair $A_i \times A_j$ belongs to X_i, X_j, and $X_i \times X_j$, respectively. For example, we have $C[X_i] = n \cdot \Pr[A_i \in X_i]$ by definition. If we let A_i be the attribute 'age', X_i be the set $\{[10, 14], \ldots, [45, 49]\}$, then $C[X_i]$ represents the number of individuals whose age falls in the range of $[10, 49]$ in the dataset D. In general, $\mathcal{S}(A_i, A_j)$ is computed as follows:

$$\mathcal{S}(A_i, A_j) = \sup_{X_i \times X_j \subset \mathcal{A}_i \times \mathcal{A}_j} \left| C[X_i, X_j] - \frac{C[X_i]C[X_j]}{n} \right|. \tag{2.2}$$

Observe that $\mathcal{S}(A_i, A_j)$ is minimized when A_i and A_j are independent of each other, and it tends to be large when A_i and A_j are highly correlated. Therefore, $\mathcal{S}$ could be a good score function for our purpose. We also extend $\mathcal{S}$ to measure the correlation between an attribute and a set of attributes with the formalization similar to Eq. (2.1). From Eq. (2.2), it is easy to verify that our score function is of sensitivity at most 2, which is relatively small compared with its range n. (We note that concurrent work (Zhang et al., 2021) also uses the same score function and analyzes its sensitivity.) Based on this observation, we generate a differentially private score $\widetilde{\mathcal{S}}(A_i, A_j)$ for each attribute pair A_i, A_j using Algorithm 1. In particular, Algorithm 1 generates $\widetilde{\mathcal{S}}(A_i, A_j)$ by adding Gaussian noise with variance σ_{NL}^2 to each $\mathcal{S}(A_i, A_j)$ $(i, j \in [1, d])$. We will explain our choice of σ_{NL}^2 in Section 2.4.

2.1.2. *Constructing the Bayesian Network based on Noisey Scores.* Given the noisy score $\widetilde{\mathcal{S}}_D(A_i, A_j)$ for each pair of attributes A_i, A_j, we construct a Bayesian network for D by first invoking Algorithm 2 and then Algorithm 3.

In particular, Algorithm 2 constructs the first AP pair (A_{k+1}, Π_{k+1}) in the Bayesian network $\mathcal{N}$, such that (i) the sum of noisy scores over all attribute pairs in $\Pi_{k+1} \cup \{A_{k+1}\}$

Algorithm 2: Constructing the first AP pair in the Bayesian network $\mathcal{N}$

```
Data: The noisy scores of all the attribute pairs
Result: An AP pair
Identify the attribute pair $(A_i, A_j)$ that maximizes $\frac{\widetilde{\mathcal{S}}(A_i,A_j)}{|\operatorname{dom}(A_i)|\cdot|\operatorname{dom}(A_j)|}$;
Initialize $\Pi = \{A_j\}$, and $\mathcal{A}' = \{A_i, A_j\}$;
Initialize $x = \frac{\tau}{|\operatorname{dom}(A_i)|\cdot|\operatorname{dom}(A_j)|}$;
while $\mathcal{A}' \subset \mathcal{A}$ do
    Identify the attribute $A_l$ in $\mathcal{A}\backslash\mathcal{A}'$ that maximizes $\sum_{A\in\Pi\cup\{A_i\}} \frac{\widetilde{\mathcal{S}}(A,A_l)}{|\operatorname{dom}(A)|\cdot|\operatorname{dom}(A_l)|}$;
    if $\frac{x}{|\operatorname{dom}(A_l)|} > 1$ then
        $x \leftarrow \frac{x}{|\operatorname{dom}(A_l)|}$;
        Insert $A_l$ into $\Pi$;
    end
    Insert $A_l$ into $\mathcal{A}'$;
end
return $(A_i, \Pi)$;
```

Algorithm 3: Constructing a subsequent AP pair in the Bayesian network $\mathcal{N}$

```
Data: The network $\mathcal{N}$ and the noisy scores of all the attribute pairs
Result: A subsequent AP pair in the network $\mathcal{N}$
Let $\mathcal{A}'$ be the set of attributes that appear in the AP pairs that have been
  constructed;
Initialize $AP \leftarrow \varnothing$;
foreach $A_j \in \mathcal{A}\backslash\mathcal{A}'$ do
    Identify the attribute set $\Pi' \subseteq \mathcal{A}'$ that gives the largest value sum of
     $\sum_{A\in\Pi'} \widetilde{\mathcal{S}}(A_j, A)$ through a greedy approach, while satisfying:
    $|\operatorname{dom}(\Pi' \cup \{A_j\})| \leqslant \tau$.
    Update $AP \leftarrow AP \cup \{(A_j, \Pi')\}$;
end
return the best AP pair among all candidates in $AP$.
```

is large, and (ii) the size of the domain formed by the attributes in $\Pi_{k+1} \cup \{A_{k+1}\}$ is no larger than a threshold τ. (We will discuss the setting of τ in Section 2.4.) Towards this end, Algorithm 2 adopts a greedy approach. It first selects the pair of attributes with the maximum *normalized* noisy score, defined as $\widetilde{S}_{\text{normalized}}(A_i, A_j) = \frac{\widetilde{\mathcal{S}}(A_i,A_j)}{|\operatorname{dom}(A_i)|\cdot|\operatorname{dom}(A_j)|}$. Note that $|\operatorname{dom}(A_i)|$ and $|\operatorname{dom}(A_j)|$ are public information and hence, computing the normalized noisy score from the noisy score consumes no privacy budget. After that, it iteratively selects the attribute among the remaining ones that maximizes its total normalized noisy scores, and includes it in the selected set of attributes, under the constraint that the size of domain formed by the selected attributes should be no more than τ. Then, Algorithm 2 returns the first AP pair as (A_{k+1}, Π_{k+1}). For the example dataset D_1, the first AP pair is (‘higrade’,{‘age’}), where {‘age’} is a set of only one attribute, as shown in Figure 1.

