Data Synthesis via Differentially Private Markov Random Fields

Kuntai Cai
National University of Singapore
caikt@comp.nus.edu.sg

Xiaoyu Lei
University of Connecticut
xiaoyu.lei@uconn.edu

Jianxin Wei
National University of Singapore
jianxinwei@u.nus.edu

Xiaokui Xiao
National University of Singapore
xkxiao@nus.edu.sg

ABSTRACT

This paper studies the synthesis of high-dimensional datasets with differential privacy (DP). The state-of-the-art solution addresses this problem by first generating a set $\mathcal{M}$ of noisy low-dimensional marginals of the input data D, and then use them to approximate the data distribution in D for synthetic data generation. However, it imposes several constraints on $\mathcal{M}$ that considerably limits the choices of marginals. This makes it difficult to capture all important correlations among attributes, which in turn degrades the quality of the resulting synthetic data.

To address the above deficiency, we propose PrivMRF, a method that (i) also utilizes a set $\mathcal{M}$ of low-dimensional marginals for synthesizing high-dimensional data with DP, but (ii) provides a high degree of flexibility in the choices of marginals. The key idea of PrivMRF is to select an appropriate $\mathcal{M}$ to construct a *Markov random field (MRF)* that models the correlations among the attributes in the input data, and then use the MRF for data synthesis. Experimental results on four benchmark datasets show that PrivMRF consistently outperforms the state of the art in terms of the accuracy of counting queries and classification tasks conducted on the synthetic data generated.

PVLDB Reference Format:
Kuntai Cai, Xiaoyu Lei, Jianxin Wei, and Xiaokui Xiao. Data Synthesis via Differentially Private Markov Random Fields. PVLDB, 14(11): 2190 - 2202, 2021.
doi:10.14778/3476249.3476272

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/caicre/PrivMRF.

1 INTRODUCTION

Releasing relational data while preserving privacy is an important problem that has attracted extensive research interests over the past decades. A canonical solution to this problem is *synthetic data generation* [11], which releases a synthesized version of the data containing carefully calibrated noise for privacy protection. For example, consider the dataset D in Table 1, where each attribute has a binary domain $\{\alpha, \beta\}$. Suppose that we are to generate a synthetic version of D. Then, a straightforward approach is to first convert D into the *contingency table* T_D in Table 2, where each row (i) corresponds to a possible tuple x in the attribute domain of D and (ii) counts the number of x's occurrences in D. After that, we add noise to the count in each row of T_D to obtain the noisy contingency table $\tilde{T}_D$ in Table 3. Finally, we round each noisy count in $\tilde{T}_D$ to the nearest non-negative integer, and then transform the $\tilde{T}_D$ to the synthetic dataset $\tilde{D}$ in Table 4. When the noise in $\tilde{T}_D$ following an appropriate distribution, we can show that $\tilde{D}$ ensures *differential privacy (DP)* [10], which is a rigorous notion of privacy that has been widely adopted in both academia and industry [8, 12, 31].

Table 1: Dataset D.

A_1	A_2	A_3
α	α	β
α	α	β
α	α	β
α	β	α
β	α	α
β	α	α
β	α	α
β	α	α
β	β	α

Table 2: Contingency table T_D.

A_1	A_2	A_3	Count
α	α	α	0
α	α	β	3
α	β	α	1
α	β	β	0
β	α	α	4
β	α	β	0
β	β	α	1
β	β	β	0

Table 3: Noisy table $\tilde{T}_D$.

A_1	A_2	A_3	Count
α	α	α	−0.2
α	α	β	2.9
α	β	α	−0.3
α	β	β	2.2
β	α	α	1.7
β	α	β	0.2
β	β	α	0.8
β	β	β	−0.1

Table 4: Synthetic data $\tilde{D}$.

A_1	A_2	A_3
α	α	β
α	α	β
α	α	β
α	β	β
α	β	β
β	α	α
β	α	α
β	β	α

Table 5: Marginal T_{D,M_1}.

A_1	A_2	Count
α	α	3
α	β	1
β	α	4
β	β	1

Table 6: Marginal T_{D,M_2}.

A_1	A_3	Count
α	α	1
α	β	3
β	α	5
β	β	0

The aforementioned approach, however, suffers from the curse of dimensionality. To explain, observe that the number m of rows in the contingency table T_D is exponential to the number d of attributes in the input data D. Therefore, m is prohibitively large when d is large. In that case, it is impractical to materialize T_D or its noisy version $\tilde{T}_D$, due to the excessive space and time overheads

Proceedings of the VLDB Endowment, Vol. 14, No. 11 ISSN 2150-8097.
doi:10.14778/3476249.3476272

incurred. Furthermore, given a large d, the number of records in D would be significantly smaller than m, which implies that the vast majority of the rows in T_D would have zero counts. Such counts will be dominated by noise after we inject noise into T_D, and hence, $\tilde{T}_D$ would be excessively noisy. As such, the synthetic data $\tilde{D}$ generated from $\tilde{T}_D$ would have poor data utility.

To circumvent the curse of dimensionality, existing work [37, 38] proposes PrivBayes, which avoids generating the contingency table T_D and instead utilizes low-dimensional *marginals* of T_D (*i.e.*, projections of T_D on subsets of attributes). For instance, given the dataset D in Table 1, we may generate two marginals T_{D,M_1} and T_{D,M_2} on attribute subsets $M_1 = \{A_1, A_2\}$ and $M_2 = \{A_1, A_3\}$, respectively, as shown in Tables 5 and 6. Intuitively, M_1 (resp. M_2) captures the joint distribution of A_1 and A_2 (resp. A_1 and A_3) in D, which could be used to approximate the data distribution in D as follows:

$$\Pr[A_1, A_2, A_3] \approx \Pr[A_1, A_2] \cdot \Pr[A_3 \mid A_1]. \tag{1}$$

Therefore, we may add noise into M_1 and M_2, and then use them to generate synthetic data based on Eq. (1). The advantage of this approach is that it only deals with low-dimensional marginals, which are more resilient to noise injection and can be materialized efficiently, since they have relatively small numbers of entries.

Motivation and Contributions. The effectiveness of PrivBayes [37, 38] depends on the set $\mathcal{M}$ of low-dimensional marginals that it uses to approximate the data distribution in the input data: if the marginals in $\mathcal{M}$ adequately capture the correlations among attributes in D, then the resulting synthetic data $\tilde{D}$ could be statistically similar to D; otherwise, the distribution of data $\tilde{D}$ may deviate significantly from that of D. As we discuss in Section 2.3, however, PrivBayes imposes some constraints on $\mathcal{M}$ that considerably limit its choices of marginals. In particular, PrivBayes requires that the marginals in $\mathcal{M}$ should correspond to a *Bayesian network* [21], due to which the number of marginals in $\mathcal{M}$ is at most d. This, coupled with the requirement that each marginal should be low-dimensional, makes it difficult for PrivBayes to capture all important correlations among the attributes in D, which in turn degrades the effectiveness of PrivBayes.

To remedy the deficiency of PrivBayes, we propose PrivMRF, a new solution for synthetic data generation with DP. Similar to PrivBayes, PrivMRF also utilizes a set $\mathcal{M}$ of low-dimensional marginals to approximate the distribution of tuples in the input data D. Nevertheless, PrivMRF does not require that the marginals in $\mathcal{M}$ form a Bayesian network; instead, it allows arbitrary marginals in $\mathcal{M}$, as long as it can efficiently derive the distribution of synthetic data from the marginals. This improved flexibility in marginal selection enables PrivMRF to more accurately capture the characteristics of the input data to produce useful synthetic data.

The key idea of PrivMRF is to choose an appropriate marginal set $\mathcal{M}$ to construct a *Markov random field (MRF)* [34] that effectively models the dependencies among the attributes in D, and then use a noisy version of $\mathcal{M}$ to generate synthetic data. Existing methods for MRF construction (*e.g.*, [17, 21, 23, 34, 36, 40]), however, is inapplicable in our scenario, because they either focus on the non-private setting, or are of theoretical interests only. Motivated by this, we devise new algorithms for selecting marginals into $\mathcal{M}$ and constructing an MRF from $\mathcal{M}$, in such a manner that not only satisfies DP but also takes into account the space and time overheads incurred by the whole process of synthetic data generation.

We empirically evaluate PrivMRF against five state-of-the-art methods using four benchmark datasets. Our experimental results show that PrivMRF consistently outperforms the competing methods in terms of the accuracy of counting queries and classification tasks conducted using the synthetic data. Furthermore, PrivMRF is able to process each dataset in at most 17 minutes on a machine with a 2.6GHz CPU and two Nvidia RTX 2080 Ti GPUs.

2 PRELIMINARIES

2.1 Problem Definition

Let D be a dataset with d attributes $\mathcal{A} = \{A_1, A_2, \ldots, A_d\}$ and n tuples $\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(n)}$. For all $A \in \mathcal{A}$ and $j \in [n]$, $\boldsymbol{x}_A^{(j)}$ is the value of attribute A of the j-th tuple, which takes values from a discrete finite space $\mathcal{X}_A$. Accordingly, $\mathcal{X} = \prod_{A \in \mathcal{A}} \mathcal{X}_A$ is the domain of each tuple in D, whose size is exponential in the number of attributes d. For any attribute set $M \subseteq \mathcal{A}$, let $\boldsymbol{x}_M = (x_A)_{A \in M}$ denote the subvector of tuple x restricted to M and $\mathcal{X}_M = \prod_{A \in M} \mathcal{X}_A$ denote the domain of x_M.

Let $T_{D,M}$ denote the vector that counts the number of occurrences of all tuples $\boldsymbol{m} \in \mathcal{X}_M$. That is,

$$T_{D,M}(\boldsymbol{m}) = \sum_{j \in [n]} \mathbb{I}(\boldsymbol{x}_M^{(j)} = \boldsymbol{m}),\ \forall \boldsymbol{m} \in \mathcal{X}_M.$$

Note that we can derive the marginal distribution of M by dividing the value of each entry of $T_{D,M}$ by n. Let $\mathcal{M}$ denote a set of attribute sets. By abuse of notation, we use $T_{D,\mathcal{M}} = (T_{D,M}, M \in \mathcal{M})$ to denote the vector obtained by concatenating $T_{D,M}$ for all $M \in \mathcal{M}$. We refer to $T_{D,\mathcal{M}}$ as the *concatenated marginal distribution vector*.

Our objective is to release a synthetic version $\tilde{D}$ of D using an algorithm that satisfies *(ϵ, δ)-differential privacy (DP)* [10], which is defined based on the notion of *neighboring datasets*, as formalized in the following.