Algorithm 4: Generating differentially private joint and conditional distributions corresponding to the network $\mathcal{N}$

```
Data: The noiseless joint and conditional distributions of the original dataset
Result: The noisy version of the joint and conditional distributions
Initialize $\mathcal{P}^* = \varnothing$;
Materialize the joint distribution $P(A_1, ..., A_k)$;
Generate differentially private $P^*(A_1, ..., A_k)$ by adding noise Gaussian$(0, {\sigma_{NC}}^2)$;
for $i = k + 1$ to $d$ do
    Materialize the joint distribution $P(A_i, \Pi_i)$;
    Generate differentially private $P^*(A_i, \Pi_i)$ by adding noise Gaussian$(0, {\sigma_{NC}}^2)$;
    Set negative values in $P^*(A_i, \Pi_i)$ to 0;
    Normalize all values in $P^*(A_i, \Pi_i)$ so that they sum up to 1;
    Derive $P^*(A_i \mid \Pi_i)$ from $P^*(A_i, \Pi_i)$, and add it to $\mathcal{P}^*$;
end
return $\mathcal{P}^*$;
```

Let k be the number of attributes in Π_{k+1} of the first AP pair. We then construct $(d-k-1)$ AP pairs by invoking Algorithm 3 $(d-k-1)$ times. Specifically, in each invocation, we first identify the set $\mathcal{A}'$ of attributes that have been selected into the network. Then, for each attribute that has not been put into the network, i.e. $A_j \in \mathcal{A} \backslash \mathcal{A}'$, we identify the attribute set $\Pi' \subseteq \mathcal{A}'$ that gives the largest value sum of $\sum_{A \in \Pi'} \widetilde{\mathcal{S}_D}(A_j, A)$ through a greedy approach similar to Algorithm 2, while satisfying the following condition. The size of domain formed by the attributes in $\Pi' \cup \{A_j\}$ is no larger than τ. Finally, we take the best AP pair among all the AP pairs constructed as above. For the example dataset D_1, the second (also the last) AP pair is ('income',{'higrade'}), as shown in Figure 1.

We next explain the reason for using the greedy approach. Note that this problem instance can be viewed as a variation (the value of one item is based on the items that are already in the knapsack) of the Knapsack problem, which requires exponential time in the size of the number of attributes to find the exact optimal solution. Using a greedy approach avoids such a prohibitive computation cost. Although approximate algorithms (Lawler, 1979) have been developed for the Knapsack problem, it remains unclear how to adapt them to our setting. Proving theoretical guarantees for Algorithms 2 and 3 or designing general approximate solutions to this problem may be of independent interest.

Remark. We note that Algorithms 2 and 3 do not consume any privacy budget, since they utilize only the noisy scores of attribute pairs.

2.2. **Distribution Learning.** Let k be the number of attributes that appear in the first AP pair constructed by Algorithm 2, and $A_1, \ldots, A_k$ denote those k attributes. Let (A_{i+k+1}, Π_i) $(i \in [1, d-k-1])$ denote the i-th AP pair constructed by the i-th invocation of Algorithm 3. To construct the approximate distribution $P_{\mathcal{N}}(\mathcal{A})$ in the Bayesian network $\mathcal{N}$, we need to approximate the joint distribution $P(A_1, ..., A_k)$ and conditional distributions $P(A_i \mid \Pi_i)$ $(i \in [k+1, d])$. Algorithm 4 shows the pseudocode for approximating these distributions while preserving privacy. The algorithm first materializes the joint distribution $P(A_1, ..., A_k)$ (Line 2), and then injects Gaussian noise into $P(A_1, ..., A_k)$ to obtain a noisy distribution $P(A_1, ..., A_k)$ (Line 3). For any $i \in [k+1, d]$, the algorithm materializes joint distribution

$P(A_i, \Pi_i)$ (Line 5), and then injects Gaussian noise into $P(A_i, \Pi_i)$ to obtain a noisy distribution $P(A_i, \Pi_i)$ (Line 6). We denote these noisy distributions as $P^*(A_1, ..., A_k)$ and $\{P^*(A_i, \Pi_i)\}_{i=k+1}^{d}$. Based on $P^*(A_i, \Pi_i)$, the algorithm derives $P^*(A_i \mid \Pi_i)$ by the definition of conditional probabilities (Line 9).

Next, we explain how to add Gaussian noise to the probability distributions. We take the marginal distribution for attribute 'age' as an example. The same approach also works for a joint distribution (with more than one attribute) and conditional distributions. For any subrange of the discretized range of 'age', we can compute the number of tuples in this subrange from the input dataset. The number for all subranges form a histogram of sensitivity 1. Hence, we can add Gaussian noise with standard deviation σ_{NC} to each entry of the histogram to generate a noisy version of it. We discuss the value of σ_{NC} in Section 2.4. For example, the original histogram may look like $(1, 6, \ldots, 1)$ with a total sum of $n = 100$. Namely, there are $1, 6, \ldots, 1$ participants of age $[0, 4], [5, 9], \ldots, [95, 99]$, respectively, in the original input dataset. The noisy histogram may look like $(1, 3, \ldots, 0)$, with a total sum of 93. Note that we cannot directly divide the noisy histogram by $n = 100$ to derive the differentially private distribution for 'age', since n is sensitive information. Instead, the noisy marginal distribution for attribute 'age' is computed by $(\frac{1}{93}, \frac{3}{93}, \cdots \frac{0}{93})$, where the denominator is the sum of all entries in the noisy histogram, instead of n.

2.3. **Data Synthesis.** Given the constructed Bayesian network and the corresponding noisy joint distribution (for the fully connected set of attributes) and conditional distributions (for all AP pairs), one can derive the following closed-form expression in the approximation for $P(\mathcal{A})$.

$$P(\mathcal{A}) \approx P^*_{\mathcal{N}}(\mathcal{A}) = P^*(A_1, ..., A_k) \cdot \prod_{i=k+1}^{d} P^*(A_i \mid \Pi_i). \tag{2.3}$$

A standard approach to generate tuples from $P^*_{\mathcal{N}}(\mathcal{A})$ is to first sample $A_1, \ldots, A_k$ from the joint distribution $P(A_1, ..., A_k)$ directly, and then iteratively sample A_i from $P^*(A_i \mid \Pi_i)$ from $i = k + 1$ to d, given the previously sampled outcome. This approach, however, is not directly compatible with differential privacy. To see this, note that DP noises cause (i) the existence of negative values in the noisy marginals, and (ii) inconsistencies among marginal distributions. We illustrate this issue with the example input dataset D_1 containing three attributes 'age', 'higrade' and 'income', with Bayesian network $\mathcal{N}_1$, as shown in Figure 1. Recall attribute 'age' consists of 20 subranges $[0, 4], [5, 9], \ldots, [95, 99]$ and the 'higrade' attribute is a categorical attribute that represents the highest grade of school attended by the participant. For illustration purposes, we let there be only two categories for 'higrade', denoted as 01 and 02. The noisy marginal distributions for attribute 'age' and attributes 'age' and 'higrade' are shown in Tables 2a and 2b, respectively.

Table 2. Example noisy marginal distributions for network $\mathcal{N}_1$.

(a) Marginal distribution for 'age'

'age'	**probability**
$[0, 4]$	0.04
$[5, 9]$	-0.1
...	...
$[95, 99]$	0.08

(b) Marginal distribution for 'age' and 'higrade'

'age' \ **'higrade'**	01	02
$[0, 4]$	0.02	0.02
$[5, 9]$	-0.1	0
...	...	...
$[95, 99]$	0	0

Table 3. Processed noisy marginal distributions for network $\mathcal{N}_1$.

(a) Marginal distribution for 'age'

'age'	**probability**
$[0, 4]$	0.05
$[5, 9]$	0
...	...
$[95, 99]$	0.1

(b) Marginal distribution for 'age' and 'higrade'

'age' \ **'higrade'**	01	02
$[0, 4]$	0.025	0.025
$[5, 9]$	0	0
...	...	...
$[95, 99]$	0	0

Table 4. Noisy marginal distributions for network $\mathcal{N}_1$ that are consistent on attribute 'age'.

(a) Marginal distribution for 'age'

'age'	**probability**
$[0, 4]$	0.05
$[5, 9]$	0
...	...
$[95, 99]$	0.067

(b) Marginal distribution for 'age' and 'higrade'

'age' \ **'higrade'**	01	02
$[0, 4]$	0.025	0.025
$[5, 9]$	0	0
...	...	...
$[95, 99]$	0.0335	0.0335

First, Table 2a contains a negative entry (see the value for entry $[5, 9]$). Second, the distributions for 'age' are inconsistent in Tables 2a and 2b. For example, the sum of 'higrade' 01 and 02 under 'age' $[95, 99]$ in Table 2b does not equal to the entry for 'age' $[95, 99]$ in Table 2a. Both issues make it impossible to sample directly from Tables 2a and 2b. To deal with the first issue, we round the negative values to zeros and then normalize the marginal distributions based on the rounded values, resulting in Tables 3a and 3b. From these processed tables, it is still impossible to generate tuples directly. For example, conditioned on the 'age' attribute being $[95.99]$ for a tuple, there is no way to generate the 'higrade' attribute, since both choices are of zero probability in Table 3b. To deal with this, we enforce consistency (explained in Section 3.2 in detail) on attribute 'age' for Tables 3a and 3b. We now present the finalized Tables 4a and 4b. From these two tables, we can use the standard approach discussed as above to generate attributes 'age' and 'higrade' for a tuple. For example, the 'age' attribute has probability 0.067 to be $[95, 99]$, and conditioned on this, the 'higrade' attribute has an equal chance to be either 01 or 02.

2.4. **Privacy Analysis.** Based on the result of Balle and Wang (2018), we present Lemma 2.1 below for the composition of Gaussian mechanisms.

Lemma 2.1. *For m queries $F_1, F_2, \ldots, F_m$ with ℓ_2-sensitivity $\Delta_1, \Delta_2, \ldots, \Delta_m$ with independent additive Gaussian noise of standard deviation σ_i. Then the composition of the m noisy answers satisfies (ϵ, δ)-differential privacy if*

$$\sqrt{\sum_{i=1}^{m} \frac{\Delta_i^2}{\sigma_i^2}} \leqslant \sqrt{2}\left(\sqrt{\chi^2+\epsilon}-\chi\right),$$

with:

$$\operatorname{erfc}(\chi) - e^{\epsilon} \operatorname{erfc}(\sqrt{\chi^2+\epsilon}) = 2\delta.$$

The proof is deferred to Appendix A.2. Based on Lemma 2.1, we now discuss how to set the Gaussian noise parameters in different phases of PrivBayes to ensure that the whole algorithm of PrivBayes satisfies (ϵ, δ)-differential privacy. Recall in PrivBayes, only the network learning and distribution learning phases require direct access to the input database. No access to the original database is invoked during the tuple generation phase. Hence, it suffices to quantify the privacy cost of the first two phases only.

In the network learning phase, there are $m_1 = \binom{d}{2}$ queries $F_1, F_2, \ldots, F_{m_1}$ for computing the scores for all $\binom{d}{2}$ attribute pairs. For each $i \in \{1, \ldots, m_1\}$, the ℓ_2-sensitivity of F_i is $\Delta_i = 2$, and the query result of F_i is added with independent Gaussian noise of standard deviation $\sigma_i = \sigma_{NL}$. In addition, there is a query F_{m_1+1} with ℓ_2-sensitivity $\Delta_{m_1+1} = 1$ to generate an approximate version n of the size of the input dataset, and we denote the added Gaussian noise amount by σ_N. In the distribution learning phase, there are m_2 queries $F_{m_1+2}, \ldots, F_{m_1+m_2+1}$ for $m_2 = d - k + 1$. For each $i \in \{m_1+2, \ldots, m_2+m_1+1\}$, the ℓ_2-sensitivity of F_i is $\Delta_i = 1$, and the query result of F_i is added with independent Gaussian noise of standard deviation $\sigma_i = \sigma_{NC}$.

From Lemma 2.1, to guarantee that the whole algorithm of PrivBayes satisfies (ϵ, δ)-differential privacy, it suffices to set $\sum_{i=1}^{m} \frac{{\Delta_i}^2}{{\sigma_i}^2} = f(\epsilon, \delta) \triangleq 2\left(\sqrt{\chi^2+\epsilon}-\chi\right)^2$, where χ is computed as in Lemma 2.1. In general, we can choose σ_{NL} such that $\frac{4m_1}{{\sigma_{NL}}^2} = \beta_1 \cdot f(\epsilon, \delta)$ for some ratio $\beta_1 \in (0,1)$. Afterwards, we choose σ_N such that $\frac{1}{{\sigma_N}^2} = \beta_2 \cdot f(\epsilon, \delta)$ for some ratio $\beta_2 \in (0,1)$. Finally, we select σ_{NC} such that $\frac{m_2}{{\sigma_{NC}}^2} = (1 - \beta_1 - \beta_2) \cdot f(\epsilon, \delta)$. The ratios β_1 and β_2 are customizable parameters used to balance the quality of network learning and distribution learning phases in PrivBayes. In our implementation, we set $\beta_1 + \beta_2$ to 0.2. For simplicity, we also evenly divide the privacy budget for score computation and approximating n among all attribute pairs and n^*; and evenly divide the privacy budget for distribution learning among all low dimension distributions.