Definition 1 (Neighboring Datasets). *Two datasets are neighboring, if and only if one of them can be obtained by removing a tuple from the other one.*

Definition 2 (Differential Privacy [10]). *A randomized algorithm F satisfies (ϵ, δ)-differential privacy (DP), if for any two neighboring datasets D_1 and D_2 and for any set O of possible outputs of F, we have*

$$\Pr[F(D_1) \in O] \leq e^{\epsilon} \cdot \Pr[F(D_2) \in O] + \delta. \tag{2}$$

2.2 Gaussian Mechanism

To make an algorithm F differentially private, a typical approach is to inject noise into F so that its output distribution satisfies Eq. (2). The amount of noise needed depends on the parameters ϵ and δ of DP, the noise distribution, as well as how *sensitive* F is with respect to the addition or omission of a tuple in its input. In this paper, we adopt the *Gaussian mechanism* [2], which injects Gaussian noise based on L_2 *sensitivity* of the algorithm, as defined in the following.

Definition 3 (L_2 Sensitivity [11]). *Let f be a function that maps its input data to a h-dimensional vector in $\mathbb{R}^h$. The L_2 sensitivity of*

f, denoted as $S(f)$, is defined as

$$S(f) = \max_{\text{neighboring datasets } D_1, D_2} \|f(D_1) - f(D_2)\|_2.$$

The following lemma shows the amount of Gaussian noise needed to achieve (ϵ, δ)-DP.

Lemma 1 ([2]). *Let f be a function that maps the input data to a h-dimensional vector in $\mathbb{R}^h$. Suppose that we add i.i.d. Gaussian noise $\mathcal{N}(0, \sigma^2)$ to each coordinate of f's output. Then, the noisy f satisfies (ϵ, δ)-DP if and only if*

$$\Phi\left(\frac{S(f)}{2\sigma} - \frac{\epsilon\sigma}{S(f)}\right) - \exp(\epsilon) \cdot \Phi\left(-\frac{S(f)}{2\sigma} - \frac{\epsilon\sigma}{S(f)}\right) \leq \delta, \quad (3)$$

where $\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} \exp\left(-\frac{y^2}{2}\right) dy$ is the cumulative distribution function of the standard normal distribution, and $S(f)$ is the L_2 sensitivity of f.

2.3 PrivBayes

PrivBayes [38] generates synthetic data using a 3-step approach:

(1) Select a set $\mathcal{M}$ of marginals from the input data D using a differentially private algorithm.
(2) Inject noise into the concatenated marginal distribution vector $T_{D,\mathcal{M}}$, obtaining a noisy version $\tilde{T}_{D,\mathcal{M}}$.
(3) Use $\tilde{T}_{D,\mathcal{M}}$ to synthesize a noisy dataset $\tilde{D}$.

In particular, Step (1) of PrivBayes ensures that the marginals in $\mathcal{M}$ can be arranged in a sequence $M_1, M_2, \ldots,$ such that for any $i \geq 2$, M_i contains *exactly one* attribute Y_i that is not in $M_1, M_2, \ldots, M_{i-1}$. As such, the marginals in $\mathcal{M}$ correspond to a specific type of *Bayesian network* [21], which is a probabilistic model of the dependencies among the attributes in D. Based on the Bayesian network, we can approximate the data distribution in D as follows:

$$\Pr[A_1, A_2, \ldots, A_d] \approx \Pr[M_1] \cdot \prod_{i=2}^{|\mathcal{M}|} \Pr[Y_i \mid M_i \setminus \{Y_i\}]. \quad (4)$$

Step (2) of PrivBayes generates a noisy version $\tilde{T}_{D,M_i}$ of each marginal T_{D,M_i}. Observe that, for any $i \geq 2$, we can derive a noisy version of $\Pr[Y_i \mid M_i \setminus \{Y_i\}]$ based on the noisy counts in $\tilde{T}_{D,M_i}$. In addition, we can obtain a noisy version of $\Pr[M_1]$ from $\tilde{T}_{D,M_1}$ in the same manner. Given these noisy distributions, Step (3) of PrivBayes generates each synthetic tuple $\tilde{x}$ in $\tilde{D}$ as follows. First, we sample $\tilde{x}_{M_1}$ from the noisy $\Pr[M_1]$. Then, for each $i = 2, \ldots, |\mathcal{M}|$, we sample $\tilde{x}_{Y_i}$ based on the noisy $\Pr[Y_i \mid M_i \setminus \{Y_i\}]$ and the sampled values of $\tilde{x}$ on M_1 and $Y_2, \ldots, Y_{i-1}$. For example, suppose that $\mathcal{M}$ contains the two marginals M_1 and M_2 in Tables 5 and 6, respectively. Then, in accordance to Eq. (1), we would generate each synthetic tuple $\tilde{x}$ by first sampling $\tilde{x}_{\{A_1,A_2\}}$ from the noisy version of $\Pr[A_1, A_2]$. After that, we examine the sampled value of $\tilde{x}_{\{A_1,A_2\}}$, and then sample $\tilde{x}_{A_3}$ from $\Pr[A_3 \mid A_1]$ based on $\tilde{x}_{A_1}$.

PrivBayes's utilization of Bayesian networks yields two advantages. First, it enables us to approximate the input data D using the low-dimensional marginals in $\mathcal{M}$, thus mitigating the curse of dimensionality. Second, it allows us to generate synthetic tuples efficiently using the noisy marginals and Eq. (4). Nevertheless, the way that PrivBayes constructs Bayesian networks considerably restricts the choices of marginals in $\mathcal{M}$. Specifically, it requires that each M_i should contain exactly one attribute absent from $M_1, \ldots, M_{i-1}$, due to which the number of marginals in $\mathcal{M}$ is at most d. Further, each marginal can only have a small number of attributes. With these constraints, it is challenging to ensure that $\mathcal{M}$ sufficiently captures the dependencies among the attributes in D. As a consequence, the quality of the synthetic data generated by PrivBayes still leaves room for improvement.

2.4 PGM and Junction Trees

To overcome the limitation of PrivBayes, we may relax its constraints and let $\mathcal{M}$ contain marginals that do not correspond to a Bayesian network. But then, an immediate challenge is that we can no longer apply Eq. (4) to approximate the joint distribution in the input data D using $\tilde{T}_{D,\mathcal{M}}$. Fortunately, there exist a relatively efficient algorithm, referred to as PGM [25], for deriving the joint distribution based on an arbitrary set of noisy marginals.

The basic idea of PGM is to construct a *junction tree* [34] from $\mathcal{M}$ to facilitate the approximation of $\Pr[A_1, A_2, \ldots, A_d]$. In particular, a junction tree consists of an ordered set of marginals $\mathcal{C} = \{C_1, C_2, \ldots\}$ that satisfies the following two conditions:

(1) Each attribute in D is contained in at least one marginal in $\mathcal{C}$;
(2) For any $i > 1$, the attribute set $S_i = C_i \cap \left(\cup_{j=1}^{i-1} C_j\right)$ appears in one of $C_1, C_2, \ldots, C_{i-1}$.

Formally, each C_i (resp. S_i) is referred to a *clique* (resp. *separator*). By the property of the junction tree, we can approximate the data distribution in D as:

$$\Pr[A_1, A_2, \ldots, A_d] \approx \Pr[C_1] \cdot \prod_{i=2}^{|C|} \Pr[C_i \setminus S_i \mid S_i]. \quad (5)$$

Specifically, PGM obtains a junction tree $\mathcal{C}$ from $\mathcal{M}$ using a standard algorithm as follows. (Interested readers are referred to [34] for details.)

(1) Construct an *attribute graph* G where (i) each node is an attribute in D, and (ii) there is an undirected edge between two nodes A and B if and only if A and B appear in the same marginal in $\mathcal{M}$.
(2) *Triangulate* G by inserting edges into G, such that for every cycle $(v_1, \ldots, v_k, v_1)$ in G, the subgraph induced by $v_1, \ldots, v_k$ contains at least one triangle. For example, if (v_1, v_2, v_3, v_4) is a cycle in G, then (v_1, v_3) or (v_2, v_4) should be an edge in G.
(3) Let $G_\triangle$ be the triangulated version of G. Take each maximal clique in $G_\triangle$ as a clique in $\mathcal{C}$.
(4) Derive a proper order of the cliques so that $\mathcal{C} = \{C_1, C_2, \ldots\}$ satisfies the requirements of a junction tree.

The above algorithm incurs zero privacy cost, since it only examines the attributes in each marginal in $\mathcal{M}$, without looking into the counts in the marginals. In addition, the algorithm guarantees that each marginal in $\mathcal{M}$ is contained in at least one marginal in $\mathcal{C}$. For convenience, we refer to the transformation from $\mathcal{M}$ to $\mathcal{C}$ as the *junction tree transform*. Once $\mathcal{C}$ is obtained, PGM derives the marginal distributions in $\mathcal{C}$ from $\tilde{T}_{D,\mathcal{M}}$ by performing graphical model estimation and inference. After that, we can generate synthetic data based on Eq. (5).

Limitation of PGM. Although PGM offers an effective means of generating synthetic data from a set of noisy marginals, it does not provide any method to decide which marginals should be used in the first place. In other words, it requires that the noisy marginals should be given by the user. However, the quality of the synthetic data generated from PGM highly depends on the given set $\mathcal{M}$ of marginals. For example, if $\mathcal{M}$ contains d single-attribute marginals (*i.e.*, one for each attribute in D), then the joint data distribution derived from $\mathcal{M}$ would be a Cartesian product of the single-attribute marginal distributions. Such a joint data distribution is highly unlikely to be close to the original data distribution in D. One may attempt to address this problem by inserting into $\mathcal{M}$ every possible multi-dimensional marginal that contains correlated attributes, but it would still lead to inferior synthetic data due to two issues:

Issue 1: Excessive noise in marginals. Recall that we need to inject noise into each marginal in $\mathcal{M}$ to achieve DP. When $\mathcal{M}$ contains a large number of marginals, the amount of private information revealed by $\mathcal{M}$ is substantial, in which case we would require a large amount of noise in each marginal for privacy protection. This severely degrades the accuracy of the noisy marginals and the resulting synthetic data. □

Issue 2: Prohibitive overheads. Recall that PGM derives a joint data distribution based on Eq. (5), which requires materializing the marginal distribution for each clique C_i in a junction tree. In turn, each C_i is a maximal clique in the graph $G_\triangle$ triangulated from the attribute graph G. When M contains a sizable number of marginals, G would be a dense graph, since every pair of attributes appearing in the same marginal is mapped to an edge in G. Accordingly, G_Δ would also be dense (since it contains all edges in G), and hence, it is likely that some maximal clique C_i in $G_\triangle$ contains a large number of attributes. In that case, the marginal corresponding to C_i is a high-dimensional marginal, which cannot be materialized without prohibitive space and time overheads. □

In summary, we can conclude that $\mathcal{M}$ needs to be carefully constructed according to three constraints:

CSTR1. The marginals in $\mathcal{M}$ should be low-dimensional and should capture important characteristics of the input data;
CSTR2. The size of $\mathcal{M}$ is reasonably small;
CSTR3. Applying a junction tree transform on $\mathcal{M}$ does not result in any clique with large domain size.