Choice of τ. The construction of AP pairs can affect the informativeness of a Bayesian network and the robustness of marginal distributions. An AP pair that contains a large number of attributes tends to capture more information from D, but its corresponding marginal distribution is more vulnerable to noisy injection since the cells in the marginal distribution table tend to have smaller values. Motivated by this, we impose an upper bound τ on the number of cells (domain size) allowed in a marginal distribution.

Let (A, Π) be an AP pair and c be the number of cells (size of domain) in the marginal distribution table. The average count in each cell of the table is n/c. On the other hand, the standard deviation of the Gaussian noise added in each cell is σ_{NC}. Intuitively, n/c should be considerably larger than σ_{NC}; otherwise, the signal in the table would be drowned by the noise. Motivated by this, we require that any AP pair should correspond to a marginal distribution table with no more than $\tau = \frac{n^*}{c \cdot (4\tilde{\sigma}_{NC})}$ cells, where n^* is the noisy version of n, and $\tilde{\sigma}_{NC}$ is the scale of Gaussian noise to be used in the distribution learning under the pessimistic assumption that $k = 1$.

3. Optimizations

This section presents three optimizations we adopt in PrivBayes. Section 3.1 discusses additional treatments that we apply on attributes with large domains, while Section 3.2 explains how we post-process the noisy marginal distributions to enforce consistency. Section 3.3 presents an improved approach to generate synthetic data using PrivBayes. We note that none of these optimizations consume extra privacy budget, and hence, they do not affect our privacy analysis in Section 2.4.

3.1. **Special Treatments for Attributes with Large Domains.** Recall that our algorithm requires that each AP pair in the Bayesian network $\mathcal{N}$ should correspond to a marginal distribution with at most a domain size of τ (τ cells). Given this constraint, attributes with large domains are not likely to be included in an AP pair, since the inclusion of such attributes would significantly increase the domain size. However, the omission of large-domain attributes in the AP pair could degrade the quality of the Bayesian network, especially when those attributes have strong correlations with other attributes.

To address this issue, we propose to (i) generate "coarsened" versions of large-domain attributes, such that the coarsened attributes have smaller domains, and (ii) add such coarsened attributes into the input data, so that they could be included in the AP pairs to improve the quality of the Bayesian network. In particular, for any attribute A_i with $|\operatorname{dom}(A_i)|$ larger than a threshold θ, we evenly divide the values in $\operatorname{dom}(A_i)$ into $\frac{|\operatorname{dom}(A_i)|}{g}$ groups (for illustration purpose we let $|\operatorname{dom}(A_i)|$ be a power of g), such that all groups has g values, where g is a customizable parameter. We then generate a coarsened version A_i' of A_i with a domain of size $\frac{|\operatorname{dom}(A_i)|}{g}$, such that the i-th value in the domain of A_i' corresponds to the i-th group. In case that $|\operatorname{dom}(A_i')|$ is still larger than our threshold, we would further coarsen A_i' into another attribute A_i'', using the same approach. In our implementation, we set $\theta = g^2$. We refer to $A_i, A_i', A_i'', \ldots$ as a *hierarchy* on A_i. Note that the generation of $A_i', A_i'', \ldots$ only depends on the parameter g and the public information of A_i, and does not utilize any private information in the input dataset D. After the high-level attributes are generated, we add them into the dataset D to obtain an augmented dataset D', and then apply our algorithm on D' to obtain the Bayesian network $\mathcal{N}$ and the noisy low dimension distributions $\mathcal{P}^*$. After enforcing consistency on these noisy distributions (to be explained in Section 3.2), we discard all the high-level attributes to obtain the final synthetic dataset. Discarding high-level attributes does not affect our privacy analysis.

3.2. **Enforcing Consistency among Noisy Marginals.** Previous work (Hay et al., 2010) shows that, by enforcing consistency on noisy query results, we can significantly improve the accuracy of a general class of histogram queries. It also applies to our problem. In particular, after we generate the noisy marginals in the distribution learning phase of our algorithm, we post-process the noisy marginals to enforce consistency and improve accuracy.

Note that such post-processing does not consume any privacy budget, since it only utilizes the noisy marginals and does not use any other information from the input dataset D.

The input to our post-processing step is a set of noisy marginal distributions, each corresponding to a set of attributes. The output is the modified versions of these distributions. The high-level idea is to sequentially compute the best approximations for these distributions that are consistent on a sequence of sets of attributes. Through this sequence of computations, latter steps will not invalidate the consistency established in former steps. This property is crucial to enforcing consistency, since it prevents the algorithm for it from running indefinitely.

3.2.1. *Consistency among Marginals on a Set of Attributes.* We first describe the method of ensuring the consistency among distributions on a set of attributes. Consider, without loss of generality, marginals $T_{V_1}, \ldots, T_{V_h}$ consisting of sets of attributes $V_1, \ldots, V_h$ respectively. Let $\mathcal{C} = V_1 \cap \ldots \cap V_h$, be the common set of attributes in these distributions. We want to ensure that for any $c \in \mathrm{dom}(\mathcal{C})$, $T_{V_i}(c) = T_{V_j}(c)$ for any $i, j \in [h]$. Namely, the distributions should agree on the marginal distribution on the domain of $\mathcal{C}$. For example, Tables 3a and 3b do not agree on 'age' (see entry $[95, 99]$) while Tables 4a and 4b are consistent.

Here, we adopt the techniques proposed in Hay et al. (2010) to achieve consistency in two steps. The first step computes the best approximation for $T_{\mathcal{C}}(c)$ for each entry $c \in \mathrm{dom}(\mathcal{C})$. We use σ_i^2 to denote the variance for each entry in the domain of $\mathcal{C}$ obtained from the noisy distribution table T_{V_i}. Then, the best approximation of $T_{\mathcal{C}}(c)$ that minimizes the overall variance is the weighted arithmetic mean from all noisy distribution tables, i.e.:

$$T_{\mathcal{C}}(c) = \frac{\sum_{i=1}^{h}(w_i \cdot T_{V_i}(c))}{\sum_{i=1}^{h} w_i}, \tag{3.1}$$

where $\frac{w_i}{w_j} = \frac{\sigma_j^2}{\sigma_i^2}$. For example, let the variance in each cell of Tables 3a and 3b be the same, then for each value in the domain of 'age' attribute, the variance in Table 3a is half as that in Table 3b. This means that the weights for Tables 3a and 3b are 0.67 and 0.33, respectively.

The second step is to update all entries according to the approximation computed as above. For each value $c \in \mathrm{dom}(\mathcal{C})$, along the projection of the table T_{V_i} on $\mathcal{C}$ do

$$T_{V_i}(c) \leftarrow T_{V_i}(c) + \frac{T_{\mathcal{C}}(c) - T_{V_i}(c)}{|\,\mathrm{dom}(V_i \backslash \mathcal{C})|}. \tag{3.2}$$

For example, the weighted average for 'age' on the entry $[95, 99]$ for Tables 4a and 4b is 0.067. Accordingly, the increase of 0.067 for 'age' on the entry $[95, 99]$ is split equally to the values of 'age'$= [95, 99]$, code=01 and 'age'$= [95, 99]$, code=01 in Table 4b. In the mean time, we update the variance in each cell accordingly

$$\sigma_i^2 \leftarrow \left(1 - \frac{w_i}{q_i}\right)^2 \sigma_i^2 + \left(\frac{w_i}{q_i}\right)^2 (q_i - 1)\sigma_i^2 + \frac{1}{q_i^2} \sum_{i=2}^{m} (w_i^2 \cdot \sigma_i^2), \tag{3.3}$$

where we use q_i as $|\,\mathrm{dom}(V_i \backslash \mathcal{C})|$.

3.2.2. *Consistency within an Attribute Hierarchy.* We now describe the approach for enforcing consistency on hierarchical attributes. Consider $A_i', A_i'', A_i''', \dots$ as a hierarchy on A_i, where a larger number of superscripts indicates that the attribute is more coarsened. We refer to A_i as the lowest-level attribute in the hierarchy and $A_i', A_i'', A_i''', \dots$ as the high-level attributes. Note that different noisy distribution tables obtained from the distribution learning phase can contain the same attribute in the hierarchy of A_i, but each table can contain at most one attribute in the hierarchy. This is ensured by the rule of network construction. The overall mechanism for enforcing hierarchical consistency is as follows:

- For each distribution table with a less generalized attribute, we extend the distribution table with its corresponding more generalized attributes, by padding extra dimensions. For example, for a distribution table with attribute A_i''', we extend the distribution table to all coarsened attributes of A_i but A_i'', A_i' and A_i itself.
- We sequentially enforce consistency across different distribution tables from the most generalized to the least generalized hierarchical attribute, following the mechanism described in the previous subsection.
- For each distribution table containing multiple attributes within the hierarchy (e.g., A_i and A_i'), we enforce the within-table consistency in a top-down manner: we fix the higher-level and adjust the lower-level iteratively. At last, we discard all of the higher-level information from the table while keeping only the attribute of the lowest level.

In the second step, enforcing consistency on a less generalized attribute does not invalidate the consistency established from all previous steps on more generalized attributes. After the second step, the consistency across tables are assured. But there is still an issue worth mentioning: the consistency within a hierarchy in each table. We next adjust the distribution of a less generalized attribute based on the distribution of the corresponding more generalized attribute. The update rule is similar to that of mutual consistency among tables (see Eq. (3.1), (3.2) and (3.3)). We apply this update rule repeatedly from the most generalized attribute to the least generalized. At last we discard all higher level attributes. It is obvious that enforcing consistency within a table does not invalidate the consistency established across tables previously, and we omit the proof here.

3.2.3. *Overall Consistency.* Finally we describe the overall mechanism to ensure that all noisy distribution tables are consistent. The challenge here is to determine the order of enforcing consistency such that the overall consistency is achieved in a finite number of steps. For the example network $\mathcal{N}_1$ (see Figure 1), we should first enforce consistency on 'age' for two tables consisting of attribute 'age' ($P^*(age)$) and attributes 'age' and 'higrade' ($P^*(higrade \mid age)$) and then enforce consistency on 'higrade' on the two tables consisting of 'higrade' and 'age' ($P^*(higrade \mid age)$), and 'income' and 'higrade' ($P^*(income \mid higrade)$). For more general cases, we adopt the technique proposed in Qardaji et al. (2014) with some minor adjustment, due to hierarchical attributes. We first need to introduce the definition of order:

Definition 3.1 (Order of Sets)**.** Consider a set of containing all attributes and their coarsened versions, $\mathbb{A} \triangleq \{A_1, A_1', \dots, A_2, \dots, A_d\}$ and all subsets of $\mathbb{A}$. We define the following partial order: $S_i < S_j$ if $S_i \subset S_j$ for $S_i, S_j \subset \mathbb{A}$.

Now we can apply the following procedure adapted from (Qardaji et al., 2014) to ensure the overall consistency. At first, we enforce consistency on $\varnothing$, i.e., we make the sum of entries

Algorithm 5: Enforce consistency among marginals

```
Data: Noisy tables $T_{S_i}$, $i = 1, \ldots$
Result: Consistent noisy tables $T_{S_i}$, $i = 1, \ldots$
$\mathcal{B} \leftarrow \varnothing$ ; foreach pair of tables $T_{S_i}, T_{S_j}$ do
    $\mathcal{B} \leftarrow \mathcal{B} \cup \{S_j \cap S_j\}$;
end
Topological sort on $\mathcal{B}$;
foreach $B \in \mathcal{B}$ following the topological sorting (in ascending order) do
    enforce consistency on $B$ across all tables $T_{S_i}$'s where $B \subset S_i$
end
return $\{T_{S_i}\}$;
```

Table 5. Histograms for tuple generation

(a) Histogram for 'age'

'age'	**count**
$[0, 4]$	500
$[5, 9]$	0
...	...
$[95, 99]$	670

(b) Histogram for 'age' and 'higrade'

'higrade' / **'age'**	01	02
$[0, 4]$	250	250
$[5, 9]$	0	0
...	...	...
$[95, 99]$	335	335

Table 6. Histograms for tuple generation after generating one tuple.