Unfortunately, there is no existing algorithm for constructing $\mathcal{M}$ that satisfies all of the above constraints. For example, the marginal selection algorithm in PrivBayes [38] satisfies CSTR2 and CSTR3 (due to the property of the Bayesian network that it uses), but it violates CSTR1 because, as mentioned in Section 2.3, its choice of marginals is inadequate to capture complex correlations in the input data. Previous work [25] also considers combining PGM with a few other existing algorithms [13, 16, 24], but shows that none of them yields better overall result than combining PGM with the marginal selection algorithm in PrivBayes. This leads to a natural question: can we design a marginal selection algorithm that satisfies DP as well as constraints CSTR1-3, so as to achieve more effective synthetic data generation? In Section 3, we answer this question positively with a new DP algorithm for marginal selection.

Table 7: Comparison of PrivMRF, PrivBayes [38], and PGM [25].

Method	Selection of Marginals	Modeling from noisy marginals
PrivBayes	based on a Bayesian network	uses a Bayesian network
PGM	-	uses a probablistic graphical model
PrivMRF	based on Markov random fields	uses PGM

3 OUR SOLUTION

This section introduces our PrivMRF method for data synthesis under differential privacy. In a nutshell, PrivMRF utilizes a DP algorithm to select a marginal set $\mathcal{M}$ that satisfies the constraints CSTR1-3 mentioned in Section 2.4, and then it feeds a noisy distribution vector $\tilde{T}_{D,\mathcal{M}}$ of $\mathcal{M}$ to PGM to generate synthetic data. Table 7 shows a comparison of PrivMRF, PrivBayes [38], and PGM [25]. The novelty of PrivMRF lies in its new DP method to select marginals that can be combined with PGM for effective synthetic data generation. In contrast, PGM itself does not provide any method for marginal selection, while PrivBayes has a restrictive marginal selection approach that degrades the accuracy of synthetic data, as discussed in Section 2.

Specifically, PrivMRF uses a two-step approach for marginal selection. First, it generates an attribute graph G, such that (i) each edge in G connects two highly correlated attributes, and (ii) the triangulation $G_\triangle$ of G does not contain any large cliques. Let $\mathcal{C} = \{C_1, C_2, \ldots\}$ be the maximal cliques in $G_\triangle$.

Second, PrivMRF selects a set $\mathcal{M}$ of marginals, such that each $M_i \in \mathcal{M}$ contains a subset of the attributes in some clique in $\mathcal{C}$. As such, if we apply a junction tree transform on $\mathcal{M}$, each clique in the resulting junction tree would be a sub-clique of some $C_j \in \mathcal{C}$ [21]. Therefore, the largest clique in the junction tree is at most the same size as the largest clique in $\mathcal{C}$. This guarantees that $\mathcal{M}$ satisfies CSTR3. To ensure that $\mathcal{M}$ also satisfies CSTR1 and CSTR2, PrivMRF carefully controls the size of $\mathcal{M}$, and it selects a marginal M_i into $\mathcal{M}$ only if M_i is low-dimensional and provides useful additional information with respect to those marginals that have been selected.

In what follows, Section 3.1 explains how PrivMRF generates the attribute graph G, while Sections 3.2-3.4 detail the construction of marginal set $\mathcal{M}$ based on $\mathcal{C}$. After that, Section 3.5 clarifies the synthesis of data based on $\mathcal{C}$, and Section 3.6 analyzes the privacy guarantee of PrivMRF.

3.1 Generation of Attribute Graph

Given a dataset D, PrivMRF starts by generating an attribute graph G where each node is an attribute in D. It aims to ensure that (i) each edge in G links up two attributes that are highly correlated, and (ii) if we triangulate G, the resulting graph $G_\triangle$ does not contain any large clique. Once G is generated, the maximal cliques in $G_\triangle$ would be used in subsequent steps of PrivMRF to construct a marginal set for synthetic data generation.

Algorithm 1 illustrates the pseudo-code of PrivMRF's construction of $G_\triangle$. It first examines every pair of attributes A, B in D, and

Algorithm 1: Construction of Attribute Graph

Input: Dataset D, noise scale σ_R, threshold τ_c
Output: Set C of cliques, noisy R-score $\tilde{R}(A, B)$ for each pair of attributes A, B

1 **for** *each pair of attributes* $A, B \in D$ **do**
2 $\tilde{R}(A, B) \leftarrow R(A, B) + \mathcal{N}(0, \sigma_R^2)$;
3 Let $\mathcal{E}$ be the set of all attribute pairs in D;
4 Let G be an edgeless graph where each node is an attribute in D;
5 $bool \leftarrow$ true;
6 **while** *bool is true* **do**
7 $bool \leftarrow$ false;
8 **for** *each attribute pair* (A, B) *in* $\mathcal{E}$ *in descending order of* $\tilde{R}(A, B)$ **do**
9 Let $G_\triangle$ be triangulation of G, assuming edge (A, B) is inserted into G;
10 **if** *every maximal clique in* $G_\triangle$ *has a domain size no more than* τ_c **then**
11 insert edge (A, B) into G, and remove (A, B) from $\mathcal{E}$;
12 $bool \leftarrow$ true;
13 **break**;
14 $C \leftarrow$ the set of maximal cliques in the graph $G_\triangle$ triangulated from G;
15 **return** C as well as $\tilde{R}(A, B)$ for each attribute pair A, B;

computes their *R-score* $R(A, B)$ [38] as follows:

$$R(A, B) = \frac{n}{2} \left\| \Pr[A, B] - \Pr[A] \Pr[B] \right\|_1, \tag{6}$$

where n is the number of tuples in D. Observe that $R(A, B)$ measures the difference between $\Pr[A, B]$ and $\Pr[A] \cdot \Pr[B]$, and it tends to be large when A and B are highly correlated. PrivMRF tries to ensure that node pairs with high R-score are mapped to edges in G. Towards this end, it first injects Gaussian noise $\mathcal{N}(0, \sigma_R^2)$ into the R-score $R(A, B)$ of each attribute pair A, B (Lines 1-2 of Algorithm 1). Let $\tilde{R}(A, B)$ be the noisy version of $R(A, B)$ thus obtained.

Then, PrivMRF greedily selects node pairs with large noisy R-scores, and adds each pair (A, B) into G as an edge, subject to one constraint: if we triangulate G after inserting (A, B), each maximal clique C in the triangulated graph $G_\triangle$ should have

$$\prod_{A \in C} |A| \leq \tau_c, \tag{7}$$

where $|A|$ denotes the domain size of A and τ_c is a constant. In other words, we require that the marginal corresponding to C has a domain size at most τ_c, which ensures that there is no over-size clique in $G_\triangle$.

Specifically, PrivMRF first initializes a set $\mathcal{E}$ that contains all attribute pairs in D, as well as an edgeless graph G where each node is an attribute in D (Lines 3-4). After that, PrivMRF iteratively inserts edges into G, while making sure that the constraint in Eq. (7) is satisfied (Lines 5-13). Specifically, in each iteration, PrivMRF identifies the attribute pair (A, B) in $\mathcal{E}$ with the largest noisy R-score, and considers the scenario when we (i) add an edge between A and B in G and then (ii) triangulate G into another graph $G_\triangle$ (Lines 8-9). If every maximal clique in $G_\triangle$ has a domain size smaller than τ_c, then PrivMRF inserts an edge (A, B) into G and removes (A, B) from $\mathcal{E}$, and proceeds to the next iteration (Lines 10-13). Otherwise, PrivMRF ignores (A, B) and considers the node pair in $\mathcal{E}$ with the *next largest* noisy R-score, and so on (Line 10). When none of the node pairs in $\mathcal{E}$ can be inserted into G without violating the constraint of τ_c, PrivMRF terminates by returning the set C of maximal cliques in the triangulation of G, as well as $\tilde{R}(A, B)$ for each attribute pair A, B (Lines 14-15).

3.2 Choosing Candidate Marginal Set

Let C be the set of cliques returned by Algorithm 1, and $G_\triangle$ be the triangulated graph from which C is generated. Given C, PrivMRF would proceed to construct a set $\mathcal{U}$ of *candidate marginals*, which will subsequently be used to derive the final marginal set $\mathcal{M}$ for synthetic data generation. The construction of $\mathcal{M}$ serves two purposes: (i) to filter out marginals that violates the constraint CSTR1 or CSTR3 mentioned in Section 2.4, and (ii) to narrow the scope of marginal selection in subsequent steps. Toward this end, PrivMRF imposes the following two requirements on the marginals in $\mathcal{U}$.

First, each marginal in $\mathcal{U}$ should contain a subset of the attributes of some clique in C. This guarantees that if we choose any subset $\mathcal{M}$ of $\mathcal{U}$ and apply a junction tree transform on $\mathcal{M}$, then any clique in the resulting junction tree is a sub-clique of some $C_j \in C$ [21]. In other words, the junction tree would not contain any clique whose domain size is larger than τ_c (see Eq. (7)), thus satisfying the constraint CSTR3.

Second, each marginal M in $\mathcal{U}$ should be θ-*useful* [38], namely,

$$\frac{n}{\prod_{A \in M} |\mathcal{X}_A|} \geq \theta \cdot g,$$

where g is the expected absolute value of the noise to be injected into each count in M, and θ is a constant. In other words, we require that the average count in M should be at least θ times the noise scale g, so that the noisy version of M could still capture useful information, which helps comply with the constraint CSTR1.

Algorithm 2 shows the pseudo-code of the derivation of $\mathcal{U}$. We first generate a noisy version $\tilde{n}$ of the number n of tuples in D, and initialize $\mathcal{U}$ as an empty set (Lines 1-2). Then, for each marginal M such that $M \subseteq C_i$ for some $C_i \in C$, we evaluate whether it satisfies θ-usefulness based on whether $\frac{\tilde{n}}{\prod_{A \in M} |\mathcal{X}_A|} \geq \theta \cdot g$ (Lines 3-5). (Note that we use $\tilde{n}$ instead of n, since n is sensitive information and needs to be perturbed with noise to ensure DP.) If $\frac{\tilde{n}}{\prod_{A \in M} |\mathcal{X}_A|} \geq \theta \cdot g$, then we insert M into $\mathcal{U}$ as a marginal to be considered (Line 6).

After we obtain $\mathcal{U}$, we proceed to select marginals from $\mathcal{U}$ to form the final marginal set $\mathcal{M}$, in a manner that satisfies constraints CSTR1-3. For this, we first initialize $\mathcal{M}$ with d marginals selected heuristically from $\mathcal{U}$ (see Section 3.3), and then iteratively apply a greedy approach to insert additional marginals into $\mathcal{M}$ (see Section 3.4).

3.3 Initialization of Marginal Set $\mathcal{M}$

Algorithm 3 shows the pseudo-code of our initialization of $\mathcal{M}$. We first set $\mathcal{M} = \emptyset$, and then inspect each attribute A in D in turn. For

Algorithm 2: Construction of Candidate Marginal Set

Input: Set C of cliques, noise scale $\sigma_{\mathcal{U}}$, constant θ, expected noise amount g
Output: Candidate marginal set $\mathcal{U}$

$\tilde{n} \leftarrow n + \mathcal{N}(0, \sigma_{\mathcal{U}}^2)$;
$\mathcal{U} \leftarrow \emptyset$;
for *each clique* $C_i \in C$ **do**
  for *each* $M \subseteq C_i$ **do**
    if $\frac{\tilde{n}}{\prod_{A \in M} |\mathcal{X}_A|} \geq \theta \cdot g$ **then**
      Insert M into $\mathcal{U}$;
return $\mathcal{U}$;

Algorithm 3: Initialization of Marginal Set $\mathcal{M}$

Input: Candidate marginal set $\mathcal{U}$, attributes set $\mathcal{A}$ in input data D, $\tilde{R}$ for each attribute pairs
Output: Marginal set $\mathcal{M}$

$\mathcal{M} \leftarrow \emptyset$;
for *each attribute* $A \in \mathcal{A}$ **do**
  Among the marginals in $\mathcal{U}$ that contains A, identify the marginal M that maximizes $\rho(A, M)$;
  Insert M into $\mathcal{M}$, and remove M from $\mathcal{U}$;
return $\mathcal{M}$;

each A, we examine the marginals in $\mathcal{U}$ that contains A; among them, we identify the marginal M that maximizes the following function $\rho(A, M)$:

$$\rho(A, M) = \frac{\sum_{B \in M \setminus \{A\}} \tilde{R}(A, B)}{\sqrt{|M| + \sum_{B \in M \setminus \{A\}} \left(\sum_{B' \in M \setminus \{A, B\}} \tilde{R}(B, B') \right)}}. \quad (8)$$

Then, we insert M into $\mathcal{M}$, and remove M from $\mathcal{U}$. We return $\mathcal{M}$ after repeating the above process for all attributes in D.

The function $\rho(A, M)$ measures the correlations between A and the other attributes in M, and is based on the function used in *correlation-based feature selector* [14] for evaluating the correlation between an attribute and a feature set. By inserting into $\mathcal{M}$ the marginal M that contains A and maximizes $\rho(A, M)$, we ensure that $\mathcal{M}$ contains at least one marginal that captures the strong correlations between A and other attributes.

3.4 Refinement of Marginal Set $\mathcal{M}$

After the initialization of $\mathcal{M}$, we proceed to refine $\mathcal{M}$ by inserting additional marginals into $\mathcal{M}$, aiming to ensure that each inserted marginal captures as much *additional* information as possible with respect to the existing marginals in $\mathcal{M}$. Towards this end, we construct a *Markov random field (MRF)* [34], which is a graphical model for representing the information encapsulated in $\mathcal{M}$. In particular, our MRF is parameterized with a real vector $\boldsymbol{\theta}$ where each element corresponds to an entry in a marginal $M \in \mathcal{M}$. Let $\boldsymbol{x}$ be an element of the domain $\mathcal{X}$ of D, and $\boldsymbol{x}_M$ be the projection of $\boldsymbol{x}$ onto the attributes in M. Let $\boldsymbol{\theta}_M(\boldsymbol{x}_M)$ be the element in θ corresponding to

Algorithm 4: Refinement of Marginal Set $\mathcal{M}$

Input: input data D, clique set C, candidate marginal set $\mathcal{U}$, initial marginal set $\mathcal{M}$, iteration number t, constant k, noise scales σ_m and σ_h
Output: the marginal distributions pertinent to C

Obtain a noisy distribution vector $\tilde{T}_{D,\mathcal{M}}$ by injecting i.i.d. Gaussian noise $\mathcal{N}(0, \sigma_m^2)$ into each entry in $T_{D,\mathcal{M}}$;
Set the initial MRF parameter $\boldsymbol{\theta}_0$ as an all-zero vector of length $\left|\tilde{T}_{D,\mathcal{M}}\right|$;
for $i = 1$ **to** t **do**
  $\boldsymbol{\theta}_i \leftarrow$ Algorithm 5 $\left(\tilde{T}_{D,\mathcal{M}}, \boldsymbol{\theta}_{i-1}\right)$;
  $\mathcal{U}' \leftarrow$ a random sample set of k marginals from $\mathcal{U}$;
  for *each marginal* $M \in \mathcal{U}'$ **do**
    $\boldsymbol{\mu}_{i,M} \leftarrow$ the marginal distribution vector of M decided by $\boldsymbol{\theta}_i$;
    $\tilde{h}(M) \leftarrow \|\boldsymbol{\mu}_{i,M} - T_{D,M}\|_1 + \mathcal{N}(0, \sigma_h^2)$;
  Let M' be the marginal in $\mathcal{U}'$ that maximizes $\tilde{h}(M')$;
  Insert M' into $\mathcal{M}$, and remove M' from $\mathcal{U}$;
  Obtain a noisy distribution vector $\tilde{T}_{D,M'}$ by injecting i.i.d. Gaussian noise $\mathcal{N}(0, \sigma_m^2)$ into each entry in $T_{D,M'}$;
  $\tilde{T}_{D,\mathcal{M}} \leftarrow$ the concatenation of $\tilde{T}_{D,\mathcal{M}}$ and $\tilde{T}_{D,M'}$;
  Update $\boldsymbol{\theta}_i$ by inserting zeroes at positions corresponding to the entries in M';
$\boldsymbol{\theta}_i \leftarrow$ Algorithm 5 $\left(\tilde{T}_{D,\mathcal{M}}, \boldsymbol{\theta}_{i-1}\right)$;
Use $\boldsymbol{\theta}_i$ to infer the marginal distributions pertinent to C;
return the marginal distributions pertinent to C;

$\boldsymbol{x}_M$. Our MRF models the distribution of $\boldsymbol{x}$ as:

$$p(\boldsymbol{x}) \propto \prod_{M \in \mathcal{M}} \exp\left(\boldsymbol{\theta}_M(\boldsymbol{x}_M)\right), \quad (9)$$

where $\exp\left(\boldsymbol{\theta}_M(\boldsymbol{x}_M)\right)$ is referred as the *potential function* for $\boldsymbol{x}_M$. In other words, given $\boldsymbol{\theta}$, we can compute $p(\boldsymbol{x})$ by first taking the product of the potential function of $\boldsymbol{x}$'s projection onto each $M \in \mathcal{M}$, and then normalizing it against all $\boldsymbol{x} \in \mathcal{X}$. Accordingly, we can use $\boldsymbol{\theta}$ to infer the marginal distribution of any marginal M' as:

$$p_{M'}(y) = \sum_{\boldsymbol{x} \in \mathcal{X}, \boldsymbol{x}_{M'} = y} p(x). \quad (10)$$

In addition, it is shown in [34] that, for any M', we can avoid materializing $p(x)$ in the computation of $p_{M'}(y)$, by utilizing a junction tree constructed from $\mathcal{M}$.

Overall, the MRF enables us to use $\mathcal{M}$ to infer the marginal distributions pertinent to C as follows:

(1) Given $\mathcal{M}$, we derive the parameter vector $\boldsymbol{\theta}$ of the MRF based on the noisy marginal distribution vector $\tilde{T}_{D,\mathcal{M}}$ of $\mathcal{M}$.
(2) Based on $\boldsymbol{\theta}$, we infer the marginal distribution of each clique $C_i \in C$, according to Eq. (10).

Apparently, this approach is effective only if $\mathcal{M}$ and $\boldsymbol{\theta}$ are chosen carefully to facilitate an accurate estimation of C_i's marginal distribution. In what follows, we explain how we address this in our method by refining $\mathcal{M}$.

Algorithm 5: Mirror Descent Estimation

Input: noisy marginal distribution vector $\tilde{T}_{D,\mathcal{M}}$, MRF parameter vector $\boldsymbol{\theta}_{i-1}$
Output: updated parameter vector $\boldsymbol{\theta}_i$, updated marginal distribution vector $\boldsymbol{\mu}_i$

1 $\boldsymbol{\theta}_i \leftarrow \boldsymbol{\theta}_{i-1}$;
2 **repeat**
3 $\quad \boldsymbol{\mu}_i \leftarrow$ the marginal distribution vector of $\mathcal{M}$ decided by $\boldsymbol{\theta}$ according to Eq. (10);
4 $\quad \boldsymbol{\theta}_i \leftarrow \boldsymbol{\theta}_i - \eta \cdot (\boldsymbol{\mu}_i - \tilde{T}_{D,M})$;
5 **until** $\boldsymbol{\mu}_i$ *converges*;
6 **return** $\boldsymbol{\theta}_i$

Algorithm 6: PrivMRF

Input: input data D, attribute set $\mathcal{A}$, noise scales σ_R, $\sigma_{\mathcal{U}}$, σ_h, and σ_m, constants τ_c, θ, t, and k
Output: the marginal distributions pertinent to C

1 $C, \{\tilde{R}\} \leftarrow$ Algorithm 1 (D, σ_R, τ_c);
2 $\mathcal{U} \leftarrow$ Algorithm 2 $\left(C, \sigma_{\mathcal{U}}, \theta, \sigma_m\sqrt{2/\pi}\right)$;
3 $\mathcal{M} \leftarrow$ Algorithm 3 $\left(\mathcal{U}, \mathcal{A}, \{\tilde{R}\}\right)$;
4 **return** the output of Algorithm 4 $(D, C, \mathcal{U}, \mathcal{M}, t, k, \sigma_m, \sigma_h)$

Algorithm 4 shows the pseudo-code of our method. We first generate the noisy marginal distribution vector $\tilde{T}_{D,\mathcal{M}}$ of $\mathcal{M}$ by injecting i.i.d. Gaussian noise into $T_{D,\mathcal{M}}$ (Line 1 in Algorithm 4). After that, we initialize the MRF parameter as a vector $\boldsymbol{\theta}_0$ that contains $|\tilde{T}_{D,\mathcal{M}}|$ zero elements (Line 2).