(a) Histogram for 'age'

'age'	**count**
$[0, 4]$	500
$[5, 9]$	0
...	...
$[95, 99]$	669$\downarrow$

(b) Histogram for 'age' and 'higrade'

'higrade' / **'age'**	01	02
$[0, 4]$	250	250
$[5, 9]$	0	0
...	...	...
$[95, 99]$	334$\downarrow$	335

in each table identical. Then we extend each table that contains a less generalized attribute to include all its more generalized attributes. For example, we include attributes $A_1', A_1'', \ldots$, for all tables containing attribute A_1. Then we compute the intersection of attributes for all pairs of tables; these intersections of sets form a partial ordering defined as above. We then apply a topological sorting on these sets and enforce consistency on each intersection following the ascending partial order. By definition, a latter step will not invalidate consistency established in former steps; and following any topological ordering will result in the same consistent tables. Algorithm 5 shows the pseudo-code for enforcing consistency across tables. At last, for each table that contains hierarchical attributes, we enforce within-table consistency and discard all but the least generalized hierarchical attributes. We remark that these extra pre and post processing steps for hierarchical attributes do not invalidate the correctness of the original algorithm. The proof for the original algorithm can be found in Qardaji et al. (2014).

3.3. **Tuple Generation.** Recall from Section 2.3 that we generate synthetic data by sampling each tuple independently based on Eq. (2.3). The shortcoming of this "sampling with replacement" strategy is that we may never generate a tuple which has a low but non-zero frequency in the original input dataset. This motivates us to consider an alternate approach for tuple generation, namely, sampling without replacement. To explain, suppose that we are given the Bayesian network structure in Figure 1 and the noisy marginal tables in Tables 4a and 4b. We first transform the distributions of Tables 3a and 3b to histograms by multiplying the probabilities by N (say $N = 10,000$), the number of tuples to generate, resulting in Tables 5a and 5b. Now, we generate one tuple at a time. The first tuple is generated by sampling from Table 5a and then Table 5b. Then, after the generation of a tuple we update Tables 5a and 5b by subtracting 1 from the entries corresponding to the generated tuple in both Tables 5a and 5b. For example, after generating a tuple of 'age' $[95, 99]$ and 'higrade' 01, we obatin updated Tables 6a and 6b. We repeat the same process until all tuples are generated. During our experiments, we found that our strategy of sampling without replacement performs significantly better than sampling with replacement for the regression task (defined in Section 4).

3.3.1. *Tuple generation from inconsistent noisy marginal distributions.* One may want to follow the standard approach described in Section 2.3 to generate a tuple from inconsistent noisy marginal distributions directly without enforcing consistency. However, it may happen during the generation of one tuple that all values for an attribute that we want to sample from form an empty histogram (recall the example noisy distributions corresponding to Tables 3a and 3b in Section 2.3). As we have discussed, the reason is that the noisy distributions are very unlikely be consistent after rounding and normalizing, neither do the noisy histograms deduced from the noisy distributions. In this case, we propose a heuristic approach. In particular, we fall back to the previous step in the generation of the tuple, i.e., the previous histogram corresponding to the previous AP pair. From there, we repeat the same sampling process in the hope of sampling a different value for the attribute. If we end up with empty histograms repeatedly (say we set the threshold to 10), we go back 1 step further and repeat the process until the tuple is generated.

4. Experiments

This section empirically evaluates PrivBayes using the data and performance metrics from the 2018 NIST Challenge. The input dataset is the Colorado Public Use Microdata Sample (PUMS) data, which consists of $662,000$ records of 98 categorical and numerical variables. The performance metrics are as follows[1]:

- Clustering: For this metric, we compare the 3-way marginal distributions between the original and synthetic datasets on randomly selected marginals. We first calculate the absolute difference of the low dimension distributions for the original and synthetic datasets, which belongs to the range between 0 (perfect match) and 2 (no overlap at all). The resulting score is defined as 10^6 times the result of 1 minus half the absolute difference, averaged over 100 repeats.

[1]The detailed implementation of the metrics can be found in `https://apps.topcoder.com/forums/?module=Thread&threadID=933618&start=0`.

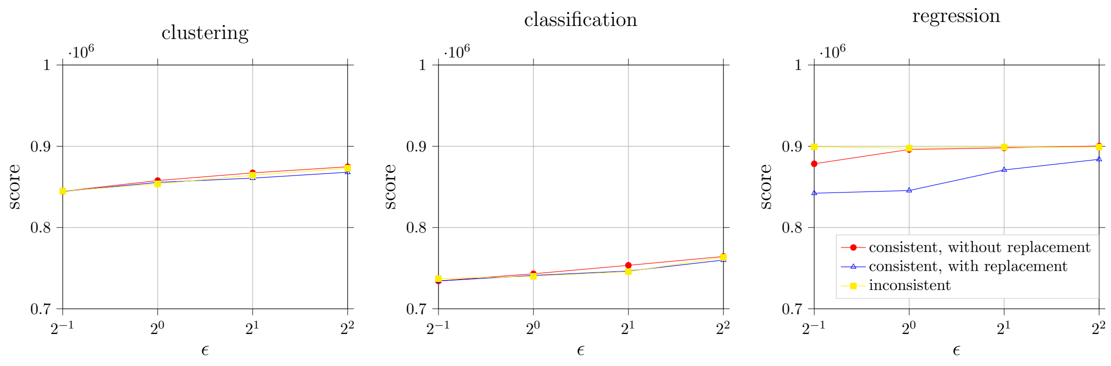

Figure 2. The performance of PrivBayes measured by the three metrics.

- Classification: For this metric, we first randomly pick one third of the attributes. If a particular attribute is categorical, we randomly select a subset of the possible values; otherwise, we select a random range of values. We then identify the frequencies of tuples in the original dataset and in the synthetic dataset that match the selected value and apply natural logarithm on the frequencies. The classification score is computed as the root mean squared difference on the log-frequencies between the original and synthetic datasets over 300 repeats, divided by $\ln(1/1000)$ and then times 10^6.
- Regression: This metric is the average of two scores. The first score is the the mean squared error between the Gini coefficients (Gini, 1936) (measured on the joint distribution of gender and income) of the original and synthetic datasets for all cities. For the second score, we rank the cities by the gender pay gap for both datasets, and calculate the score component based on the mean-square deviation between the resulting city ranks. The final score is the average of these two scores normalized to the range $[0, 10^6]$.