The subsequent part of the algorithm consists of t iterations (Lines 3-13). In the i-th ($i \geq 1$) iteration, we first invoke Algorithm 5 to update $\boldsymbol{\theta}_{i-1}$ into a new parameter vector $\boldsymbol{\theta}_i$ that is as consistent as $\tilde{T}_{D,\mathcal{M}}$ as possible. Algorithm 5 is an adoption of the *mirror descent* method used in PGM [25] for estimating graphical model parameters from noisy marginals. It first sets $\boldsymbol{\theta}_i = \boldsymbol{\theta}_{i-1}$, and then derives the marginal distribution vector $\boldsymbol{\mu}_i$ decided by $\boldsymbol{\theta}_i$ based on Eq. (10). After that, it measures the difference between $\boldsymbol{\mu}_i$ and $\tilde{T}_{D,\mathcal{M}}$, and updates $\boldsymbol{\theta}_i$ based on this difference (Line 4 of Algorithm 5). Subsequently, it repeatedly computes $\boldsymbol{\mu}_i$ from $\boldsymbol{\theta}_i$ and then updates $\boldsymbol{\theta}_i$ based on $\boldsymbol{\mu}_i - \tilde{T}_{D,\mathcal{M}}$, until $\boldsymbol{\mu}_i$ converges. This iterative process makes $\boldsymbol{\mu}_i$ as close to $\tilde{T}_{D,\mathcal{M}}$ as possible, by adjusting the parameter vector $\boldsymbol{\theta}$. Finally, the algorithm returns $\boldsymbol{\theta}_i$ and terminates.

After obtaining $\boldsymbol{\theta}_i$ and $\boldsymbol{\mu}_i$, we proceed to select a random set $\mathcal{U}'$ of marginals from $\mathcal{U}$, and choose the "best" among them to insert into $\mathcal{M}$ (Lines 5-8 in Algorithm 4). (We avoid inspecting all marginals in $\mathcal{U}$, so as to reduce the privacy and computation overheads incurred.) Specifically, for each marginal $M \in \mathcal{U}'$, we compute its marginal distribution vector $\mu_{i,M}$ decided by $\boldsymbol{\theta}_i$, and then computes $\tilde{h}(M) \leftarrow \|\boldsymbol{\mu}_{i,M} - \boldsymbol{T}_{D,M}\|_1 + \mathcal{N}(0, \sigma_h^2)$ as a noisy version of the L_1 distance between $\boldsymbol{\mu}_{i,M}$ and $\boldsymbol{T}_{D,M}$. Intuitively, a large $\tilde{h}(M)$ indicates that our MRF based on the current marginal set $\mathcal{M}$ is unable to accurately model the information in M; in that case, it is beneficial to insert M into $\mathcal{M}$ and update $\boldsymbol{\theta}_i$ accordingly, so as to improve the quality of our MRF. Therefore, we identify the marginal M' in $\mathcal{U}'$ with the maximum $\tilde{h}(M')$, and insert M' into $\mathcal{M}$ after removing it from $\mathcal{U}$ (Lines 9-10). After that, we update $\tilde{T}_{D,\mathcal{M}}$ and $\boldsymbol{\theta}_i$ to reflect the insertion of M' into $\mathcal{M}$ (Lines 11-13), and then proceed to the next iteration to insert new marginals. After t iterations, we invoke Algorithm 5 one last time to update $\boldsymbol{\theta}_i$, and then use it to infer the marginal distributions pertinent to C, which are then returned as the output of Algorithm 4.

Discussion. One may wonder why we construct the initial marginal set $\mathcal{M}$ based on the function ρ in Eq. (8) but refine $\mathcal{M}$ based on an MRF instead. The reason is as follows. First, for the refinement of $\mathcal{M}$, we need to identify new marginals that *complements* those in $\mathcal{M}$ in terms of the information captured. The function ρ is unsuitable for this task, since it only measures the correlations among the attributes in the same marginal, but does not lead to a meaningful way to infer whether a new marginal is useful with respect to the existing marginal set $\mathcal{M}$. In contrast, the MRF provides us a principled approach to gauge the amount of new information that a new marginal provides on top of $\mathcal{M}$. Therefore, the MRF is much more suitable for the refinement of $\mathcal{M}$.

Alternatively, one may omit Algorithm 3, and directly invoke Algorithm 4 with $\mathcal{M} = \emptyset$ to construct $\mathcal{M}$ from scratch. In that case, however, the first few iterations of Algorithm 4 (in Lines 3-13) would have to evaluate a large random set $\mathcal{U}'$ of marginals, so as to ensure that $\mathcal{U}'$ contains good choices for the initialization of $\mathcal{M}$. This would incur tremendous computation and privacy costs, since the evaluation of each marginal $M \in \mathcal{U}'$ requires (i) deriving the marginal distribution of M from the MRF parameter vector and (ii) injecting noise to the L_1 distance between this marginal distribution and $T_{D,M}$. In contrast, initializing $\mathcal{M}$ using Algorithm 3 strikes a much better trade-off between efficiency, privacy, and the quality of $\mathcal{M}$, which explains why we incorporate both Algorithms 3 and 4 in PrivMRF.

3.5 Synthesizing Data from C

Given the marginal distributions of C returned by Algorithm 5, we generate each synthetic tuple $\tilde{x}$ based on Eq. (5) as follows. First, we sample $\tilde{x}_{C_1}$ (*i.e.*, the subvector of $\tilde{x}$ on C_1) from the marginal distribution of C_1. After that, for $i = 2, 3, \ldots$, we examine $\tilde{x}_{S_i}$, and then sample $\tilde{x}_{C_i \setminus S_i}$ from the conditional distribution $\Pr[C_i \setminus S_i \mid S_i = \tilde{x}_{S_i}]$ (which can be derived from the marginal distribution of C_i). In other words, $\tilde{x}$ can be generated by sampling once from each $C_i \in C$. The generation of each synthetic tuple is processed independently.

3.6 Privacy Analysis

Algorithm 6 presents the pseudo-code of PrivMRF. There are only four places in PrivMRF that access the input dataset D:

(1) Lines 1-2 in Algorithm 1 computes the R-score of every pair of attributes in D, and injects Gaussian noise $\mathcal{N}(0, \sigma_R^2)$ into each R-score.
(2) Line 1 in Algorithm 2 derives a noisy version $\tilde{n}$ of the number of tuples in D, using Gaussian noise $\mathcal{N}(0, \sigma_{\mathcal{U}}^2)$.
(3) Lines 3-13 in Algorithm 4 consists of t iterations, each of which inspects k marginals in a random set $\mathcal{U}'$. For each

marginal $M \in \mathcal{U}'$, Algorithm 4 computes the L_1 distance between $\boldsymbol{\mu}_{i,M}$ and $T_{D,M}$, and injects Gaussian noise $\mathcal{N}(0, \sigma_h^2)$ into it.

(4) Each iteration in Lines 3-13 in Algorithm 4 also selects a marginal M' from $\mathcal{U}'$ and computes a noisy version $\tilde{T}_{D,M'}$ of its marginal distribution, using Gaussian noise $\mathcal{N}(0, \sigma_m^2)$.

The following lemma shows the privacy guarantee of PrivMRF.

Lemma 2. *Let*

$$g = \frac{2d(d-1)}{\sigma_R^2} + \frac{1}{\sigma_{\mathcal{U}}^2} + \frac{t \cdot k}{\sigma_h^2} + \frac{d+t}{\sigma_m^2}. \tag{11}$$

Then, PrivMRF *satisfies (ϵ, δ)-differential privacy if*

$$\Phi\left(\frac{\sqrt{g}}{2} - \frac{\epsilon}{\sqrt{g}}\right) - e^{\epsilon} \cdot \Phi\left(-\frac{\sqrt{g}}{2} - \frac{\epsilon}{\sqrt{g}}\right) \leq \delta. \tag{12}$$

The proof of Lemma 2 is included in the full version of this paper [28], and it utilizes the following three lemmas.

Lemma 3. *For m queries $Q_1, Q_2, \ldots, Q_m$ with L_2 sensitivity $S(Q_1), S(Q_2), \ldots, S(Q_m)$, we inject independent Gaussian noise of standard deviation σ_i into the query result of Q_i. Let*

$$g = \sum_{i=1}^{m} \frac{S(Q_i)^2}{\sigma_i^2} \tag{13}$$

Then, the composition of the m queries satisfies (ϵ, δ)-DP if:

$$\Phi\left(\frac{\sqrt{g}}{2} - \frac{\epsilon}{\sqrt{g}}\right) - e^{\epsilon} \cdot \Phi\left(-\frac{\sqrt{g}}{2} - \frac{\epsilon}{\sqrt{g}}\right) \leq \delta. \tag{14}$$

Lemma 4. *The L_2 sensitivity of the R-score is at most* 2.

Lemma 5. *For any marginal M and any MRF parameter θ, let μ be the marginal distribution vector of M decided by θ, and $h(M) = \|\boldsymbol{\mu}_{i,M} - T_{D,M}\|_1$. Then, the L_2 sensitivity of h is* 1.

In our implementation of PrivMRF, we set its parameters as follows. First, given privacy parameters ϵ and δ, we compute the largest g that satisfies Eq. (12). Then, we set $\sigma_R, \sigma_{\mathcal{U}}, \sigma_h, \sigma_m$ so that

$$\frac{2d(d-1)}{\sigma_R^2} = 0.1 \cdot g, \qquad \frac{1}{\sigma_{\mathcal{U}}^2} = 0.01 \cdot g,$$
$$\frac{t \cdot k}{\sigma_h^2} = 0.1 \cdot g, \qquad \frac{d+t}{\sigma_m^2} = 0.79 \cdot g.$$

In other words, we allocate 10% of the "privacy budget" to computing R-scores in Algorithm 1, 1% to deriving $\tilde{n}$ in Algorithm 2, 10% to choosing a marginal $M' \in \mathcal{U}'$ in each iteration in Algorithm 4, and 79% to computing the marginal distributions pertinent to $\mathcal{M}$. In addition, we set $t = \lfloor 0.8 \cdot d \rfloor$, $k = 400$, and $\theta = 6$ (see Line 5 in Algorithm 3); these values are chosen based on our experiments in Section 5.2. Finally, for the threshold τ_c on the domain size of each maximal clique C in the triangulated graph $G_\triangle$ (see Algorithm 1), we set $\tau_c = 10^7$, which is a relatively large value that provides flexibility in the choices of C without incurring excessive space and time overheads on the machine used in our experiments.

4 OTHER RELATED WORK

In the existing literature on differential privacy (DP), the work most related to ours is on synthetic data generation and Markov random fields (MRF). In what follows, we review the existing methods on these two topics.

A comprehensive survey of non-DP data synthesis methods (*e.g.*, [33]) is beyond the scope of this paper.

Data Synthesis with DP. Early work on synthetic data generation with DP focuses on releasing the full distribution of low-dimensional data [3, 7, 18, 35]. The common idea is to first project the input data D onto a different space (*e.g.*, the Fourier domain [3] or the wavelet domain [35]) that is more resilient to noise injection, and then perturb the transformed data and project it back to the original space. As pointed out in [38], however, the solutions in [3, 7, 18, 35] all suffer from the curse of dimensionality, in the sense that their computation cost and/or data utility degrades significantly when d increases. PrivBayes [37, 38] addresses this issue by using a Bayesian network to approximate the input data distributions with a set M of noisy low-dimensional marginals. As we discuss in Section 2.3, however, PrivBayes has considerable restrictions on the choice of marginals in M, which degrades its ability to accurately model the input data. A subsequent study [6] attempts to improve PrivBayes using an alternative approach for Bayesian network construction, but as pointed in [39], the solution in [6] fails to achieve differential privacy, due to a gap in the theoretical analysis.

More recently, Bindschaedler *et al.* [5] also present a Bayesian-network-based approach (referred to as *BSG*) that is similar to PrivBayes but differs in the DP procedure used for Bayesian network construction. However, our experiments in Section 5 show that BSG does not offer better data utility than PrivBayes does.

There also exist a number of other DP data synthesis methods based on alternative techniques, including DP-Copula [22] (based on Copula functions), DP-WGAN [30] and PATE-GAN [19] (based on generative adversarial networks), and MWEM [16] and DualQuery [13] (based on game theoretic approaches). Among them, both MWEM and DualQuery require as input a set Q of linear queries, and the synthetic data that they generate is specifically optimized for Q. Similarly, PATE-GAN assumes that one of the attributes in the data is a dependent variable while all other attributes are explanatory variables, and it optimizes the synthetic data for classification of the dependent variable. In contrast, DP-Copula and DP-WGAN do not assume such prior knowledge of the query workload or machine learning tasks to be performed, which makes them more comparable to PrivMRF. Therefore, we include DP-Copula and DP-WGAN in our expeirments in Section 5, but omit MWEM, DualQuery, and PATE-GAN.

MRF with DP. Existing studies on Markov random fields (MRF) (*e.g.*, [15, 21, 23, 32, 34, 36, 40]) has mostly considered the non-private setting, and they typically tackles two issues:

(1) *Structure learning*: given a dataset D, how to identify a suitable set of potential functions to construct an MRF?

(2) *Parameter learning*: given a set of potential functions for an MRF, how to learn the parameter vector $\boldsymbol{\theta}$ of the MRF?

In the context of our paper, the structure learning of an MRF is equivalent to the selection of a marginal set $\mathcal{M}$ for MRF construction. To our knowledge, the only existing work on MRF structure learning under DP is a theoretical study by Zhang *et al.* [36], and it assumes that the MRF is either *pairwise* or *binary*. In our context, this means that either all marginals in $\mathcal{M}$ should contain at most two attributes, or all attributes in D should be Boolean. We note that these conditions seldom hold in practice. This motivates us to design new algorithms for MRF construction in PrivMRF.

In contrast, for MRF parameter learning with DP, existing work [4, 25] has presented two practical algorithms for real data. In particular, Bernstein *et al.* [4] present a DP parameter learning method based on expectation maximization, while McKenna *et al.* propose *PGM*, which is an improved method based on mirror descent that achieves faster convergence. Nevertheless, neither of these two methods addresses the structure learning problem, and hence, they require that the set M of marginals for the MRF is obtained in advance using other DP techniques, such as PrivBayes. As we show in Section 5, however, the combination of PrivBayes and PGM is significantly outperformed by PrivMRF, which demonstrates the effectiveness of the structure learning approach in PrivMRF.

5 EXPERIMENTS

This section empirically evaluates PrivMRF against the state of the art. All of our experiments are conducted on a machine with an Intel 2.6GHz 18-core CPU, 384GB RAM, and two Nvidia RTX 2080 Ti GPUs.

5.1 Settings

Datasets. We use four benchmark datasets that are also used in previous work [5, 6, 25, 38]. Table 8 shows the details of the datasets. ACS contains 47,461 records of personal information obtained from IPUMS [29]. Adult consists of 45,222 records from the 1994 US Census [9]. BR2000 contains 38,000 census records collected from Brazil in 2000 [29]. NLTCS consists of 21,574 records from the National Long Term Care Survey [20].

Tasks. We evaluate the performance of PrivMRF on two different tasks, namely, *SVM classification* and *α-way marginals*. For SVM classification, we use 80% of the dataset D as training data, and the other 20% as testing data. We generate a synthetic version of the training data, and then use it to train d SVM classifiers, such that the i-th classifier takes the i-th attribute in D as the target attribute, while using all other attributes as features. We measure the mis-classification rate of each classifier on the testing data, and compute the average mis-classification rate. Further, we use 5-fold cross-validation, and report the average results over 5 runs.

For α-way marginals, we generate a synthetic version of the input dataset D, and then randomly select 300 α-way marginals (*i.e.*, marginals containing α attributes each). For each marginal M, we compare the *total variation distance (TVD)* between the noisy and original versions of M, defined as $\frac{1}{n}\|T_{\tilde{D},M} - T_{D,M}\|_1$. For each $\alpha \in \{3, 4, 5\}$, we measure the average TVD over all α-marginals, and report the average measurement over 10 runs.

Table 8: Dataset characteristics

Dataset	# of Tuples	# of Attributes	Domain size
NLTCS	21,574	16	$\approx 6.55 \times 10^4$
ACS	47,461	23	$\approx 8.39 \times 10^6$
Adult	45,222	15	$\approx 9.06 \times 10^{14}$
BR2000	28,000	14	$\approx 3.23 \times 10^9$

Methods. We compare PrivMRF against five state-of-the-art methods, PrivBayes [38], PGM [25], BSG [5], DP-Copula [22], and DP-WGAN [30]. We implement PrivBayes[1], BSG, and PrivMRF using Python, and adopt the Python implementations of PGM, DP-Coupla, and DP-WGAN available from [26, 27, 30].

Note that PGM requires an external method that selects an appropriate set of marginals for synthetic data generation. Following [25], we first invoke the marginal selection algorithm in PrivBayes to identify a set of marginals and inject noise into them, and then apply PGM on the noisy marginals to generate synthetic data. We refer to this combination of PrivBayes and PGM as PB-PGM. For both PrivBayes and PB-PGM, we follow the setting in [38] to allocate 30% of the privacy budget to marginal selection.

PrivBayes, PB-PGM, and DP-Copula all satisfy ϵ-DP. In contrast, PrivMRF, BSG, and DP-WGAN are designed for (ϵ, δ)-DP. We set $\delta = 10^{-5}$, which is a small value commonly used in previous work (*e.g.*, [1]). The other parameters of PrivMRF are set in accordance with our discussion in Section 3.6, unless otherwise specified.

5.2 Experimental Results

Parameter tuning for PrivMRF**.** In our first set of experiments, we evaluate the performance of PrivMRF on NLTCS while varying its three internal parameters: (i) the constant θ in Algorithm 2 to decide whether a maringal is θ-useful; (ii) the number t of iterations in Algorithm 4; (iii) the size k of the random marginal set $\mathcal{U}'$ used in each iteration of Algorithm 4.

Figure 1a shows the average total variation distance (TVD) of the 5-way marginals generated from PrivMRF's output, when θ and ϵ vary. Observe that a small θ leads to relatively large TVD when ϵ is large, while a large θ results in considerably increased TVD when ϵ is small. Setting $\theta = 6$ yields balanced performance for all tested values of ϵ. Meanwhile, Figure 1b illustrates the mis-classification rate of the SVM model trained on PrivMRF's output, with various θ and ϵ. Again, $\theta = 6$ provides more balanced performance for all ϵ, when compared with larger or smaller θ. Therefore, we set $\theta = 6$ as the default for PrivMRF.

Figure 2 illustrates the performance of PrivMRF for 5-way marginal queries and SVM, when t varies. Recall that (i) t equals the number of marginals that PrivMRF inserts into the initial marginal set $\mathcal{M}$ generated by Algorithm 3, and (ii) the selection of those t marginals is allocated 10% of the total privacy budget. When t is small, PrivMRF can only add a small number of marginals into the initial $\mathcal{M}$, which compromises its ability to accurately model the

[1]Note that the original PrivBayes in [38] considers an alternative definition of neighboring datasets: two datasets are neighboring if they *have the same size but differ in exactly one tuple*. In our implementation of PrivBayes, we adopt the notion of neighboring datasets in Definition 1 and revise the algorithm accordingly, so that we can have a a fair comparison between PrivBayes and PrivMRF. As a consequence, the performance of PrivBayes in our experiments is generally better than that in the experiments of [38].