We evaluate the performance of PrivBayes when ϵ varies in $\{0.5, 1, 2, 4\}$ and $\delta = 10^{-9}$. Figure 2 illustrates the performance of PrivBayes measured by the three evaluation metrics described as above. We consider three variants of PrivBayes: sampling with replacement from consistent noisy distributions, sampling without replacement from consistent noisy distributions, and sampling from inconsistent noisy distributions. Overall, PrivBayes yields better data utility when given with more privacy budget (i.e., a larger ϵ). Among the three metrics, the clustering score is more aligned with the original optimization goal of PrivBayes in Zhang et al. (2017), namely, approximating low dimension marginal distributions of the input dataset. As a result, PrivBayes performs well on clustering. Although we did not optimize PrivBayes for the task of regression or classification (defined as above), the regression score of PrivBayes is much better than its classification score. This may be because classification is a more difficult task to perform than regression. In addition, there is a negligible difference between sampling from inconsistent distributions and consistent distributions, for clustering and classification. For regression, the former even outperforms the latter under strong privacy guarantees (small ϵ). This surprising result indicates the effectiveness of the heuristic approach that we adopted for sampling from inconsistent noisy marginal distributions (see Section 3.3.1).

5. Summary and Future Work

This paper details the implementation of PrivBayes that we used to participate in the 2018 NIST Differential Privacy Synthetic Data Challenge, and presents experimental results based on the data and performance metrics from the challenge. After the challenge was concluded, we have looked into methods that can address two limitations of PrivBayes that we observed. First, as we discuss in Sections 2.3 and 3.2, inconsistencies in low dimension noisy distributions forbid us from generating tuples directly. In addition to enforcing consistency, we find that this issue could also be mitigated using the graphical-model-based algorithm in McKenna et al. (2019), as it circumvents the issue of inconsistency. Second, the Bayesian network constructed by PrivBayes has at most d low dimension distributions, where d is the number of attributes in the input dataset. This restricts the modelling capability of PrivBayes, which in turn affects the utility of the synthetic data it generates. To address this issue, we are developing a new algorithm that can construct more effective graphical models for synthetic data generation.

Acknowledgment

The authors wish to thank the reviewers for their thoughtful and constructive comments. This work was supported by the the Ministry of Education Singapore (Number MOE2018-T2-2-091), and by the National Research Foundation, Singapore under its Strategic Capability Research Centres Funding Initiative. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not reflect the views of the funding agencies.

References

B. Balle and Y. Wang. Improving the gaussian mechanism for differential privacy: Analytical calibration and optimal denoising. In *Proceedings of the 35th International Conference on Machine Learning, ICML*, pages 403–412, 2018. http://proceedings.mlr.press/v80/balle18a.html.

T. M. Cover and J. A. Thomas. *Elements of Information Theory*. Wiley, 2001. https://doi.org/10.1002/0471200611.

C. Dwork. Differential privacy. In *ICALP*, pages 1–12, 2006. https://doi.org/10.1007/11787006_1.

C. Dwork and A. Roth. The algorithmic foundations of differential privacy. *Theoretical Computer Science*, 9(3-4):211–407, 2013. https://doi.org/10.1561/0400000042.

C. Dwork and G. N. Rothblum. Concentrated differential privacy. *CoRR*, abs/1603.01887, 2016. http://arxiv.org/abs/1603.01887.

C. Dwork, F. McSherry, K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. In *Theory of Cryptography Conference*, pages 265–284, 2006. https://doi.org/10.1007/11681878_14.

C. Gini. *On the Measure of Concentration with Special Reference to Income and Statistics*. Colorado College Publication, 1936.

M. Hay, V. Rastogi, G. Miklau, and D. Suciu. Boosting the accuracy of differentially private histograms through consistency. *Proceedings of the VLDB Endowment*, 3(1-2):1021–1032, 2010. https://doi.org/10.14778/1920841.1920970.

D. Koller and N. Friedman. *Probabilistic Graphical Models - Principles and Techniques*. MIT Press, 2009. https://doi.org/10.1017/S0269888910000275.

E. L. Lawler. Fast approximation algorithms for knapsack problems. *Mathematics of Operations Research*, 4(4):339–356, 1979. ISSN 0364765X, 15265471. http://www.jstor.org/stable/3689221.

R. McKenna, D. Sheldon, and G. Miklau. Graphical-model based estimation and inference for differential privacy. In *Proceedings of the 36th International Conference on Machine Learning*, pages 4435–4444, 2019. http://proceedings.mlr.press/v97/mckenna19a.html.

National Institute of Standards and Technology. Differential privacy synthetic data challenge. https://www.nist.gov/ctl/pscr/open-innovation-prize-challenges/past-prize-challenges/2018-differential-privacy-synthetic, 2018.

W. Qardaji, W. Yang, and N. Li. PriView: Practical differentially private release of marginal contingency tables. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 1435–1446, 2014. https://doi.org/10.1145/2588555.2588575.

J. Zhang, G. Cormode, C. M. Procopiuc, D. Srivastava, and X. Xiao. PrivBayes: Private data release via Bayesian networks. *ACM Trans. Database Syst.*, 42(4):25:1–25:41, 2017. https://doi.org/10.1145/3134428.

Z. Zhang, T. Wang, N. Li, J. Honorio, M. Backes, S. He, J. Chen, and Y. Zhang. PrivSyn: Differentially private data synthesis. In *Usenix Security Symposium*, 2021. http://www.usenix.org/conference/usenixsecurity21/presentation/zhang-zhikun.

J. Zhao and D. Zhang. Code for: PrivBayes - Private Data Release via Bayesian Networks, Dec. 2021.

APPENDIX

A.1. **Proof of Lemma 1.2.**

Proof. We start with the definition of the privacy loss random variable (Dwork and Roth, 2013). Let L be the privacy loss random variable defined on two neighboring datasets D_1 and D_2 and a randomized algorithm $\mathcal{M}$, then:

$$L_{\mathcal{M},D_1,D_2}(y) \triangleq \log\left(\frac{\Pr[\mathcal{M}(D_1)=y]}{\Pr[\mathcal{M}(D_2)=y]}\right). \tag{A.1}$$

The randomness in $L_{\mathcal{M},D_1,D_2}$ is over the coin flips by the random algorithm $\mathcal{M}$ on input dataset D_1. By Theorem 5 in Balle and Wang (2018), a mechanism $\mathcal{M}$ is (ϵ,δ)-DP if and only if the following holds for every neighboring datasets D_1 and D_2:

$$\Pr[L_{\mathcal{M},D_1,D_2} \geqslant \epsilon] - e^{\epsilon}\Pr[L_{\mathcal{M},D_2,D_1} \leqslant -\epsilon] \leqslant \delta. \tag{A.2}$$

To convert a function $F : \mathcal{D} \to \mathbb{R}^d$ to a differentially private function, we inject a Gaussian noise sampled from $\mathcal{N}(\mathbf{0},\sigma^2\cdot\mathbf{I})$ to the outcome of F. Namely, $\mathcal{M}(D) = F(D) + Z$, where $Z \sim \mathcal{N}(\mathbf{0},\sigma^2\cdot\mathbf{I})$. We denote $\eta = \frac{S^2(F)}{2\sigma^2}$. Then the privacy loss random variable of the Gaussian mechanism also follows a Gaussian distribution $\mathcal{N}(\eta,2\eta)$ (Dwork and Rothblum, 2016). We proceed on the left hand side of Eq. (A.2):

$$\begin{aligned}\Pr[L_{\mathcal{M},D_1,D_2} \geqslant \epsilon] - e^{\epsilon}\Pr[L_{\mathcal{M},D_2,D_1} \leqslant -\epsilon] &= \Pr[\mathcal{N}(\eta,2\eta) \geqslant \epsilon] - e^{\epsilon}\Pr[\mathcal{N}(\eta,2\eta) \leqslant -\epsilon]\\ &= \Pr[\mathcal{N}(\eta,2\eta) \geqslant \epsilon] - e^{\epsilon}\Pr[\mathcal{N}(\eta,2\eta) \geqslant 2\eta+\epsilon]\\ &= \Pr[\mathcal{N}(0,1) \geqslant \frac{\epsilon-\eta}{\sqrt{2\eta}}] - e^{\epsilon}\Pr[\mathcal{N}(0,1) \geqslant \frac{\epsilon+\eta}{\sqrt{2\eta}}]\\ &= \frac{1}{2}\left(\mathrm{erfc}(\frac{\epsilon-\eta}{2\sqrt{\eta}}) - e^{\epsilon}\mathrm{erfc}(\frac{\epsilon+\eta}{2\sqrt{\eta}})\right)\end{aligned}$$

Let $\chi = \frac{\epsilon-\eta}{2\sqrt{\eta}}$, we have:

$$\sqrt{\chi^2+\epsilon} - \chi = \sqrt{\eta}.$$

Then Eq. (A.2) holds if and only if

$$\mathrm{erfc}(\chi) - e^{\epsilon}\mathrm{erfc}(\sqrt{\chi^2+\epsilon}) \leqslant 2\delta,$$

where

$$\mathrm{erfc}(x) \triangleq 1 - \frac{2}{\sqrt{\pi}}\int_0^x e^{-t^2}\,dt.$$

□

A.2. **Proof of Lemma 2.1.**

Proof. We consider m queries $Q_1, Q_2, \dots, Q_m$ with ℓ_2-sensitivity $\Delta_1, \Delta_2, \dots, \Delta_m$. The query result of Q_i on a input database is added with independent Gaussian noise of standard deviation σ_i.

The proof can be regarded as a simple extension from the proof of Lemma 1.2. We first recall our proof for a general Q, where we add Gaussian noise of standard deviation σ to the result. The way we show that injecting additive Gaussian satisfies (ϵ, δ)-DP is to show that Eq. (A.2) holds for every neighboring dataset D_1 and D_2. Now we extend this result to a concatenated query $\mathbf{Q} = (Q_1, \dots, Q_m)$. Suppose the mechanism $\mathbf{M}$ inject Gaussian noises with zero mean variance $\sigma_1^2, \dots, \sigma_m^2$ to each dimension of $\mathbf{Q}$, then by Theorem 5 in Balle and Wang (2018), it suffices to show that for every neighboring datasets D_1 and D_2:

$$\Pr[L_{\mathbf{M},D_1,D_2} \geqslant \epsilon] - e^{\epsilon} \Pr[L_{\mathbf{M},D_2,D_1} \leqslant -\epsilon] \leqslant \delta, \tag{A.3}$$

where

$$L_{\mathbf{M},D_1,D_2}(y) \triangleq \log\left(\frac{\Pr[\mathbf{M}(D_1) = y]}{\Pr[\mathbf{M}(D_2) = y]}\right). \tag{A.4}$$

To proceed, we use $\mathcal{M}_i$ to denote the mechanism that injects Gaussian noise of standard deviation σ_i to query Q_i. Then we can write $\mathbf{M} = (\mathcal{M}_1, \dots, \mathcal{M}_m)$ and $y = (y_1, \dots, y_m)$. Accordingly, the privacy loss of the composed mechanism $\mathbf{M}$ can be written as:

$$L_{\mathbf{M},D_1,D_2}(y) = \sum_{i=1}^{m} \log\left(\frac{\Pr[\mathcal{M}_i(D_1) = y_i]}{\Pr[\mathcal{M}_i(D_2) = y_i]}\right). \tag{A.5}$$

Recall each random variable $\log\left(\frac{\Pr[\mathcal{M}_i(D_1)=y_i]}{\Pr[\mathcal{M}_i(D_2)=y_i]}\right)$ follows a Gaussian distribution of mean η_i and variance $2\eta_i$, where $\eta_i = \frac{\Delta_i^2}{2\sigma_i^2}$ for $i = 1, \dots, m$ (Dwork and Rothblum, 2016). Since each $\mathcal{M}_i$ samples the Gaussian noise independently, random variable $L_{\mathbf{M},D_1,D_2}$ follows a Gaussian distribution $L_{\mathbf{M},D_1,D_2} \sim \mathcal{N}(\sum_{i=1}^{m} \eta_i, 2\sum_{i=1}^{m} \eta_i)$. Similar to the proof for Lemma 1.2, plugging the distribution for $L_{\mathbf{M},D_1,D_2}$ into Eq. (A.3) completes the proof. □