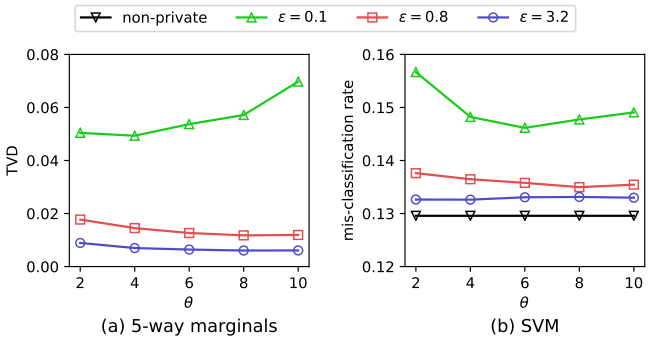

Figure 1: Performance of PrivMRF on NLTCS vs. θ

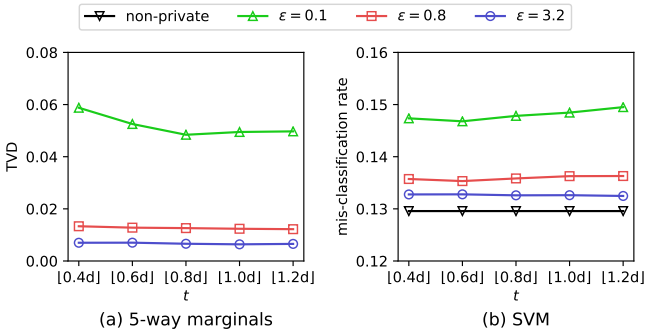

Figure 2: Performance of PrivMRF on NLTCS vs. t

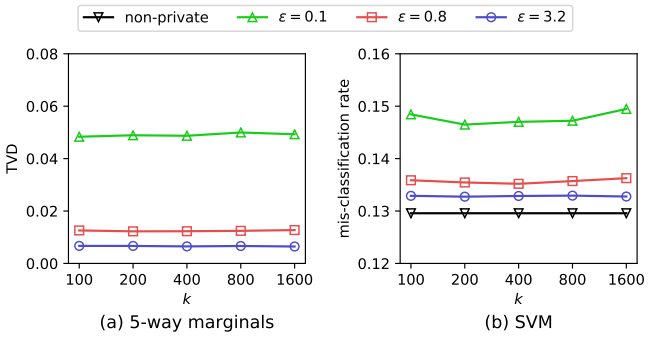

Figure 3: Performance of PrivMRF on NLTCS vs. k

input data. On the other hand, when t is large, the selection of each marginal can only consume a small share of the privacy budget, in which case the selection would be perturbed with considerable noise and become error-prone. This explains why $t = \lfloor 0.4d \rfloor$ and $t = \lfloor 1.2d \rfloor$ yield inferior overall results in Figure 2. Based on these results, we set $t = \lfloor 0.8d \rfloor$ as the default for PrivMRF.

Figure 3 plots the 5-way marginal TVD and SVM mis-classification rate of PrivMRF when k varies. Observe that the overall performance of PrivMRF is optimized when k is not too large or too small. This is because when k is small, the random marginal set $\mathcal{U}'$ by PrivMRF may not contain any useful marginals, in which case PrivMRF is unable to substantially improve $\mathcal{M}$ by inserting a marginal from $\mathcal{U}'$. On the other hand, when k is large, the selection of the "best" marginal from $\mathcal{U}'$ would be plagued by noise, since (i) we need to inject noise into the quality score $\tilde{h}(M)$ of each marginal, and (ii) the privacy budget for each noise injection decreases as k increases. Based on these observations, we set $k = 400$ as the default for PrivMRF.

Note that the above tuning of parameters is only based on NLTCS, without inspecting the other three datasets. Therefore, our choices of parameters do not reveal any sensitive information from the other datasets, nor do they unfairly favor PrivMRF on those datasets.

Comparison on α-way marginals. Figure 4 compares all methods in terms of the average TVD of the α-way marginals derived from the synthetic data. Observe that PrivMRF consistently outperforms all competing methods by a large margin, regardless of ϵ, α, or the dataset used. This demonstrates that the noisy MRF constructed by PrivMRF is able to more accurately model the input data than the other methods.

Meanwhile, PB-PGM incurs a lower TVD than PrivBayes does, which is consistent with the experimental results in [25]. In turn, PrivBayes consistently outperforms BSG, DP-Copula, and DP-WGAN. The TVD of DP-Copula is larger than that of BSG on NLTCS and ACS in most settings, but is slightly better than the latter on Adult and BR2000. Meanwhile, the TVD of DP-WGAN is significantly larger than those of the other methods. In particular, we omit DP-WGAN from Figures 4(a), 4(f), and 4(j), since its TVD on NLTCS is off the scale when compared to other methods.

Comparison on SVM classification. Figure 5 illustrates the average mis-classification rates of the SVM classifiers built on the synthetic data generated by the DP methods, as well as that of a non-private SVM classifier trained on the input data. When $\epsilon = 3.2$, the mis-classification rates of PrivMRF, PrivBayes, and PB-PGM are similar, since a large privacy budget makes it easier to model important correlations among the attributes in the input data. For the cases when $\epsilon < 3.2$, however, PrivMRF provides much higher classification accuracy than PrivBayes and PB-PGM do, which is consistent with the results in Figure 4.

The performance PB-PGM and PrivBayes are comparable in almost all cases, with PB-PGM having a slight edge on NLTCS and ACS. This indicates that the Bayesian network constructed by PrivBayes is relatively effective for SVM classifiers, which leaves little room for improvement by combining PGM with PrivBayes.

On the other hand, the mis-classification rates of BSG is considerably higher than those of PrivMRF, PrivBayes, and PB-PGM in all cases. In turn, BSG consistently outperforms DP-Copula and DP-WGAN. Note that we omit DP-WGAN in Figure 5(a), since its mis-classification rate on NLTCS is excessively large when compared with other methods.

Computation cost of PrivMRF. Figure 6 shows the running time of PrivMRF as a function of ϵ. Generally speaking, the computation time of PrivMRF tends to increase when ϵ is large. To explain, observe that a large ϵ results in a smaller amount of noise in each marginal cell, and hence, there would be a larger number of marginals satisfying θ-usefulness. This increases the size of candidate marginal set $\mathcal{U}$ considered by PrivMRF, thus incurring a high computation cost in general. Nevertheless, the computation time of PrivMRF is not strictly monotone with respect to ϵ, and the reason is as follows. When ϵ varies, the MRF constructed by PrivMRF also varies. In some cases, a small ϵ may lead to an MRF

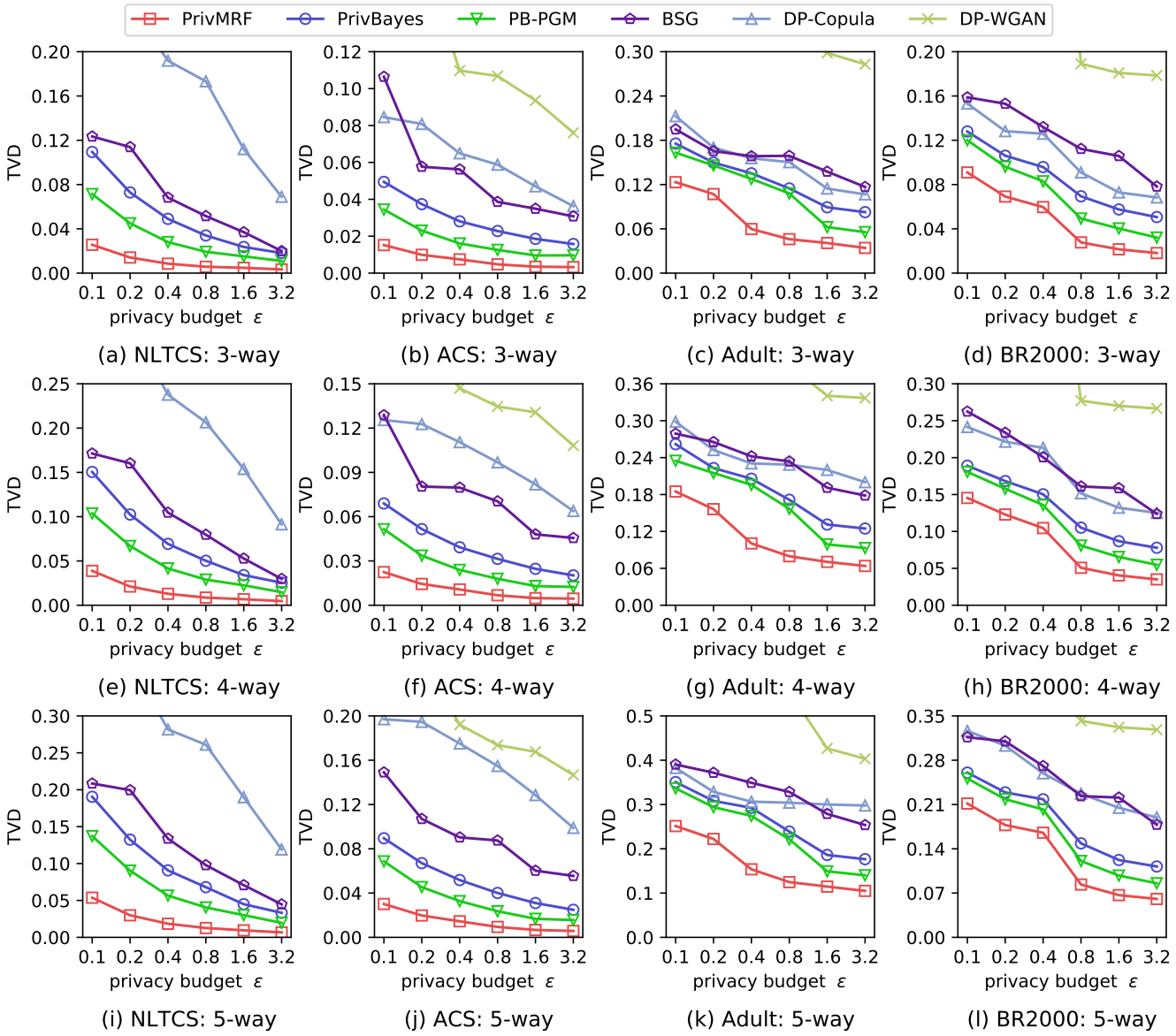

Figure 4: Accuracy of α-way marginals vs. ϵ.

whose parameter vector takes a longer time to learn using Algorithm 5. Therefore, the running time of PrivMRF does not always increase when ϵ increases. In all cases tested, PrivMRF takes at most 17 minutes to generate a synthetic dataset.

Summary. In summary, our experimental results show that PrivMRF significantly outperforms competing methods in terms of the quality of the synthetic data, and its computation cost is reasonable when using GPUs. Therefore, PrivMRF is a preferable method for data synthesis under (ϵ, δ)-DP. On the other hand, if the user requires using ϵ-DP instead of (ϵ, δ)-DP, then PB-PGM is preferred, since it offers the best data utility among all of the ϵ-DP algorithms tested in our experiments.

6 CONCLUSION

This paper presents PrivMRF, a new algorithm for synthesizing high-dimensional data with (ϵ, δ)-differential privacy. The basic idea of PrivMRF is to select an appropriate set $\mathcal{M}$ of marginals of the input data, and then construct a Markov random field (MRF) based on $\mathcal{M}$ for synthetic data generation. Experimental results on benchmark datasets demonstrate that PrivMRF consistently outperforms the state of the art in terms of the accuracy of counting queries and classification tasks conducted on the synthetic data generated. For future work, we plan to investigate how we may extend PrivMRF to synthesize other types of data, *e.g.*, trajectories.

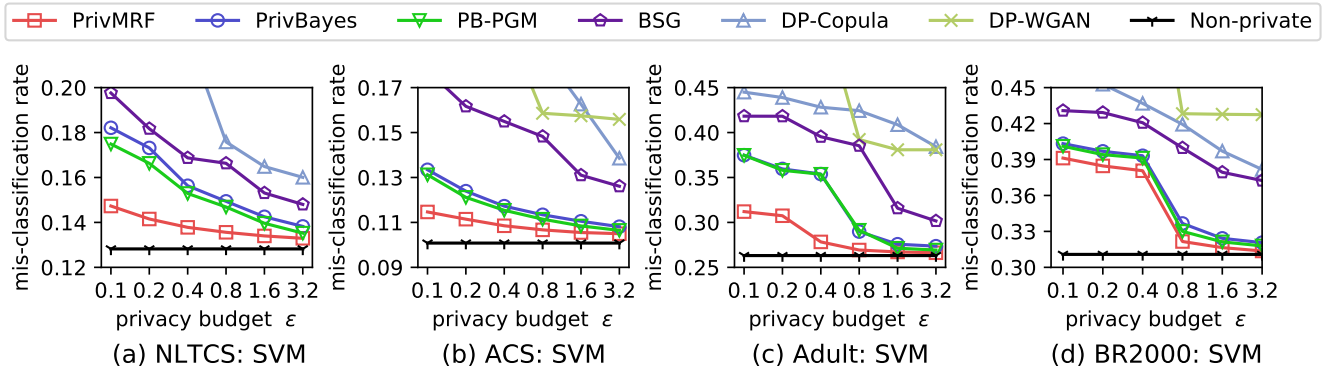

Figure 5: Mis-classification rates of different algorithms vs. ϵ

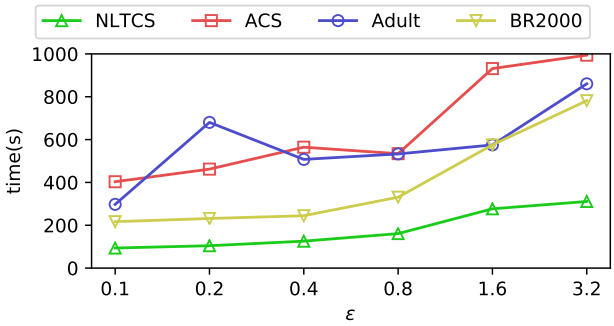

Figure 6: Computation cost of PrivMRF vs. ϵ.

ACKNOWLEDGMENTS

This research is supported by the Ministry of Education, Singapore under Grant 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

[1] Martín Abadi, Andy Chu, Ian J. Goodfellow, H. Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. 2016. Deep Learning with Differential Privacy. In *CCS*. 308–318.
[2] Borja Balle and Yu-Xiang Wang. 2018. Improving the Gaussian Mechanism for Differential Privacy Analytical Calibration and Optimal Denoising. In *ICML*. 403–412.
[3] Boaz Barak, Kamalika Chaudhuri, Cynthia Dwork, Satyen Kale, Frank McSherry, and Kunal Talwar. 2007. Privacy, accuracy, and consistency too: a holistic solution to contingency table release. In *PODS*. 273–282.
[4] Garrett Bernstein, Ryan McKenna, Tao Sun, Daniel Sheldon, Michael Hay, and Gerome Miklau. 2017. Differentially Private Learning of Undirected Graphical Models Using Collective Graphical Models. In *ICML*. 478–487.
[5] Vincent Bindschaedler, Reza Shokri, and Carl A. Gunter. 2017. Plausible Deniability for Privacy-Preserving Data Synthesis. *PVLDB* 10, 5 (2017), 481–492.
[6] Rui Chen, Qian Xiao, Yu Zhang, and Jianliang Xu. 2015. Differentially private high-dimensional data publication via sampling-based inference. In *SIGKDD*. 129–138.
[7] Graham Cormode, Cecilia M. Procopiuc, Divesh Srivastava, and Thanh T. L. Tran. 2012. Differentially private summaries for sparse data. In *ICDT*. 299–311.
[8] Bolin Ding, Janardhan Kulkarni, and Sergey Yekhanin. 2017. Collecting telemetry data privately. In *NeurIPS*. 3571–3580.
[9] Dheeru Dua and Casey Graff. 2017. UCI Machine Learning Repository. University of California, Irvine, School of Information and Computer Sciences.
[10] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. 2006. Calibrating noise to sensitivity in private data analysis. In *TCC*. 265–284.
[11] Cynthia Dwork, Aaron Roth, et al. 2014. The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science* 9, 3-4 (2014), 211–407.
[12] Úlfar Erlingsson, Vasyl Pihur, and Aleksandra Korolova. 2014. Rappor: Randomized aggregatable privacy-preserving ordinal response. In *CCS*. 1054–1067.
[13] Marco Gaboardi, Emilio Jesús Gallego Arias, Justin Hsu, Aaron Roth, and Zhiwei Steven Wu. 2014. Dual query: Practical private query release for high dimensional data. In *ICML*. 1170–1178.
[14] Mark A. Hall. 2000. Correlation-based Feature Selection for Discrete and Numeric Class Machine Learning. In *ICML*. 359–366.
[15] Linus Hamilton, Frederic Koehler, and Ankur Moitra. 2017. Information theoretic properties of Markov random fields, and their algorithmic applications. In *Advances in Neural Information Processing Systems*. 2463–2472.
[16] Moritz Hardt, Katrina Ligett, and Frank McSherry. 2012. A simple and practical algorithm for differentially private data release. In *NeurIPS*. 2339–2347.
[17] Trevor Hastie, Robert Tibshirani, and Martin Wainwright. 2015. *Statistical learning with sparsity: the lasso and generalizations*. CRC press.
[18] Michael Hay, Vibhor Rastogi, Gerome Miklau, and Dan Suciu. 2010. Boosting the Accuracy of Differentially Private Histograms Through Consistency. *PVLDB* 3, 1 (2010), 1021–1032.
[19] James Jordon, Jinsung Yoon, and Mihaela van der Schaar. 2019. PATE-GAN: Generating Synthetic Data with Differential Privacy Guarantees. In *ICLR*.
[20] G. Manton Kenneth. 2010. National Long-Term Care Survey: 1982, 1984, 1989, 1994, 1999, and 2004. Inter-university Consortium for Political and Social Research.
[21] Daphne Koller and Nir Friedman. 2009. *Probabilistic graphical models: principles and techniques*. MIT press.
[22] Haoran Li, Li Xiong, and Xiaoqian Jiang. 2014. Differentially Private Synthesization of Multi-Dimensional Data using Copula Functions. In *EDBT*. 475–486.
[23] Andrey Y Lokhov, Marc Vuffray, Sidhant Misra, and Michael Chertkov. 2018. Optimal structure and parameter learning of Ising models. *Science advances* 4, 3 (2018), e1700791.
[24] Ryan McKenna, Gerome Miklau, Michael Hay, and Ashwin Machanavajjhala. 2018. Optimizing error of high-dimensional statistical queries under differential privacy. *PVLDB* 11, 10 (2018), 1206–1219.
[25] Ryan McKenna, Daniel Sheldon, and Gerome Miklau. 2019. Graphical-model based estimation and inference for differential privacy. In *ICML*. 4435–4444.
[26] Implementation of DP-Copula. [n.d.]. https://github.com/thierryr/dpcopula_kendall.
[27] Implementation of PGM. [n.d.]. https://github.com/ryan112358/private-pgm.
[28] Technical report. [n.d.]. https://drive.google.com/file/d/17bJohzhPevbnclmuheiHSok_OZm4Y7go/view?usp=sharing.
[29] Steven Ruggles, Katie Genadek, Ronald Goeken, Josiah Grover, and Matthew Sobek. 2015. IPUMS USA: VERSION 6.0. Minneapolis: University of Minnesota.
[30] Mani Srivastava and Moustafa Alzantot. 2019. Differentially Private Dataset Release using Wasserstein GANs. https://github.com/nesl/nist_differential_privacy_synthetic_data_challenge.
[31] Jun Tang, Aleksandra Korolova, Xiaolong Bai, Xueqiang Wang, and XiaoFeng Wang. 2017. Privacy Loss in Apple's Implementation of Differential Privacy on MacOS 10.12. *CoRR* abs/1709.02753 (2017).

[32] Marshall F Tappen. 2007. Utilizing variational optimization to learn markov random fields. In *2007 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 1–8.
[33] Synthetic Data Vault. [n.d.]. https://sdv.dev/.
[34] Martin J Wainwright and Michael Irwin Jordan. 2008. *Graphical models, exponential families, and variational inference*. Now Publishers Inc.
[35] Xiaokui Xiao, Guozhang Wang, and Johannes Gehrke. 2010. Differential privacy via wavelet transforms. In *ICDE*. 225–236.
[36] Huanyu Zhang, Gautam Kamath, Janardhan Kulkarni, and Zhiwei Steven Wu. 2020. Privately Learning Markov Random Fields. In *ICML*. 11129–11140.
[37] Jun Zhang, Graham Cormode, Cecilia M. Procopiuc, Divesh Srivastava, and Xiaokui Xiao. 2014. PrivBayes: private data release via bayesian networks. In *SIGMOD*. 1423–1434.
[38] Jun Zhang, Graham Cormode, Cecilia M Procopiuc, Divesh Srivastava, and Xiaokui Xiao. 2017. Privbayes: Private data release via bayesian networks. *TODS* 42, 4 (2017), 1–41.
[39] Jun Zhang, Xiaokui Xiao, and Xing Xie. 2016. Privtree: A differentially private algorithm for hierarchical decompositions. In *SIGMOD*. 155–170.
[40] Jun Zhu, Ni Lao, and Eric P Xing. 2010. Grafting-light: fast, incremental feature selection and structure learning of Markov random fields. In *SIGKDD*. 303–312.